Learning Sensory Representations with Minimal Supervision

Citation for published version (APA):

Document status and date:
Published: 24/06/2021

Document Version:
Other version

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 07. Feb. 2022
Learning Sensory Representations with Minimal Supervision

THESIS

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op donderdag 24 juni 2021 om 13:30 uur

door

Aaqib Saeed

egenren te Hyderabad, Pakistan
Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter: prof.dr. E.R. van den Heuvel  
1e promotor: prof.dr. J.J. Lukkien  
2e promotor: dr. T. Ozcelebi

Promotiecommissieleden: prof.dr. C. Mascolo (University of Cambridge)  
prof.dr. F. Kawsar (Technische Universiteit Delft, Nokia Bell Labs)  
prof.dr. F.D. Salim (RMIT University)  
prof.dr. M. Pechenizkiy  
dr. M. Funk

Het onderzoek of ontwerp dat in dit thesis wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.
Learning Sensory Representations with Minimal Supervision

by Aaqib Saeed
A catalogue record is available from the Eindhoven University of Technology.

Keywords: deep learning, ubiquitous computing, self-supervised learning, low-data regimes, audio recognition, sensors, time-series, multi-task learning, internet-of-things, federated learning

The work described in this thesis has been primarily carried out in the Interconnected Resource-aware Intelligent Systems (IRIS) group within the Department of Mathematics and Computer Science of the Eindhoven University of Technology and in part during internships at Google Research within Brain and Cerebra. The research carried out at the Eindhoven University of Technology was funded by SCOTT (www.scott-project.eu) project. It received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No. 737422. This Joint Undertaking received support from the European Union’s Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway.

Copyright © 2021 by Aaqib Saeed, Eindhoven, the Netherlands.
Typeset using \LaTeX
Cover background image from vecteezy.com
Printed by ADC Dereumaux, ’s-Hertogenbosch.
Summary

The ubiquity of interconnected systems has given rise to a world enriched with ambient computing where computing is ingrained in our routine such that mostly we do not realize an interaction with a computing platform. The proliferation of devices embedded with sophisticated sensors in our daily lives generates data at an unprecedented scale, providing valuable information about the environment and the people. Effectively harnessing and getting insights out of massive sensory data in a scaleable manner can unlock opportunities to provide innovative solutions to problems in various domains ranging from healthcare, human-computer interaction, wildlife monitoring, and more. Data-driven predictive models are now at the core of embedded intelligence through leveraging advances in machine learning, especially deep learning methods. These approaches utilize a massive amount of manually labeled data to learn generalizable models. Despite the fact that deep learning consistently achieves and even matches human-level performance on several tasks, deep neural networks lack the ability to learn from only a few semantically labeled examples of a concept in a way as humans learn continuously from unlabeled data (or without supervision). The requirement of providing a large amount of well-annotated data is not just difficult due to cost, time constraints, and domain expertise, but it is an unscalable path towards having intelligent computational devices that can continuously sense, learn and adapt. Similarly, another key challenge associated with a well-functioning predictive system for ambient (or pervasive) sensing is to safeguard it against catastrophic failures (e.g., due to sensor malfunctions, heterogeneous signals, domain mismatch, interpersonal variations, and more) in a real-world environment. Despite the growing body of literature to address the topic of learning robust and generalizable representations from multi-sensor data in a label-efficient manner, several challenges have yet to be overcome to achieve effective generalization.

In this thesis titled, Learning Sensory Representations with Minimal Supervision, we introduce novel techniques that lie on the intersection of deep learning, ambient sensing, and ubiquitous computing to address issues pertaining to learning from unlabeled data and making models robust to various input artifacts. Our focus is on representation learning with deep neural networks to realize the vision of self-learning for embedded intelligence in everyday devices, such as smartphones, wearables, earables, and more. Our proposed methods are primarily based on the theme of self-supervised learning to extract generic representations from multi-modal sensory inputs, such as electroencephalogram, audio, accelerometer and more. We present learning approaches that do not require semantic labels from humans but extract supervisory signals from the input itself, i.e., in a self-supervised manner. Our strate-
gies enable deep neural networks to learn broadly useful representations that perform well on a spectrum of downstream tasks, are robust to noise and other artifacts and generalize also when transferred to other domains. The developed techniques can also harness massive unlabeled data to reduce the requirement of semantic labeling, effectively use multi-modal signals, exploit continuously growing decentralized (on-device) data in a federated setting, and leverage multi-task learning to utilize shared structure among tasks.

We make several contributions in the thesis towards learning general-purpose and robust deep neural networks with minimal supervision. First, we study self-supervised learning approaches in the initial chapters to develop alternative strategies to supervising deep models than using semantic labeling. We propose several auxiliary tasks to learn representations from a wide variety of sensory data without any human involvement in the annotation process. In particular, we introduce transformation recognition, feature prediction from a masked window, blend detection, scalogram-signal correspondence learning, contrastive learning for audio, and many other self-supervised methods to pretrain deep neural networks with large-scale unlabeled data. We demonstrate state-of-the-art performance compared with supervised methods in low-data regimes and transfer learning settings on a wide-range of tasks. Second, we show that self-supervision can be effectively leveraged in federated learning to harness unlabeled decentralized data residing on users’ devices without aggregating it on a central server. Third, we focus on techniques to make deep neural networks robust to input artifacts and other forms of noises to have graceful degradation of predictive performance. We present a novel attention-based learning approach to map inputs with inconsistent channels to a fixed canonical order to create models that are invariant to channel ordering. Likewise, we introduce an adversarial autoencoder-based technique to handle multiple missing sensory modalities at inference time with negligible to no loss in performance. We also provide an extension of the adversarial model for generating class-conditional synthetic data, which can be used for data augmentation and other purposes. Fourth, we propose a subjects-as-tasks strategy to personalize deep models with multi-task learning, i.e., instead of learning different tasks, we learn the same task with a distinct set of layers in the model focusing on particular individuals and sharing layers across subjects. We also provide a simple yet effective approach for unsupervised domain adaptation based on joint input-reconstruction and task-specific loss functions. Finally, we present a multi-task and multi-modal network for learning representations from cross-domain data (i.e., from different input modalities, sensors, users, data collection protocols, and tasks) in both supervised and self-supervised manners. The proposed approach is effective in training a unified model across multiple tasks. It achieves similar performance as individual task-specific models but with better parameter utilization and exploitation of shared structure among tasks.
List of Publications

The work in this thesis is based on the following articles:


The key ideas, implementation, experiments, and text originate from the work of the first author except for the paper Learning from Heterogeneous EEG Signals with Differentiable Channel
Reordering, where the last author coined the central idea. All other authors had advisory roles and/or contributed to writing few individual sections of the papers listed above.

The author has also contributed to the following publications and preprints that go beyond the scope of this thesis:


Contents

Summary v
List of Publications vii

1 Introduction 1
  1.1 Sensing, Deep Learning and Challenges .............................. 1
  1.2 Towards Self-Learning Systems for Embedded Intelligence .......... 3
  1.3 Objectives and Research Questions ................................... 5
  1.4 Thesis Organization .................................................... 8

2 Background 11
  2.1 Notation ..................................................................... 11
  2.2 Neural Networks and Representations ................................ 12
  2.3 Multi-task Learning ...................................................... 14
  2.4 Knowledge Transfer ..................................................... 15
  2.5 Self-Supervision ........................................................... 16
  2.6 Modeling Signals and Time-Series ................................... 17

3 Self-Supervised Learning with Transformation Prediction 19
  3.1 Introduction ............................................................... 19
  3.2 Approach ................................................................. 21
    3.2.1 Overview ................................................................ 21
    3.2.2 Self-Supervised Task: Signal Transformation ................. 22
    3.2.3 Network Architecture and Implementation .................... 24
  3.3 Experiments ............................................................... 25
    3.3.1 Datasets ............................................................... 26
    3.3.2 Pre-Processing and Assessment Strategy ....................... 27
    3.3.3 Results ................................................................ 28
  3.4 Related Work .............................................................. 38
  3.5 Conclusion ................................................................. 41

4 Sense and Learn: Self-Supervision for Omnipresent Sensors 45
  4.1 Introduction ............................................................... 45
  4.2 Approach ................................................................. 48
    4.2.1 Motivation and Overview ......................................... 48
    4.2.2 Suite of Pretext Tasks .............................................. 49
    4.2.3 Network Architecture Design .................................... 52
  4.3 Experiments ............................................................... 54
### 4.3.1 Datasets

54

### 4.3.2 Pre-processing and Evaluation

56

### 4.3.3 Results and Discussion

58

### 4.3.4 Impact and Limitations

68

### 4.4 Related Work

70

### 4.5 Conclusion

72

### 5. Federated Self-Supervised Learning of Multi-Sensor Representations

75

#### 5.1 Introduction

75

#### 5.2 Background

76

##### 5.2.1 Federated Learning

77

##### 5.2.2 Wavelet Transform

78

#### 5.3 Approach

79

##### 5.3.1 Scalogram-Signal Correspondence Learning

79

##### 5.3.2 Network Architecture

81

##### 5.3.3 Implementation Details

82

#### 5.4 Experiments

83

##### 5.4.1 Datasets and Preprocessing

83

##### 5.4.2 Results

84

#### 5.5 Conclusion

88

### 6. Contrastive Learning of General-Purpose Audio Representations

89

#### 6.1 Introduction

89

#### 6.2 Approach

90

#### 6.3 Experiments

92

##### 6.3.1 Datasets and Tasks

92

##### 6.3.2 Model Architecture and Implementation Details

94

##### 6.3.3 Results

94

#### 6.4 Conclusion

95

### 7. Differentiable Channel Reordering for Heterogeneous Signals

97

#### 7.1 Introduction

97

#### 7.2 Approach

98

##### 7.2.1 Learnable Channel Remapping

99

##### 7.2.2 CMSAugment: Shuffling and Masking Channels

101

##### 7.2.3 Network Architecture Design and Implementation

101

#### 7.3 Experiments

102

##### 7.3.1 Datasets

102

##### 7.3.2 Generalizing to Shuffled and Masked Channels

103

##### 7.3.3 Performance in Structured Masking Conditions

103

##### 7.3.4 Transfer Learning

103

#### 7.4 Conclusion

104

### 8. Synthesizing and Reconstructing Missing Sensory Modalities

105

#### 8.1 Introduction

105

#### 8.2 Approach

107

##### 8.2.1 Autoencoder

107

##### 8.2.2 Adversarial Autoencoder

108
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.3</td>
<td>Context Classification</td>
<td>109</td>
</tr>
<tr>
<td>8.2.4</td>
<td>Model Architecture and Training</td>
<td>110</td>
</tr>
<tr>
<td>8.2.5</td>
<td>Implementation Details</td>
<td>112</td>
</tr>
<tr>
<td>8.3</td>
<td>Experiments</td>
<td>112</td>
</tr>
<tr>
<td>8.3.1</td>
<td>ExtraSensory Dataset</td>
<td>112</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Performance Evaluation</td>
<td>113</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Results</td>
<td>114</td>
</tr>
<tr>
<td>8.4</td>
<td>Related Work</td>
<td>121</td>
</tr>
<tr>
<td>8.5</td>
<td>Conclusion</td>
<td>124</td>
</tr>
<tr>
<td>9</td>
<td>Model Adaptation and Personalization</td>
<td>127</td>
</tr>
<tr>
<td>9.1</td>
<td>Introduction</td>
<td>127</td>
</tr>
<tr>
<td>9.2</td>
<td>Approach</td>
<td>129</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Problem Definition</td>
<td>129</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Unsupervised Model Adaptation</td>
<td>130</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Personalization</td>
<td>131</td>
</tr>
<tr>
<td>9.3</td>
<td>Dataset and Pre-processing</td>
<td>133</td>
</tr>
<tr>
<td>9.4</td>
<td>Experiments</td>
<td>135</td>
</tr>
<tr>
<td>9.5</td>
<td>Conclusion</td>
<td>137</td>
</tr>
<tr>
<td>10</td>
<td>Unified Model for Cross-Domain Sensing Tasks</td>
<td>139</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td>139</td>
</tr>
<tr>
<td>10.2</td>
<td>Approach</td>
<td>141</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Problem Formulation</td>
<td>141</td>
</tr>
<tr>
<td>10.2.2</td>
<td>Unified Sensor-based Multi-task, Multi-modal Learning</td>
<td>142</td>
</tr>
<tr>
<td>10.2.3</td>
<td>Multi-task Contrastive Predictive Coding</td>
<td>143</td>
</tr>
<tr>
<td>10.2.4</td>
<td>Semi-Supervised Learning for Sensing</td>
<td>144</td>
</tr>
<tr>
<td>10.3</td>
<td>Experiments</td>
<td>144</td>
</tr>
<tr>
<td>10.3.1</td>
<td>Tasks and Datasets</td>
<td>145</td>
</tr>
<tr>
<td>10.3.2</td>
<td>Implementation Details</td>
<td>147</td>
</tr>
<tr>
<td>10.3.3</td>
<td>Results</td>
<td>148</td>
</tr>
<tr>
<td>10.4</td>
<td>Related Work</td>
<td>151</td>
</tr>
<tr>
<td>10.5</td>
<td>Conclusion</td>
<td>152</td>
</tr>
<tr>
<td>11</td>
<td>Conclusion</td>
<td>155</td>
</tr>
</tbody>
</table>

Bibliography  
Acknowledgements  
Curriculum Vitae
Chapter 1

Introduction

1.1 Sensing, Deep Learning and Challenges

Our ability to see, hear, and sense the environment provides us with remarkable prior knowledge to learn, reason, and adapt in an ever-changing world. It also equips us with situational awareness to better understand each other (e.g., through behavioral cues), places, things and whereabouts for making sound decisions [1]. One of the key characteristics of human cognition is the ability to learn from the world around us without explicit supervision [2]. Developing systems that can foster such capabilities in computational devices to recognize context, surroundings, human behavior, and other states of interest is a grand challenge in artificial intelligence. Similarly, the objective of making (networked) devices intelligent has driven significant research in the sensing realm, particularly in pervasive sensing, ubiquitous computing, and human-computer interaction [3, 4, 5, 6, 7, 8, 9]. It also led to the development of tiny and low-cost sensors with negligible power requirements, which can be reliably embedded in devices of general use, such as wearables, earables, smartphones, and others [5, 10].

Nowadays, contemporary devices are equipped with a multitude of sensors to capture various physical phenomena and networking capabilities to significantly expand devices’ potential for unobtrusive, passive, and contactless (or contact-based) multi-modal data collection. Likewise, the ubiquity of interconnected systems has given rise to a world enriched with ambient (ubiquitous) computing where computing is ingrained in our routine such that mostly we do not realize an interaction with a computing platform. The proliferation of such devices in our daily lives generates data at an unprecedented scale, providing valuable information about the environment and the people. Effectively harnessing and getting insights out of massive sensory data in a scaleable manner is vital for recognizing certain events of interest and taking appropriate actions on them. To this end, we need methods that can endow intelligence in existing networked devices to address challenging problems in various domains and help in discovering opportunities for novel applications e.g., in life-logging, healthcare, safety, wildlife surveillance, context-awareness, and more.
The advancements in wireless communication, embedded systems, artificial intelligence, and human-computer interaction realize the vision of the Internet of Things (IoT), revolutionizing several fields ranging from healthcare, manufacturing to agriculture\textsuperscript{[11]}. IoT devices with the capability to sense, compute and communicate over the Internet enable a broad set of embedded intelligence applications, e.g., in consumer and industrial devices. At the core of making distributed devices smart are the advances in machine learning, especially deep learning. These advancements enable contemporary devices to understand their environment, recognize events of interest and act on them by making optimal decisions. With the development of these methods to learn predictive models using a variety of input modalities, a network of the ambient and personal devices becomes capable of performing complex sensing and recognition tasks.

The work in this thesis is grounded in machine learning for ubiquitous computing and ambient (or pervasive) sensing. The former is a sub-field of artificial intelligence widely studied to develop methods for recognizing patterns in a large amount of data. The latter is concerned with augmenting everyday objects or devices with computational and communication capabilities to perform useful tasks for the users. The high-level objective of machine learning is to design algorithms that can learn from data to solve a specific task without the need of being explicitly programmed to do it. It is ideal from the point that for solving many problems, humans can not explain the procedure or write a comprehensive list of rules. For instance, a clinician can examine an electroencephalogram to seemingly diagnose a brain disorder without much difficulty, but writing down the corresponding interpretation steps can be extremely daunting, if not entirely impossible. Here, learning comes into play, which mainly involves fitting a model on the observations or experiences from the past in order to learn patterns that generalize well on unseen data.

Typically, machine learning models are built on top of a set of features extracted from raw inputs, mostly determined based on domain expertise, for instance, computing basic summary statistics, number of peaks, amplitude, and kurtosis from a 1D signal (e.g., skin conductance). Similarly, for speech analysis, Mel-frequency cepstral coefficients is another prime example of a feature extracted from audio waveforms. The features provide discriminative information about each example that a learning algorithm leverages to differentiate between instances, as in classification problems, where data instances, for example audio clips are grouped under specific categories. The procedure of designing sophisticated feature extraction techniques or hand-crafting them is referred to as feature engineering. Nevertheless, it became a major bottleneck in improving the predictive performance of the models as developing useful features depended on human creativity, and classic approaches lack the power to capture underlying explanatory factors in the milieu of low-level sensory inputs.

The field of deep learning provides a set of methods to overcome the limitations of prior approaches and automate the discovery of disentangled features through jointly learning features (or representations) along with the predictive model in an end-to-end manner\textsuperscript{[12]}. The key building block is a deep neural network, which is a composition of multiple parameterized non-linear transformations stacked together to form a model. We learn parameters (or weights) of the model through feeding raw input data to optimize an objective function using gradient descent techniques. This process yields useful representations for solving complex tasks without much human effort. Specifically, deep learning has achieved indisputable em-
pirical success across a broad spectrum of problems over the last decade, such as in computer vision, speech recognition, medical diagnosis, and chemical discovery. Likewise, the success of deep learning has driven a large body of research in the sensing domain to address important problems, such as arrhythmia detection and analysis of electroencephalogram for seizure recognition.

Behind the success of existing approaches are a) better inductive biases that prominently come in the form of neural network architectures, training objectives, data augmentation, and regularization strategies, b) increased availability of computation power, and c) datasets of well-annotated samples. In spite of the fact that deep learning consistently achieves and even matches the human-level performance on many tasks, deep neural networks lack the ability to learn when there are only few semantically labeled examples available of a concept in a way as humans learn continuously from unlabeled data (or without supervision) and can also efficiently adapt to changing environments. Providing a massive amount of semantically annotated input is not just difficult but an unscalable path towards having intelligent computational devices. Particularly, strong supervision that necessitates annotation of instances with one or more labels to sufficiently describe the example for the deep neural networks is hard due to following issues a) label granularity—it is unclear how coarse or fine-grained the labels should be—, b) task definition—if we want to use a model for many prediction problems, the task selection is ambiguous—, c) expensive—the process of acquiring annotations for certain important problems could be prohibitively costly and time-consuming—, d) privacy—the labeling of certain data can not be done due to privacy and safety concerns—, e) expert-level knowledge—in some domains the labels can only be provided by the expert through close inspection of the input—, and f) decentralized nature—data on edge or distributed devices like smartphones or other IoT devices can not be readily aggregated and labeled.

These observations motivate the introduction and design of techniques for training deep neural networks from raw sensory data without explicit and semantic supervision; while being robust to noise and other artifacts. We argue that one promising way of achieving this is to exploit intrinsic supervision from the input itself to continuously learn and build a generalized repertoire of high-level concepts of the modalities and underlying phenomena. Then a model can utilize the obtained knowledge to solve important tasks and develop applications ranging from activity recognition, sleep stage scoring, WiFi sensing, physiological stress detection, audio understanding, and more. This argument leads to a class of methods for learning models named self-supervision and is a central theme of research in the thesis, in addition to techniques that safe-guard deep models against input artifacts.

1.2 Towards Self-Learning Systems for Embedded Intelligence

The key factor in the success of predictive systems based on modern deep neural networks is largely attributed to the supervised learning paradigm. As described in the aforementioned section, these methods utilize a large amount of carefully curated examples to learn generalizable specialist models of tasks. Briefly, to develop a classical supervised model in the sensing realm, such as for smartphone-based context recognition, there are several steps involved in research and development. Firstly, the data are acquired from the sensors embedded in
IoT devices (e.g., smartphones and wearables) with a focus on solving a particular problem. Secondly, the data are pre-processed and manually labeled by human annotators for the process of designing, learning, and evaluating deep neural network architectures. Finally, after multiple iterations of the train-test phase, the optimal model is prepared for inference mode and deployed to make predictions on the incoming stream of data. Along these lines, significant research efforts have been made in the last few years to design effective task-specific architectures, improving efficiency, reducing model size for resource-constrained devices, and developing better tools and frameworks for on-device inference. Although devices equipped with deep models can now actively sense and detect events of interest, the existing approaches do not address issues relating to robustness, decentralized learning, privacy, personalization, and exploiting unlabeled data. Furthermore, the model stays fixed, the devices have no opportunity for learning on their own, and in case the model needs to be updated, the entire procedure has to be followed again as defined earlier. These issues impose severe limitations on learning models with supervised learning and restrict the ability of sensor rich devices to learn generalist models capable of performing more than one task and that can rapidly acquire new skills without huge amounts of labeled data.

Our work aims to harness the full potential of network technologies such as the IoT, which enable embedded intelligence. Towards reaching this goal, we seek to develop novel methods that can enable devices to sense, learn, and adapt the models in a self-learning manner without requiring human intervention. In particular, we design strategies that can be utilized to learn from decentralized data residing on devices without aggregating it in a central repository. Such approaches are now even more desirable as the computational power of the edge devices is growing, and multi-modal data are being generated at an unprecedented rate, which makes the accumulation and labeling extremely expensive. In this thesis, we argue for the introduction and design of techniques that exploit a massive amount of unlabeled sensory data to learn general-purpose representations and then use few-labeled examples to fine-tune a pretrained model on the task of interest. To this end, our core approach is self-supervised learning that leverages natural supervision available in the input without requiring semantic labels. It opens up exciting possibilities for IoT devices to learn continuously without focusing on a specific task and adapting the models as needed on a chosen end-task of interest to the user. For instance, a self-supervised model of audio on a device (such as a smartphone or virtual assistant device) can continuously learn from an audio stream in-the-wild without requiring any interaction with the user for labels and extract representations that it later use to solve the downstream task (e.g., keyword spotting) with few labeled audio examples.

Furthermore, there are other issues of high importance pertaining to a well functioning intelligent IoT system to avoid it from failing catastrophically. Particularly, the robustness of the models to input artifacts and sensor failures is crucial in a real-world setting. Similarly, the models might become too general or not personalized at all for a specific user; hence, fail to take into account interpersonal variations. To address these key concerns for any sensing system and have reliable models, we propose methods that tackle input artifacts and missing modalities while learning an end-task and that allow practitioners to incorporate inductive biases via prior domain knowledge. These techniques can then also be readily combined with self-supervision to enrich IoT systems and provide the basis for on-device continual learning from heterogeneous sensory inputs.
1.3 Objectives and Research Questions

This thesis studies the problem of learning generalizable representations with deep neural networks from a variety of sensory data to realize the vision of self-learning in smart devices for pervasive sensing and other domains. The key focus is on developing methods for learning deep models with raw multi-modal data subject to the constraints deriving from the IoT context: 1) utilize self-supervision for pretraining to perform well on various downstream tasks, 2) be robust to noise and other input artifacts, 3) transfer well across domains, i.e., on data from other devices, environments and users, 4) harness massive unlabeled data to reduce the requirement of semantic labeling, 5) effectively use multi-modal signals, 6) exploit continuously growing decentralized (on-device) data in a federated setting, and 7) leverage multi-task learning to utilize shared structures among tasks. The central guiding hypothesis of the thesis is the following: A deep neural network that leverages intrinsic supervision from the raw input itself to acquire ground-truth for learning, achieves performance that is better than or similar to the models relying entirely on labeled data on a broad range of tasks, domains, and modalities. In particular, we argue that self-supervision outperforms classical supervised learning given large-scale unlabeled data for pretraining and in the low-data (low-label, high-data) regime, i.e., when few labeled examples are available to learn a model in a fully-supervised manner. Furthermore, we address other important aspects to make sensing models generalize better, including data augmentation, adaptation, personalization, and avoiding catastrophic failures in case of inconsistent and missing modalities. Overall, these research themes provide a natural division of the thesis relating to self-supervised learning and devising techniques to tackle challenging issues in learning better models from sensor-based data.

The core contributions of the thesis are based on the following topics and the corresponding research questions:

**How can we utilize large-scale sensory data without semantic labels to learn high-level representations?**

**Research Question 1:** Do self-supervised pretext tasks enable learning useful representations with deep neural networks from unlabeled sensor data that are competitive with the fully-supervised counterparts?

To address the question, we introduce auxiliary tasks for pretraining deep temporal convolutional networks in Chapters 3 and in [14, 15]. Our tasks provide an effective way to utilize a large amount of unlabeled data for learning without relying on labeling processes. Notably, the transformation prediction task demonstrates significant improvement on human activity recognition problems using signals from inertial measurement units embedded in smartphones. Our devised pretext tasks within the *sense and learn* framework, especially, feature prediction from masked windows, temporal shift, and fusion magnitude estimation, provides a way to apply self-supervised learning on other signals (e.g., electrocardiogram), where using transformations may introduce unintended artifacts in the input or require careful selection of the transformations based on domain expertise. Specifically, the tasks provide a simple and inexpensive process to acquire ground-truth (or supervisory signal) from raw
input for the deep neural network, which can be of high value for learning on-device models with limited computational power. We demonstrate the usefulness of our pretext tasks on a broad range of problems (such as sleep stage scoring, stress detection, Wi-Fi sensing, and more) involving various signals, achieving generalization superior to auto-encoding methods, and competitive with fully-supervised counterparts.

**Research Question 2:** Does self-supervised pretraining improve label efficiency to achieve better generalization on downstream tasks with few-labeled examples and to induce inductive bias required for transfer across the domain?

We investigate the effectiveness of network pretraining with self-supervision to improve the semi-supervised learning capability of the models in Chapters 3 and 14, 15. We show that our novel approaches provide an effective initialization of the deep model by harnessing unlabeled data that, when fine-tuned with very few labeled examples on downstream tasks, significantly improves predictive performance compared to training from scratch. Likewise, in a real-world learning setup, there is a high chance that we are interested in a different task than the one originating from the unlabeled data (with different distribution) accessible for pretraining. For instance, the accelerometer data available for self-supervised pretraining could be acquired in-the-wild for human activity recognition but it can be used to pretraining models for user authentication or transport mode detection tasks. To study knowledge transferability, we further explore unsupervised transfer of learned representations across different tasks and datasets in Chapters 3, 4, 6, and in 14, 15, 16. We demonstrate its applications on various end-tasks, including activity recognition, audio classification, and problems related to electroencephalography. For sensor-based learning tasks, we highlight further that model transfer is beneficial for learning in the low-data regime of target tasks.

**Research Question 3:** Does learning general purpose representations from unlabeled data improve performance on various recognition tasks? This question is investigated on sound recognition tasks.

Our contribution towards addressing this question is a technique that we term contrastive learning for audio (COLA) and introduce it in Chapter 6 and in 16. Our simple, lightweight, and easy-to-implement approach enables deep models to learn representations through assigning high similarity to audio segments extracted from the same recording (or clips) while attributing lower similarity to segments from different recordings. We present pretraining with millions of unlabeled audio clips without relying on augmentation for an anchor-positive generation or using a memory bank for negative mining. We show COLA embedding significantly improves recognition rate over earlier work in unsupervised learning for audio across a broad range of classification tasks, including keyword spotting, language identification, acoustic scene detection, and more.

What are the effective ways to leverage unlabeled distributed data for training deep neural networks without aggregating it in a centralized repository?

**Research Question 4:** How to perform self-supervised federated learning to utilize decentralized unlabeled data?
We introduce scalogram-signal correspondence learning (SSCL) technique in Chapter 5 and in [17] to show learning multi-sensor representations is possible from unlabeled decentralized data residing on users’ devices without aggregating it on a central server. SSCL is a self-supervised method exploiting a multi-view strategy to extract supervisory signals using a wavelet transform [18]. It uses a contrastive objective for training a deep model to estimate if a given pair of a raw signal and its complementary view (i.e., a scalogram) align with each other or not. We demonstrate the capability of SSCL in learning useful representations across several sensing tasks, where it achieves performance on-par with centralized models.

How to avoid catastrophic failure of deep neural networks on noisy inputs with learning-based approaches?

Research Question 5: Can a learnable channel remapping be used to handle inconsistent inputs? and what is a good strategy for tackling missing input modalities for a deep model at inference-time?

To address first question, we develop channel reordering module in Chapter 7 and in [19] that we refer to as CHARM. Here, a channel refers to a single source of information about a certain phenomena. In case of an electroencephalogram, a signal acquired from a particular electrode over the scalp acts as a channel. Likewise, for microphone array an audio waveform of a single microphone can be treated as a channel. The signals from multiple sensors are combined (or stacked depthwise) to form a multi-channel input that can be used as input to deep neural networks. Our approach builds upon attention mechanisms [20] to estimate a latent reordering matrix from each input signal (or channel) and maps inconsistent input channels to canonical order. Our learnable module is differentiable and can be composed further with architectures expecting a consistent channel ordering to build end-to-end trainable classifiers without requiring prior channel location information. We also introduce channel masking and shuffling augmentation to improve the generalization of standard models to operate on inconsistent inputs. We highlight the applicability of CHARM to electroencephalography, where data collection protocols from different headsets result in varying channel ordering and number, limiting the robustness and feasibility of transferring models [21] across headsets. To answer the second question, we propose an adversarial autoencoder-based [22] technique in Chapter 8 and in [23]. Here, the model pretraining via joint input reconstruction and adversarial training strategy aims to counter missing sensory modalities in natural conditions. We also provide an extension to generate class-conditional synthetic data similar to the original examples. We demonstrate applications of our adversarial model to restore features from several modalities and generate high-quality artificial samples on a multi-label classification task of human context recognition.

How effective is a single deep neural network at learning shared representations for multiple related tasks?

Research Question 6: Is it possible to effectively leverage multi-task learning for model personalization and adaptation?
We present a *subjects-as-tasks* strategy for personalizing deep models with multi-task learning in Chapter 9 and in [24]. We substitute tasks with subjects in a standard-setting of jointly learning to solve multiple tasks, but as opposed to learning different tasks, we learn the same task with a distinct hierarchy of layers in the model focusing on a particular individual. Specifically, our model comprises a shared set of layers with hard weight-sharing to extract features common across subjects and utilize user-specific layers to learn representations that are personalized or specialized towards a particular subject. We show the effectiveness of our method on physiological stress detection, where personalization considerably improves the predictive performance of the model. Likewise, for unsupervised domain adaptation in a cross-domain and cross-user setting, we introduce a multi-stream architecture with a shared encoder between source and target domains and jointly optimize network weights to reconstruct input with a decoder on a target domain and solve a source task with a supervised bi-directional recurrent network on labeled data. Our deep reconstruction and classification model successfully adapts itself during training on data collected in a simulated (source) environment to a real-world (target) unlabeled data.

**Research Question 7:** *To what degree can a unified deep neural network learn to solve multiple tasks using multi-modal sensory data?*

This question focuses on the development of one model that has the capability to learn many tasks. In Chapter 10, we propose *unified model* or UniModel, in short, which is a deep temporal convolutional network that is capable of learning representations for several recognition problems from cross-domain data (i.e., from different input modalities, sensors, users, data collection protocols and tasks) in both supervised and unsupervised manner. We also introduce an extension of *contrastive predictive coding* [25] to a multi-task setting for self-supervised learning with diverse types of multi-modal inputs. We show that it is feasible to train a single network shared across tasks achieving a similar performance as individual task-specific models but with better parameter utilization and exploiting shared structure among tasks. It can also be trained effectively with unlabeled data to improve generalization in the low-data regime. Likewise, we demonstrate that UniModel introduces an inductive bias necessary for generalization on out-of-domain data with a transfer of the learned model to other tasks and datasets.

### 1.4 Thesis Organization

This thesis is organized around several publications, which take the form of chapters (see Figure 1.1 for a high-level overview of core research areas with a focus on corresponding chapters). We lightly edited the papers to fit the format of the thesis. We provide a brief overview of several important background topics in Chapter 3, which serves as a ground for the methods introduced in the rest of the chapters. In rest of the chapters, we present novel techniques that can be used to learn useful representations from a wide array of time-series or signals using self-supervised learning 3, 4, 5, and 10. Particularly, in Chapter 6, we provide a method for learning unsupervised sensory models in a federated setting without relying on semantic labels from humans. In Chapter 7, we propose an approach to learning general-purpose representations from the audio waveform. Furthermore, in Chapters 7 and 10, we introduce a differentiable
channel reordering module to handle inconsistent channels in an end-to-end manner without ordering information. Similarly, in Chapter 8, we provide an effective way to handle missing modalities at inference time with a generative model, which can also be utilized for generating synthetic data. Chapter 9, we present an approach for unsupervised model adaptation and personalization with multi-task learning. Chapter 10, we study the problem of learning a unified model for multiple tasks in a supervised and self-supervised manner. Lastly, Chapter 11 describes conclusions where we summarize our findings with answers to the research questions established in Chapter 1 and providing directions for future research.

Figure 1.1: Overview of research areas that are the focus of the work in this thesis. Each chapter tackles specific problems within the context of sensory representation learning and ubiquitous computing.
Chapter 2

Background

In this chapter, we provide a brief overview of key background topics, different machine learning paradigms, notation, and other necessary information that is extensively used throughout the thesis. We cover these subjects as they either serve as fundamental building blocks or benefit from our developed techniques. We present additional background material in subsequent chapters as necessary. In short, we review neural networks and representation learning in Section 2.2, leaning multiple tasks in Section 2.3, knowledge transfer in deep networks in Section 2.4, self-supervised learning or learning without strong labels in Section 2.5, and lastly, the processing of time-series or signal data collected from sensors for learning deep models in Section 2.6.

2.1 Notation

In this section, we present the most commonly utilized conventions across the thesis. We use standard lowercase weight letters $x$, $y$, or $z$ for scalar values, the lowercase bold letters $\mathbf{x}$ to denote vectors, and the bold uppercase letters $\mathbf{X}$ to represent matrices. The bold letters with subscripts $x_i$ refer to the row or an instance, whereas subscripts with standard letters $x_i$ show a specific element. Likewise, the calligraphic letters $\mathcal{Z}$ denote sets, except $\mathcal{L}$ which represents a loss or objective function. We introduce additional notation in each chapter whenever necessary.
### Notation and Meaning

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>Scalar value</td>
</tr>
<tr>
<td>$\mathbf{x}$</td>
<td>Vector</td>
</tr>
<tr>
<td>$\mathbf{X}$</td>
<td>Matrix</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$i$-th element of a vector</td>
</tr>
<tr>
<td>$X_{i,j}$</td>
<td>$i, j$-th element of a matrix</td>
</tr>
<tr>
<td>$\mathcal{M}$</td>
<td>Set of data instances</td>
</tr>
<tr>
<td>$\mathbf{W}$</td>
<td>Weight matrix</td>
</tr>
<tr>
<td>$\mathbf{b}$</td>
<td>Bias vector</td>
</tr>
<tr>
<td>$\mathcal{T}$</td>
<td>Learning task</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>Domain or dataset</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Model parameters</td>
</tr>
<tr>
<td>$\mathcal{L}$</td>
<td>Loss function</td>
</tr>
<tr>
<td>$f_\theta(\cdot)$</td>
<td>Parameterized function</td>
</tr>
</tbody>
</table>

### 2.2 Neural Networks and Representations

The invention of perceptron [26] in 1951—a simplistic pattern classification machine with a single unit—lead to modern trainable computing architecture, which becomes widely known as the deep neural network (NN). In a rudimentary form, we can think of it as a parametric function approximator with a composition of affine transformations (or functions) $f(\cdot)$. These functions are generally represented with a set of layers $l \in \{1, 2, \ldots, L\}$ each having its own learnable parameters $\theta$ and are interleaved with non-linear activation functions $\sigma$:

$$f_\theta(\cdot) = f^L \circ \sigma \circ f^{L-1} \circ \sigma \circ \ldots \circ f^1,$$

where $\circ$ denotes function composition. Here, we employ NN to learn a function $f$ from data with optimal values of the parameters w.r.t to a specific loss function. For instance, a classifier $y = f(x; \theta)$ mapping an input $x$ to a category $y$. Over the years, several variants of deep architectures have been proposed, which principally differ in their constituents to incorporate a variety of inductive biases (which broadly refers to the set of assumption a learning algorithm makes to solve a problem) depending on the input modalities with the aim of improving generalization, e.g., dropout, data augmentation, choice of non-linearity and the neural network architecture are prominent examples, see [27] for a detailed discussion.

We largely utilize contemporary deep feed-forward networks or multi-layer perceptrons [28], comprising of $L$ layers that are chained together to form a network. The overall chain length and dimensionality of the layers describe the depth and width of a model, respectively. In these networks, the information flows in a forward-only direction without feedback to itself for processing an input $x$ through computations of intermediate layers to produce an output $y$. We define a typical network $f(\cdot)$ as follows:

\[^1\text{it computes a weighted sum of its inputs}\]
\( \mathbf{z}_1 = \sigma (\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1) \)

\[\vdots\]

\( \mathbf{z}_{L-1} = \sigma (\mathbf{W}_{L-1}^\top \mathbf{z}_{L-2} + \mathbf{b}_{L-1}) \)

\( \mathbf{y} = \text{softmax}(\mathbf{W}_L^\top \mathbf{z}_{L-1} + \mathbf{b}_L) \),

where \( \mathbf{W}_l \) and \( \mathbf{b}_l \) are learnable weight matrix and bias vector, respectively. The \( \mathbf{z}_l \) refers to latent representations (or features or embeddings) in a network. For activation function \( \sigma \), we use ReLU(\( z \)) = max(0, \( z \)) or its variants, such as PReLU and ELU to enable a network to learn non-linear representations. For a classification task, the \text{softmax} operation provides distribution over \( K \) classes through normalizing penultimate layer’s output or logits as:

\[
\text{softmax}(\mathbf{z}) = \frac{\exp (z_i)}{\sum_j \exp (z_j)}
\]

The central problem in machine learning is feature extraction or as also prominently known as representation learning in the deep learning community. It describes a set of techniques for designing (or deriving) features from the input that are suitable for an application domain or learning problem. The quality of the representations, especially their discriminative power, substantially influence the generalizability of a predictive model on any task of interest. In the past, most of the efforts were spent on developing (and manually engineering) feature extraction methods based on domain expertise with the aim of incorporating prior knowledge in the learning process. However, these methods are found to be relatively limited because of their reliance on human creativity to come up with novel features and to lack the power to capture underlying explanatory factors in the domain of low-level sensory input.

The recent resurgence of deep neural networks with better architectures, improved training strategies, and availability of computing power led to the proposal of utilizing them as feature extractors. Particularly, to automate the discovery of useful features while learning to solve a task, the neural networks based approaches are widely adopted. These techniques have achieved indisputable empirical success across a broad spectrum of problems in both supervised or unsupervised learning settings. Nevertheless, representation learning still stands as a fundamental problem in machine intelligence and is an active area of research (see [29] for a detailed survey).

In this thesis, we use specialized architectures known as convolutional network [30] (CNN) to learn high-level representations directly from raw inputs. We adopt CNNs as our focus is on learning models from data with grid-like topology, e.g., multivariate time-series (1-D) or images with a 2-D grid of pixel values. These networks provide desirable properties of weight-sharing, sparse connectivity as well as efficiency in terms of training and on-device deployment through leveraging \textit{steaming} convolutions. On a high-level, the convolution operation is a central building block in a CNN, which comprises convolutional layers having several kernels (with learnable parameters) of a certain receptive field. The kernels are convolved over input and passed through a non-linear activation to produce feature maps \( \mathbf{z} \); ideally, each feature map learning something distinctive about the input. The subsequent layers use feature maps...
from preceding ones to learn even complex representations. Occasionally, a pooling layer supplements to reduce input dimensionality and to help the representations become invariant to small translation in the input. Depending on the pooling operator, it replaces values in a neighborhood with their average or maximum value. Further, we compute the output using a classification layer similar to Equation 2.2 through either flattening the representations or using a global pooling function over the penultimate layer to aggregate the features.

For training NNs, we make use of backpropagation and variants of stochastic gradient descent like Adam optimizer [31] to minimize a loss (or objective or cost) function \( L \). Particularly, in case of classification tasks, we optimize negative log-likelihood as:

\[
L_c = -\frac{1}{M} \sum_m [y_m \log(f(x_m))]
\]  

(2.4)

and for regression problems, a mean squared error as:

\[
L_r = \frac{1}{M} \sum_m (y_m - f(x_m))^2
\]  

(2.5)

over \( M \) examples comprising a dataset \( D \) with \( x_m \) and \( y_m \) being the input and class label or real-valued output, respectively. For a detailed treatment of neural networks and other key elements, especially architectures, optimization, and regularization strategies, we recommend an excellent book \textit{deep learning} by Goodfellow et al. [12].

2.3 Multi-task Learning

The field of \textit{multi-task learning} [32] (MTL) aims to enhance the learning efficiency and generalization of the model through simultaneously optimizing objectives emerging from several related tasks. Specifically, it acts as a form of regularization enforced on the model to use shared intermediate representations and exploit relations among the tasks to lead the model's parameters towards a region of values that generalize well compared to individually learning to solve each task. In addition to preventing overfitting, MTL is also beneficial for improving data efficiency in supervised models through leveraging auxiliary tasks, as in a real-world setting, we can encounter a situation where labeled data is fairly limited for some tasks. In a practical setting, several domains, including ubiquitous computing, natural language processing, computer vision, and audio recognition, adopted MTL to improve model generalization on respective applications.

Formally, given \( T \) learning tasks which could be either supervised or unsupervised along with their corresponding datasets \( D_{s,\tau} \in \{(x, y)\}^M_{m=1} \) and \( D_{u,\tau} = \{x\}^M_{m=1} \) respectively. The goal of MTL is to improve the performance of the neural network on task \( T_i \) by using knowledge from other \( T_{i-1} \) tasks. Here, the model is normally divided into distinct blocks with their associated parameters. One block comprises of task-specific parameters \( \theta_{\tau} \) that learn features important for a particular task with each task having its own set of parameters.
(or model), and these could be either in high or lower layers of the neural network. The other
block has shared parameters that are used by all the tasks to learn generic representations; here,
parameter sharing could be either hard or soft depending on the problem. For model training,
we create a multi-objective loss through a weighted linear sum of the individual tasks’ losses
as:

\[ L_{\text{aggregated}} = \sum_{\tau \in T} \psi_{\tau} \times L_{\tau} \]  

(2.6)

where \( \psi \) denotes a task weight and \( L_{\tau} \) (generally between 0 and 1) is a task-specific loss
function. It is important to note, MTL itself does not impose any restriction on the type
of loss for an individual task. Therefore, unsupervised and supervised tasks or tasks having
different objective functions can be conveniently combined for learning representations. We
reiterate that for learning numerous tasks together to be successful, we need to ensure the
task relatedness and how it will be encoded in the neural architecture design to capture linked
aspects of the input. We also need to consider several other factors to gain benefits from
joint training. For instance, we can not assume that training all tasks together is useful for all
the tasks as there can be task interference leading to degenerate models. A review of common
issues arising from applying MTL to real-world problems and potential solutions is beyond the
scope of this section, and we recommend an interested reader to consult \[33\]. We use multi-
task learning in conjunction with self-supervision in Chapters 3 and 10 and in Chapter 9 in
connection with model personalization.

2.4 Knowledge Transfer

The transfer learning describes a class of techniques that aims at leveraging and preserving
previously acquired knowledge from solving prior tasks to accelerate the learning of the fu-
ture novel task. Importantly, depending on the tasks and input modalities, the knowledge
transfer can take on various forms. Relevant to our purpose throughout this thesis, it involves
learned representations and weights by the deep neural networks. We either use pretrained
models as fixed feature extractors or as model initialization to address the inadequacy of the
classical supervised learning paradigm in learning from few-labeled data. The transfer learn-
ing domain has seen a rapid rate of progress and diversity of methods in recent years. It has
shown remarkable improvement in performance on very challenging problems, especially in
areas where little-labeled data are available, e.g., natural language understanding tasks \[34\].

In transfer learning, the broad objective is to reuse the learned knowledge from a source
domain \( D_{\text{SRC}} \) to a target domain \( D_{\text{TRG}} \). Precisely, we refer to domain comprising an input space
\( \mathcal{X} \), a output space \( \mathcal{Y} \), and associated probability distribution \( p \), which describes a sample \( x_i \sim p \)
is drawn from a particular domain. We can consider domains \( D_{\text{SRC}} \) and \( D_{\text{TRG}} \) with learning
tasks \( T_{\text{SRC}} \) and \( T_{\text{TRG}} \), respectively. The goal is to help improve the learning of a predictive
function \( f_{\theta}(\cdot) \) in \( D_{\text{TRG}} \) using knowledge extracted from \( D_{\text{SRC}} \) and \( T_{\text{SRC}} \), where, \( D_{\text{SRC}} \neq D_{\text{TRG}} \)
and/or \( T_{\text{SRC}} \neq T_{\text{TRG}} \), meaning that domains or tasks can be different. Specifically, when the
source and target tasks are the same, it is typically known as transductive transfer, while in the
inductive transfer the source task is different from the target. The earlier discussed multi-task setting is a prime example of inductive transfer learning. Given this learning formulation, we can develop high-quality models under different knowledge transfer scenarios (such as features, instances, weights, architectures) from existing labeled or unlabeled data of some related task or domain. For a detailed survey, we refer the interested reader to see [21]. We extensively explore knowledge transfer in relation to self-supervised learning in Chapters 3, 4, as a joint learning scheme to solve multiple tasks in Chapter 10, and for personalization and domain adaptation in Chapter 9.

2.5 Self-Supervision

Several chapters in this thesis explore self-supervised learning, for which we provide a brief introduction here. The field of self-supervision describes a class of methods that exploits the natural supervision available within the input to learn deep models without relying on semantic annotations. In particular, we define a surrogate (or auxiliary or pretext) task for which supervision can be acquired from the data itself, and importantly learning to solve the specified task can force the network to learn broadly-useful representations. It is ideal for ambient sensing models (both in-cloud and on-device) utilizing sensory inputs as getting such data labeled for developing fully-supervised models is prohibitively expensive. Further, it reduces the overhead of human intervention in the annotation process, providing a promising way for learning representations from a huge amount of unlabeled data.

Formally, given an unlabeled data \( D = \{x\}^{M}_{m=1} \) and network \( f_\theta(\cdot) \), the aim is to pre-train the network with a surrogate task, where, labels \( y \) for the standard objective functions are extracted automatically from \( x \). The learned model is then utilized either as a fixed feature extractor or as an initialization for rapidly learning an end-task of interest. To that end, very recently, numerous pretext tasks are proposed in different domains. Most prominently, geometric transformation detection [35], colorization of gray-scale images [36], solving jigsaw puzzle [37], masked language modeling [38], audio-visual synchronization [39], arrow of time in videos [40], and many more. Among them, techniques based on noise contrastive estimation [41, 42] and deep metric learning [43] have seen significant research interest due to their applicability on several modalities (such as images, audio, text, and time-series). Briefly, the core idea of contrastive learning is to learn representations through maximizing similarity between semantically related data pairs \((x, x^+)\) while minimizing it among unrelated ones \((x, x^-)\). The selection of positive and negative input pairs is essential to the success of contrastive learning. The common approach is to generate positive pairs via augmentation, such as cropping in the case of images, whereas negative pairs can be generated by randomly sampling from other instances. Thus, a typical objective function that is minimized to determine correct positives from a set of negatives takes on the form:

\[
\mathcal{L} = - \log \frac{\exp(s(x, x^+))}{\sum_{x^- \in \mathcal{X}^-|x| \cup \{x^+\}} \exp(s(x, x^-))} \tag{2.7}
\]

where \( \mathcal{X}^- \) refers to the set of negative distractors. The \( s \) denotes a similarity score be-
between instances e.g., cosine or bilinear similarity applied over embeddings from an encoder \( f_e(\cdot) \) and projection head \( g(\cdot) \). Specifically, in case of bilinear score, \( s \) becomes \( s(x, x') = g(f_e(x))^\top W g(f_e(x')) \). This objective function, unlike the triplet loss \([44]\), leverages multiple distractors at a time. Likewise, a closely related area of metric learning employs a distance metric, such as squared Euclidean distance \( d(x_i, x_j) = \|x_i - x_j\|^2 \) to ensure greater similarity between embedding of alike pairs (also called anchor-positive) as compared to others through optimizing a type of hinge loss Equation \(2.8\) with \( \alpha \) being a margin hyperparameter which is enforced between positive and negative samples.

\[
\mathcal{L} = y \|f(x_i) - f(x_j)\|^2 + (1 - y) \max(\alpha - \|f(x_i) - f(x_j)\|^2, 0) \quad (2.8)
\]

We review additional prior work and provide additional descriptions of the related methods in subsequent chapters as required. We develop several pretext tasks for the self-supervision of deep networks across a wide range of modalities and problems in ambient, personal and embedded intelligence.

### 2.6 Modeling Signals and Time-Series

The modern IoT devices are embedded with a multitude of sensors that have the capabilities to constantly monitor various physical phenomena and record them in digital format for further analysis. Here, the majority of our work concerns learning representations from the sensory streams or time-series collected from a wide variety of sensors; we provide a brief description of time-series processing for deep models, sensing, and related applications. The overview is important to put things into perspective for motivating subsequently developed methods to augment IoT devices with self-learning capabilities. Time-series modeling has a broad range of applications in several domains, from healthcare to industrial and consumer applications.

We can categorize time-series input into two classes a) univariate, where only a single variable is available at each time-step and b) multivariate, which can have more than one variable sampled uniformly for each time interval. For instance, when a wearable device collects heart rate at a sampling rate of 1Hz for a minute, it results in a univariate time-series with 60 data samples. Additionally, if we choose to collect electrodermal activity (or skin conductance) and temperature at the same interval, we will have a multivariate time-series. This sequential data or time-series intrinsically exhibits temporal dependencies among samples, which we can model efficiently with deep models (such as with temporal convolutional networks) to learn high-level representations solely using raw data as input and without requiring any manual feature engineering.

We focus primarily on using a deep model for time-series classification (and regression) problems, but our developed techniques can be potentially used for other similar tasks, e.g., change point detection, retrieval and alignment. We define \( \mathcal{D} = \{(x_1, y_1, \ldots, (x_m, y_m)\} \), where \( x_i = [x_{1,i}, \ldots, x_{T,i}] \) is a segment of a longer sequence or an instance generated through a segmentation operation with \( x_{t,i} \in \mathbb{R}^d \) being a data point at \( t \)-th time-step, \( T \) denoting the total length of the segment and \( d \) representing the number of multivariate signals. For classification tasks, each segment \( x \) has an associated class label \( y \in \mathbb{Y} \) from a
predefined label set with $|\mathcal{Y}|$ the number of total classes. Similarly, in the case of regression problems, $y$ is a real-valued scalar value. For our purpose, we consider segmented data given as input-label pairs, where we create input subsequences through applying a sliding window with a window length of $T$ and stride (step-size) of $\mu$ on a stream of incoming data. Likewise, in cases where fixed well-aligned labels per segment are not available, we create them by taking the mode (i.e., a value that appears most often) of the values falling within a corresponding window unless specified otherwise. This process is widely used in time-series data processing for machine learning tasks as it avoids manual semantic segmentation, which is both costly and time-consuming. Now, given a dataset $\mathcal{D}$, the objective of the time-series learning task is to train a classifier or regression model $f_\theta(x) \rightarrow y$ to predict the class label or scalar value depending on the problem.
Chapter 3

Self-Supervised Learning with Transformation Prediction

This chapter is based on our paper Multi-task Self-Supervised Learning for Human Activity Recognition published in ACM IMWUT 2019 [14].

3.1 Introduction

In light of these challenges, we pose the question whether it is possible to learn semantic representations in an unsupervised way to circumvent the manual annotation of the sensor data with strong labels, e.g., activity classes. In particular, the goal is to extract features that are on par with those learned with fully-supervised methods. There is an emerging paradigm for feature learning called self-supervised learning that defines auxiliary (also known as pretext or surrogate) tasks to solve, where labels are readily extractable from the data without any human intervention, i.e., self-supervised. The availability of strong supervisory signals from the surrogate tasks enables us to leverage objective functions as utilized in a standard supervised learning setting [45]. For instance, the vision community proposed a considerable number of self-supervised tasks for advancing representation learning¹ from static images, videos, and audio (see Section 3.4). Most prominent among them are: colorization of grayscale images [36], predicting image rotations [35], solving jigsaw puzzles [37], predicting the direction of video playback [40], temporal order verification [47], odd sequence detection [48], audio-visual correspondence [39, 49], and curiosity-driven agents [50]. The presented methodology for sensor representation learning takes inspiration from these methods and takes leverage of signal transformations to extract highly generalizable features for the down-stream² task, i.e., HAR.

¹ also known as feature learning
² or an end-task
Our work is motivated by the success of jointly learning to solve multiple self-supervised tasks [32, 45] and we propose to learn accelerometer representations (i.e., features) by training a temporal convolutional neural network (CNN) to recognize the transformations applied to the raw input signal. Particularly, we utilize a set of signal transformations [51, 52] that are applied on each input signal in the datasets, which are then fed into the convolutional network along with the original data for learning to differentiate among them. In this simple formulation, a group of binary classification tasks (i.e., to recognize whether a transformation such as permutation, scaling, and channel shuffling was applied on the original signal or not) act as surrogate tasks to provide a rich supervisory signal to the model. In order to extract highly generalizable features for the end-task of interest, it is essential to utilize transformations that exploit versatile invariances of the temporal data (further details are provided in Section 3.2). To this end, we utilize eight transformations to train a multi-task network for simultaneously recognizing each of them. The visual illustration of the proposed approach is given in Figure 3.1. In the pretraining phase, the network consisting of a common trunk with a separate head for each task is trained on self-supervised data, and in the second step, the features learned by the shared layers are utilized by the HAR model. Importantly, we want to emphasize that in order for the convolutional network to recognize the transformations,
it must learn to understand the core signal characteristics through acquiring knowledge of underlying differences in the accelerometer signals for various activity categories. We support this claim through an extensive evaluation of our method on six publicly available datasets in unsupervised, semi-supervised and transfer learning settings, where it achieves noticeable improvements in all the cases while not requiring manually labeled data for feature learning.

Our main contributions are as follows:

- We propose to utilize self-supervision from large unlabeled data for human activity recognition.
- We design a signal transformation recognition problem as a surrogate task for annotation free supervision, which provides a strong training signal to the temporal convolutional network for learning generalizable features.
- We demonstrate through extensive evaluation that the self-supervised features perform significantly better in the semi-supervised and transfer learning settings on several publicly available datasets. Moreover, we show that these features achieve performance that is superior to or comparable with the features learned via the fully-supervised approach (i.e., trained directly with activity labels).
- We illustrate with SVCCA, saliency mapping, and t-SNE visualizations that the features extracted via self-supervision are very similar to those learned by the fully-supervised network.
- Our method substantially reduces the labeled data requirement, effectively narrowing the gap between unsupervised and supervised representation learning.

3.2 Approach

In this section, we present our self-supervised representation learning framework for HAR. First, we provide an overview of the methodology. Next, we discuss various learning tasks (i.e. transformation classification) and their benefits for generic features extraction from unlabeled data. Finally, we provide a detailed description of the network architecture, its implementation, and the optimization process.

3.2.1 Overview

The objective of our work is to learn general-purpose sensor representations based on a temporal convolutional network in an unsupervised manner. To achieve this goal, we introduce a self-supervised deep network named Transformation Prediction Network (TPN), which simultaneously learns to solve multiple (signal) transformation recognition tasks as shown in Figure 3.1. Specifically, the proposed multi-task TPN $M_\theta(\cdot)$ is trained to produce estimates of the transformations applied to the raw input signal. We define a set of distinct transformations (or tasks) $\{J_t(\cdot)\}_{t \in T}$, where $J_t(\cdot)$ is a function that applies a particular signal
alteration technique \( t \) to the temporal sequence \( x \in \mathbb{R}^{(N,C)} \) to yield a transformed version of the signal \( J_t(x) \). The network \( M_\theta(.) \) that has a common trunk and individual head for each task, it takes an input sequence and produces a probability of the signal being a transformed version of the original, i.e. \( P(J_t|x) = M_\theta(x) \). Note, that given a set of unlabeled signals (e.g. of accelerometer), we can automatically construct a self-supervised labeled dataset \( D = \{(J_t(x_i), True), (x_i, False)\}_{t \in T}^m \). Hence, given this set of \( m \) training instances, the multi-task self-supervised training objective that a model must learn to solve is:

\[
\min_{\theta} \sum_{t \in T} \psi_t \left[ -\frac{1}{m_t} \sum_{i=1}^{m_t} (y_t^i \log(M_\theta(x_t^i)) + (1 - y_t^i) \log(1 - M_\theta(x_t^i))) \right]
\]

(3.1)

where \( M_\theta(x^t) \) is the predicted probability of \( x \) being a transformed version \( t \) and \( \theta \) are the learnable parameters of the network. \( m_t \) represents the number of instances for a task (which can vary but are equal in our case) and \( \psi_t \) is the loss-weight of task \( t \).

We emphasize that, although the network has a separate layer to differentiate between original and each of the \( T \) transformations it can be extended in a straightforward manner to recognize multiple transformations applied to the same input signal or for multi-label classification. In the following subsection, we explain the types of signal transformations that are used in this work.

### 3.2.2 Self-Supervised Task: Signal Transformations

The aforementioned formulation requires the signal transformations \( J \) to define a multi-task classification that enables the convolutional model to learn disentangled semantic representations useful for downstream tasks, e.g. activity detection. We aimed for conceptually simple, yet diverse tasks to possibly cover several invariances that commonly arise in temporal data \([52]\). Intuitively, a diverse set of tasks should lead to a broad spectrum of features, which are more likely to span the feature-space domain needed for a general understanding of the signal’s characteristics. In this work, we propose to utilize eight straightforward signal transformations (i.e. \( |T| = 8 \)) \([51, 52]\) for the self-supervision of a network. More specifically, when transformations are applied on an input signal \( x \), they result in eight variants of \( x \). As mentioned earlier, the temporal convolutional model is then trained jointly on all the tasks’ data to solve a problem of transformation recognition, which allows the model to extract high-level abstractions from the raw input sequence. The transformations utilized in this work are summarized below:

- **Noised**: Given sensor readings of a fixed length, a possible transformation is the addition of random noise (or jitter) in the original signal. Heterogeneity of device sensors, software, and other hardware can cause variations (noisy samples) in the produced data. A model that is robust against noise will generalize better as it learns features that are invariant to minor corruption in the signal.

- **Scaled**: A transformation that changes the magnitude of the samples within a window through multiplying with a randomly selected scalar. A model capable of handling
scaled signals produces better representations as it becomes invariant to amplitude and offset invariances.

- **Rotated**: Robustness against arbitrary rotations applied on the input signal can achieve sensor-placement (orientation) invariance. This transformation inverts the sample signs (without changing the associated class-label) as frequently happens if the sensor (or device) is, for example, held upside down.

- **Negated**: This simple transformation is an instance of both scaled (scaling by \(-1\)) and rotated transformations. It negates samples within a time window, resulting in a vertical flip or a mirror image of the input signal.

- **Horizontally Flipped**: This transformation reverses the samples along the temporal dimension, resulting in a complete mirror image of an original signal as if it were evolved in the opposite time direction.

- **Permuted**: This transformation randomly perturbs the events within a temporal window through slicing and swapping different segments of the time-series to generate a new one, hence, facilitating the model to develop permutation invariance properties.

- **Time-Warped**: This transformation locally stretches or warps a time-series through a smooth distortion of time intervals between the values (also known as local scaling).

- **Channel-Shuffled**: For a multi-component signal such as a triaxial accelerometer, this transformation randomly shuffles the axial dimensions.

There are several benefits of utilizing transformations recognition as auxiliary tasks for feature extraction from unlabeled data.

**Enabling the learning of generic representations**: The primary motivation is that the above-defined pretext tasks enable the network to capture the core signal characteristics. More specifically, for the TPN to successfully recognize if the signal is transformed or not, it must learn to detect high-level semantics, sensor behavior under different device placements, time-shift of the events, varying amplitudes, and robustness against sensor noise, thus, contributing to solving the ultimate task of HAR.

**Task diversification and elimination of low-level input artifacts**: A clear advantage of using multiple self-supervised tasks as opposed to a single one is that it will lead to a more diverse set of features that are invariant to low-level artifacts of the signals. Had we chosen to utilize signal reconstruction, e.g., with autoencoders, this would learn to compress the input, but due to a weak supervisory signal (as compared to self-supervision), it may discover trivial features with no practical value for the activity recognition or any other task of interest. We compare our approach against other methods in Section 3.3.3.

**Transferring knowledge**: Furthermore, with our approach, the unlabeled sensor data that are produced in huge quantity can be effectively utilized with no human intervention to pre-train a network that is suitable for semi-supervised and transfer learning settings. It is particularly of high value for training networks in a real-world setting, where very little or no supervision is available to learn a model of sufficient quality from scratch.
**Other benefits:** Our self-supervised method has numerous other benefits. It has an equivalent computational cost to supervised learning but with better convergence accuracy, making it a suitable candidate for continuous unsupervised representation learning in-the-wild. Moreover, our technique neither requires a sophisticated pre-processing (apart from z-normalization) nor needs a specialized architecture (which also requires labeled data) to exploit invariances. We will show in Section 3.3.3 through extensive evaluation that the self-supervised models learn useful representations and dramatically improve performance over other learning strategies. Despite the simplicity of the proposed scheme, it allows utilizing data collected through a wide variety of devices from a diverse set of users.

### 3.2.3 Network Architecture and Implementation

We implement the TPN $M_\theta(\cdot)$ as a multi-branch temporal convolutional neural network with a common trunk (shared layers) and a distinct head (private layers) for each task with a separate loss function. Hard parameter sharing is employed between all the task-specific layers to encourage strong weight utilization from the trunk. Figure 3.4 illustrates the TPN containing three 1D convolutional layers consisting of 32, 64, and 96 feature maps with kernel sizes of 24, 16 and 8 respectively, and having a stride of 1. Dropout is used after each of the layers with a rate of 0.1, and L2 regularization is applied with a rate of 0.0001. Global max pooling is used after the last convolution layer to aggregate high-level discriminative features. Moreover, each task-specific layer is comprised of a fully-connected layer of 256 hidden units followed by a sigmoidal output layer for binary classification. We use ReLU as non-linearity in all the layers (except the output) and train a network with Adam optimizer for a maximum of 30 epochs with a learning rate of 0.0003, unless stated otherwise. Furthermore, the activity recognition model has a similar architecture to the TPN except for a fully-connected layer that consists of 1024 hidden units followed by a softmax output layer with units depending on the activity detection task under consideration. Additionally, during training of this model, we apply early-stopping, if the network fully converges on the training set to avoid overfitting.

The motivation for keeping the TPN architecture simple arises from the fact that we want to show the performance gain does not come from the number of parameters (or layers) or due to the utilization of other sophisticated techniques such as batch normalization but the improvement is due to self-supervised pretraining. Likewise, the choice of multi-task learning setting, where each task has an additional private layer manifests in letting the model push pretext task-specific features to the last layers and let the initial layers extract generic representations that are important for a wide variety of end-tasks. Moreover, our architectural specification allows for a straightforward extension to add other related tasks, if needed, such as input reconstruction. Although, we do not explore applying multiple transformations to the same sequence or train models for their recognition the network design is intrinsically capable of performing this multi-label classification task.

During the training process, for every instance, we first generate transformed versions of a signal for the self-supervised pretraining of the network. At each training iteration of the TPN model, we feed the data from all tasks simultaneously, and the overall loss is calculated as a weighted sum of the losses of different tasks. Once pretraining converges, we transfer the weights of convolutional layers from model $M_\theta$ to an activity recognition network $C_\theta$. 
for learning the final supervised task. Here, either all the transferred layers are kept frozen, or the last convolutional layer is fine-tuned depending on the learning paradigm. Figure 3.2 depicts this process graphically, where shaded convolutional layers represent frozen weights, while others are either trained from scratch or optimized further on the end-task. To avoid ambiguity, in the experiment section, we explicitly mention when the results are from a fully-supervised or self-supervised (including fine-tuned) network.

Figure 3.2: Detailed architectural specification of transformation prediction and activity recognition networks. We propose a framework for self-supervised representation learning from unlabeled sensor data (such as an accelerometer). Various signal transformations are utilized to establish supervisory tasks, and the network is trained to differentiate between an original and transformed version of the input. The three blocks of Conv + ReLU and Dropout layers, which is followed by a Global Max Pooling are similar across both networks. However, the multi-task model has a separate head for each task. Likewise, the activity recognizer has an additional densely connected layer. The TPN is pretrained on self-supervised data, and the learned weights are transferred (depicted by a dashed arrow) and kept frozen to the lower model, which is then trained to detect various activities.

3.3 Experiments

In this section, we conduct an extensive evaluation of our approach on several publicly available datasets for human activity recognition (HAR) in order to determine the quality of learned representations, transferability of the features, and benefits of this in the low-data regime. The self-supervised tasks (i.e., transformation predictions) are utilized for learning rich sensor representations that are suitable for an end-task. We emphasize that achieving high performance on these surrogate tasks is not our focus.
Table 3.1: Summary of datasets used in our evaluation. These datasets are selected based on the diversity of participants, device types and activity classes. Further details on the pre-processing of each data source and the number of users utilized are discussed in Section 3.3.1.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No. of users</th>
<th>No. of activity classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHAR</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>UniMiB</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>UCI HAR</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>MobiAct</td>
<td>67</td>
<td>11</td>
</tr>
<tr>
<td>WISDM</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>MotionSense</td>
<td>24</td>
<td>6</td>
</tr>
</tbody>
</table>

3.3.1 Datasets

We consider six publicly available datasets to cover a wide variety of device types, data collection protocols, and activity recognition tasks performed with smartphones in different environments. Some important aspects of the data are summarized in Table 3.1. Below, we give brief descriptions of every dataset summarizing its key points.

HHAR

The Heterogeneity Human Activity Recognition (HHAR) dataset contains signals from two sensors (accelerometer and gyroscope) of smartphones and smartwatches for 6 different activities, i.e. biking, sitting, standing, walking, stairs-up and stairs-down. The 9 participants executed a scripted set of activities for 5 minutes to get equal class distribution. The subjects had 8 smartphones in a tight pouch carried around their waist and 4 smartwatches, 2 worn on each arm. In total, they used 36 different smart devices of 13 models from 4 manufacturers to cover a broad range of devices for sampling rate heterogeneity analysis. The sampling rate of signals varied significantly across phones with values between 50-200Hz.

UniMiB

This dataset contains triaxial accelerometer signals collected from a Samsung Galaxy Nexus smartphone at 50Hz. Thirty subjects participated in the data collection process forming a diverse sample of the population with different height, weight, age, and gender. The subject placed the device in her trouser's front left pocket for a partial duration and in the right pocket for the remainder of the experiment. We utilized the data of 9 activities of daily living (i.e., standing up from sitting, standing up from lying, walking, running, upstairs, jumping, downstairs, lying down from sitting, sitting).

UCI HAR

The UCI HAR dataset is obtained from a group of 30 volunteers with a waist-mounted Samsung Galaxy S2 smartphone. The accelerometer and gyroscope signals are collected at
50Hz when subjects performed the following six activities: standing, sitting, laying down, walking, downstairs and upstairs.

**MobiAct**

The MobiAct³ dataset [59] contains signals from a smartphone’s inertial sensors (accelerometer, gyroscope, and orientation) for 11 different activities of daily living and 4 types of falls. It is collected with a Samsung Galaxy S3 smartphone from 66 participants of different gender, age group, and weight through more than 3200 trials. The device is placed in a trouser’s pocket freely selected by the subject in any random orientation to capture everyday usage of the phone. We used the data from 61 participants who have data samples for any of the following 11 activities: sitting, walking, jogging, jumping, stairs up, stairs down, stand to sit, sitting on a chair, sit to stand, car step-in, and car step-out.

**WISDM**

The dataset from the Wireless Sensor and Data Mining (WISDM) project [60] was collected in a controlled study from 29 volunteers, who carried the cell phone in their pockets. The data were recorded for 6 different activities (i.e., sit, stand, walk, jog, ascend stairs, descend stairs) via an app developed for an Android phone. The accelerometer signal was acquired every 50ms (sampling rate of 20Hz). We use the data of all the users available in the raw data file with user ids ranging from 1 to 36.

**MotionSense**

The MotionSense dataset [61] comprises an accelerometer, gyroscope, and altitude data from 24 participants of varying age, gender, weight, and height. It was collected using an iPhone6s, which is kept in the user’s front pocket. The subjects performed 6 different activities (i.e., walking, jogging, downstairs, upstairs, sitting, and standing.) in 15 trials under similar environments and conditions. The study aimed to infer physical and demographics attributes from time-series data in addition to the detection of activities.

### 3.3.2 Pre-Processing and Assessment Strategy

We applied minimal pre-processing on the accelerometer signals as deep neural networks are very good at learning abstract representations directly from raw data [62]. We segmented the signals into fixed size windows that have 400 samples with 50% overlap, for all the datasets under consideration. The appropriate window size is a task-specific parameter and could be tuned or chosen based on prior knowledge for improved performance. Here, we utilize the same window size based on earlier exploration across datasets and to keep experimental evaluation impartial towards the effect of this hyper-parameter. Next, we divide each dataset

³second release
into training and test sets through randomly selecting 20–30% of the users for testing and the rest for training and validation; depending on the dataset size. We used the ceiling function to select number of users, e.g. from HHAR dataset 3 users are used for evaluation out of 9. The training set users’ data are further divided into 80% for training the network and 20% for validation and hyper-parameter tuning. Importantly, we also evaluate our models through user-split based 5-folds cross-validation, wherever it is appropriate. Finally, we normalize the data by applying z-normalization with summary statistics calculated from the training set. We generate self-supervised data from an unlabeled training set that is produced as a result of the processing as mentioned earlier. We utilize the data generation procedure as explained earlier in Section 3.2.3.

Furthermore, due to the large size of the HHAR dataset and in order to reduce computational load, we randomly sample 4000 instances from each users’ data to produce transformed signals. Likewise, in the case of UniMiB because of its relatively small size, we generate 5 times more transformed instances. We evaluate the performance with Cohen’s kappa, a weighted version of precision, recall and f-score metrics to be robust against inherent imbalanced nature of the datasets. It is important to highlight that, we use a network architecture with the same configuration across the datasets to evaluate models’ performance in order to highlight improvement is indeed due to self-supervision and not due to architectural modifications.

3.3.3 Results

Quantifying the Quality of Learned Feature Hierarchies

We first evaluate our approach to determine the quality of learned representations versus the model depth (i.e., the layer number from which the features come). This analysis helps in understanding whether the features coming from different layers vary in quality concerning their performance on an end-task and if so, which layer should be utilized for this purpose. To this end, we first pretrain our TPN in a self-supervised manner and learn classifiers on top of ConvA, ConvB, and ConvC layers independently, for several activity recognition datasets. These classifiers (see Figure 3.2) are trained in a supervised way while keeping the learned features fixed during the optimization process. Figure 3.3 provides kappa values on test sets averaged across 10-independent runs to be robust against differences in weight initializations of the classifiers. We observe that for a majority of the datasets the model performance improves with increasing depth apart from HHAR, where features from ConvB layer results in improved detection rate with a kappa of 0.774 compared to 0.679 of ConvC. It may be because the representation of the last layer starts to become too specific on the transformation prediction task or it may also be because we did not utilize the entire dataset for the self-supervision. To be consistent, in the subsequent experiments we used features from the last convolutional layer for all the considered datasets. For a new task or recognition problem, we recommend performing a similar analysis to identify layer/block of the network that gives optimal results on the particular dataset.
Figure 3.3: Evaluation of activity classification performance using the features learned based on self-supervision (per layer). We train an activity classifier on-top of each of the temporal convolution blocks (ConvA, ConvB, and ConvC) that are pretrained with self-supervision. The reported results are averaged over 10 independent runs (i.e., training an activity classifier from scratch). ConvA, ConvB, and ConvC have 32, 64, and 96 feature maps, respectively.

Comparison against Fully-Supervised and Unsupervised Approaches

In this subsection, we assess our self-supervised representations learned with TPN against other unsupervised and fully-supervised techniques for feature learning. Table 3.2 summarizes the results with respect to four evaluation metrics (namely, precision, recall, f-score, and kappa) for 10-independent runs on the six datasets described earlier. For the Random Init. entries, we keep the convolutional network layers frozen during optimization and train only a classifier in a supervised manner. Likewise, for an Autoencoder, we keep the network architecture the same and pretrain it in an unsupervised way. Afterward, the weights of the encoder are kept frozen, and a classifier is trained on top as usual. The Self-Supervised entries show the result of the convolutional network pretrained with our proposed method, where a classifier is trained on top of the frozen network in a supervised fashion. Furthermore, Self-Supervised (FT) entries highlight the performance of the network trained with self-supervision but the last convolution layer, i.e. ConvC is fine-tuned along with a classifier during training on the activity recognition task. Training an activity classification model on top of randomly initialized convolutional layers poorly performs as expected, which is evidence that the performance improvement is not only because of the activity classifier. These results are followed by a widely used unsupervised learning method, i.e. an autoencoder. The self-supervised technique outperforms existing methods and achieves results that are on par with the fully-supervised model. It is important to note that, for our proposed technique, only the classifier layers are randomly initialized and trained with activity specific labels (the rest is transferred from the self-supervised network). We also observe that fine-tuning the last convolutional layer further improves the classification performance of the down-stream tasks on several datasets such as UniMiB, HHAR, MobiAct, and UCI HAR. The results show that TPN can learn highly generalizable representations, thus reducing the performance gap of feature learning with the
(end-to-end) supervised case. For a more rigorous evaluation, we also performed 5-folds (user split based) cross-validation for every method on all the datasets. The results are provided in Table 3.2, which also shows that the self-supervised method reduces the performance gap with the supervised setting.

Table 3.2: Task Generalization: Evaluating self-supervised representations for activity recognition. We compare the proposed self-supervised method for representation learning with fully-supervised and unsupervised approaches. We use the same architecture across all the experiments. The self-supervised TPN is trained to recognize transformations applied on the input signal while the activity classifier is trained on top of these learned features where Self-Supervised (FT) entry provides results when the last convolution layer is fine-tuned. The Random Init. entries present results when the convolution layers are randomly initialized and kept frozen during the training of the classifier. The results reported are averaged over 10 independent runs to be robust against variations in the weight initialization and the optimization process.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) HHAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.388±0.0537</td>
<td>0.310±0.0409</td>
<td>0.241±0.0404</td>
<td>0.174±0.0488</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.762±0.0312</td>
<td>0.733±0.0308</td>
<td>0.727±0.0297</td>
<td>0.681±0.0317</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.731±0.0451</td>
<td>0.667±0.0663</td>
<td>0.658±0.0724</td>
<td>0.599±0.0784</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.798±0.0155</td>
<td>0.777±0.0199</td>
<td>0.766±0.0234</td>
<td>0.731±0.0243</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.8218±0.0256</td>
<td>0.797±0.0211</td>
<td>0.786±0.0187</td>
<td>0.755±0.0255</td>
</tr>
<tr>
<td>(b) UniMiB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.4256±0.0468</td>
<td>0.3546±0.037</td>
<td>0.2775±0.0491</td>
<td>0.2243±0.0474</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.8276±0.0148</td>
<td>0.8096±0.0266</td>
<td>0.8097±0.0248</td>
<td>0.7815±0.0299</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.5922±0.0191</td>
<td>0.557±0.0232</td>
<td>0.537±0.0339</td>
<td>0.4824±0.0275</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.8133±0.0077</td>
<td>0.7954±0.014</td>
<td>0.7929±0.016</td>
<td>0.7642±0.0162</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.8106±0.007</td>
<td>0.8432±0.0049</td>
<td>0.8425±0.0054</td>
<td>0.8197±0.005</td>
</tr>
<tr>
<td>(c) UCI HAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.689±0.0648</td>
<td>0.4392±0.0692</td>
<td>0.3713±0.0952</td>
<td>0.3133±0.0866</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.9059±0.033</td>
<td>0.8998±0.0139</td>
<td>0.8981±0.0148</td>
<td>0.8789±0.0168</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.814±0.059</td>
<td>0.7877±0.1112</td>
<td>0.7772±0.1306</td>
<td>0.7425±0.1359</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.9100±0.0081</td>
<td>0.9011±0.0139</td>
<td>0.8987±0.0155</td>
<td>0.8803±0.0169</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.9057±0.0121</td>
<td>0.897±0.0185</td>
<td>0.8946±0.019</td>
<td>0.8754±0.0222</td>
</tr>
<tr>
<td>(d) MobiAct</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.474±0.1128</td>
<td>0.3452±0.1128</td>
<td>0.2811±0.0982</td>
<td>0.1915±0.1017</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.908±0.0066</td>
<td>0.895±0.0167</td>
<td>0.897±0.0133</td>
<td>0.8665±0.0202</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.749±0.0328</td>
<td>0.7581±0.0354</td>
<td>0.7293±0.0452</td>
<td>0.6772±0.0517</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.9095±0.0035</td>
<td>0.9059±0.0059</td>
<td>0.906±0.0053</td>
<td>0.8795±0.0073</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.9194±0.0057</td>
<td>0.9102±0.0114</td>
<td>0.9117±0.0093</td>
<td>0.8855±0.014</td>
</tr>
<tr>
<td>(e) WISDM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.5942±0.0599</td>
<td>0.3543±0.077</td>
<td>0.358±0.0837</td>
<td>0.2224±0.0656</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.9024±0.0206</td>
<td>0.8617±0.0206</td>
<td>0.8764±0.0168</td>
<td>0.8211±0.0258</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.6561±0.2737</td>
<td>0.6631±0.1623</td>
<td>0.6358±0.2355</td>
<td>0.3106±0.288</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.8894±0.0096</td>
<td>0.8484±0.0269</td>
<td>0.8931±0.0215</td>
<td>0.7986±0.034</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.8999±0.0111</td>
<td>0.8368±0.0175</td>
<td>0.8686±0.0154</td>
<td>0.8106±0.0466</td>
</tr>
<tr>
<td>(f) MotionSense</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.5999±0.0936</td>
<td>0.5029±0.0931</td>
<td>0.4681±0.1105</td>
<td>0.376±0.1176</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.9164±0.0053</td>
<td>0.8993±0.0091</td>
<td>0.9027±0.0085</td>
<td>0.8761±0.011</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.8255±0.0132</td>
<td>0.8166±0.0195</td>
<td>0.8109±0.0169</td>
<td>0.7659±0.0226</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.8979±0.0073</td>
<td>0.8816±0.0087</td>
<td>0.8864±0.0083</td>
<td>0.8589±0.0106</td>
</tr>
</tbody>
</table>
Table 3.2 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Activity 1</th>
<th>Activity 2</th>
<th>Activity 3</th>
<th>Activity 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.915 ± 0.0088</td>
<td>0.8979 ± 0.0092</td>
<td>0.9005 ± 0.0094</td>
<td>0.8744 ± 0.0112</td>
</tr>
</tbody>
</table>

Assessment of Individual Self-Supervised Tasks in Contrast with Multiple Tasks

In Figure 3.4, we show comparative performance analysis of single self-supervised tasks with each other and importantly with a multi-task setting. This assessment helps us in understanding whether self-supervised features extracted via jointly learning to solve multiple tasks are any better (for activity classification) than independently solving individual tasks and whether multi-task learning helps in learning more useful sensor semantics. To achieve this, we pretrain a TPN on each of the self-supervised tasks and transfer the weights for learning an activity recognition classifier. We observe in all the cases that learning representations via solving multiple tasks lead to far better performance on the end-task. This further highlights that the features learned through various self-supervised tasks have different strengths and weaknesses. Therefore, merging multiple tasks results in an improvement in learning a diverse set of features. However, we notice that some tasks (such as Channel Shuffled, Permuted, and Rotated) consistently performed better compared to others across datasets; achieving a kappa score above 0.60 as evaluated on different activity recognition problems. It highlights an important point that there may exist a group of tasks, which are reasonably sufficient to achieve a model of good quality. Furthermore, in Figure 3.10, we plot the kappa score achieved by a multi-task TPN on transformation recognition tasks as a function of the number of training epochs. This analysis highlights that task complexity varies greatly from one dataset to another and may help with the identification of trivial auxiliary tasks that may lead to non-generalizable features.

In addition to activity classification, for any learning task involving time-series sensor data (e.g., as encountered in a various Internet of Things applications), we recommend extracting features through first solving individual tasks and later focusing on the multi-task scenario; discarding low performing tasks or assigning low-weights to the loss functions of the respective tasks. Another approach could be to auto-tune the task-loss weight by taking homoscedastic uncertainty of each task into account [63].

Effectiveness under Semi-Supervised Setting

Our proposed self-supervised feature learning method attains very high performance on different activity recognition datasets. This brings up the question, whether the self-supervised representations can boost performance in the semi-supervised learning setting as well or not. In particular, can we use this to perform activity detection with very little labeled data? Intrigued by this, we also evaluate the effectiveness of our approach to semi-supervised learning. Specifically, we initially train a TPN on an entire training set for transformation prediction. Subsequently, we learn a classifier on top of the last layer’s feature maps with only a subset of the available accelerometer samples and their corresponding activity labels. For training an activity classifier, we use for each category (class) 2, 5, 10, 20, 50, and 100 examples. Note that, 2-10 samples per class represent a real-world scenario of acquiring a (small) labeled dataset from human users with minimal interruption to their daily routines, hence, making self-
supervision from unlabeled data of great value. Likewise, we believe, our analysis of learning with very few labeled instances across datasets is the first attempt in quantifying the amount of labeled data required to learn an activity recognizer of decent quality. For self-supervised models, as earlier, we either kept the weights frozen or only fine-tune the last ConvC layer.

In Figure 3.4, we plot the average kappa of 10-independent runs as a function of the number of available training examples. For each run, we randomly sample desired training instances and train a model from scratch. Note that, we utilize the same instances for evaluating both

<table>
<thead>
<tr>
<th>Tasks</th>
<th>HHAR</th>
<th>Metrics</th>
<th>UniMiB</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jittered</td>
<td>0.1293</td>
<td>0.2060</td>
<td>0.1164</td>
<td>0.2002</td>
</tr>
<tr>
<td>Permutated</td>
<td>0.6357</td>
<td>0.6633</td>
<td>0.7571</td>
<td>0.7584</td>
</tr>
<tr>
<td>Rotated</td>
<td>0.6602</td>
<td>0.6641</td>
<td>0.5864</td>
<td>0.6067</td>
</tr>
<tr>
<td>Horizontal Flipped</td>
<td>0.7605</td>
<td>0.7868</td>
<td>0.5048</td>
<td>0.5365</td>
</tr>
<tr>
<td>Negation</td>
<td>0.1650</td>
<td>0.2866</td>
<td>0.7000</td>
<td>0.7048</td>
</tr>
<tr>
<td>Channel Shuffled</td>
<td>0.6407</td>
<td>0.6396</td>
<td>0.5847</td>
<td>0.5983</td>
</tr>
<tr>
<td>Time Warped</td>
<td>0.6909</td>
<td>0.6971</td>
<td>0.5352</td>
<td>0.5638</td>
</tr>
<tr>
<td>Scaled</td>
<td>0.6102</td>
<td>0.6186</td>
<td>0.4845</td>
<td>0.5292</td>
</tr>
<tr>
<td>All</td>
<td>0.7666</td>
<td>0.7770</td>
<td>0.7929</td>
<td>0.7954</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tasks</th>
<th>UCI HAR</th>
<th>Metrics</th>
<th>MobAct</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jittered</td>
<td>0.6680</td>
<td>0.6844</td>
<td>0.3993</td>
<td>0.4698</td>
</tr>
<tr>
<td>Permutated</td>
<td>0.8881</td>
<td>0.8914</td>
<td>0.8538</td>
<td>0.8581</td>
</tr>
<tr>
<td>Rotated</td>
<td>0.8624</td>
<td>0.8656</td>
<td>0.8926</td>
<td>0.8927</td>
</tr>
<tr>
<td>Horizontal Flipped</td>
<td>0.8854</td>
<td>0.8856</td>
<td>0.8640</td>
<td>0.8606</td>
</tr>
<tr>
<td>Negation</td>
<td>0.8042</td>
<td>0.8048</td>
<td>0.7386</td>
<td>0.7586</td>
</tr>
<tr>
<td>Channel Shuffled</td>
<td>0.7307</td>
<td>0.7479</td>
<td>0.8544</td>
<td>0.8529</td>
</tr>
<tr>
<td>Time Warped</td>
<td>0.6991</td>
<td>0.6994</td>
<td>0.8330</td>
<td>0.8248</td>
</tr>
<tr>
<td>Scaled</td>
<td>0.8456</td>
<td>0.8441</td>
<td>0.8655</td>
<td>0.8684</td>
</tr>
<tr>
<td>All</td>
<td>0.8987</td>
<td>0.9011</td>
<td>0.9060</td>
<td>0.9059</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tasks</th>
<th>WISDM</th>
<th>Metrics</th>
<th>MotionSense</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jittered</td>
<td>0.7512</td>
<td>0.7220</td>
<td>0.8096</td>
<td>0.8086</td>
</tr>
<tr>
<td>Permutated</td>
<td>0.8180</td>
<td>0.7996</td>
<td>0.8789</td>
<td>0.8763</td>
</tr>
<tr>
<td>Rotated</td>
<td>0.8074</td>
<td>0.7931</td>
<td>0.8756</td>
<td>0.8713</td>
</tr>
<tr>
<td>Horizontal Flipped</td>
<td>0.7619</td>
<td>0.7391</td>
<td>0.8345</td>
<td>0.8297</td>
</tr>
<tr>
<td>Negation</td>
<td>0.4729</td>
<td>0.5469</td>
<td>0.2747</td>
<td>0.3362</td>
</tr>
<tr>
<td>Channel Shuffled</td>
<td>0.7651</td>
<td>0.7440</td>
<td>0.8384</td>
<td>0.8381</td>
</tr>
<tr>
<td>Time Warped</td>
<td>0.7597</td>
<td>0.7322</td>
<td>0.8521</td>
<td>0.8507</td>
</tr>
<tr>
<td>Scaled</td>
<td>0.8011</td>
<td>0.7860</td>
<td>0.8641</td>
<td>0.8673</td>
</tr>
<tr>
<td>All</td>
<td>0.8384</td>
<td>0.8274</td>
<td>0.8684</td>
<td>0.8856</td>
</tr>
</tbody>
</table>

Figure 3.4: Comparison of individual self-supervised tasks with the multi-task setting. The TPN is pretrained for solving a particular task and the activity classifier is trained on-top of the learned features. We report the averaged results of evaluation metrics for 10 independent runs, where F, K, P, and R refer to F-score, Kappa, Precision and Recall, respectively. We observe that multi-task learning improves performance in all the cases with tasks such as Channel Shuffled, Permutated, and Rotated consistently performed better compared to other tasks across datasets.
supervised baseline and our proposed method. The fully-supervised baseline (blue curve) shows network performance when a model is trained only with the labeled data. The proposed self-supervised pretraining technique, in particular, the version with fine-tuning of the last ConvC layer, tremendously improved the performance. The difference in the performance between supervised and self-supervised feature learning is significant on MotionSense, UCI HAR, MobiAct, and HHAR datasets in low-data regime (i.e. with 2-10 labeled instances per class). More notably, we observe that pretraining helps more in a semi-supervised setting when the data are collected from a wide variety of devices; simulating a real-life setting. Finally, we highlight that a simple convolutional network is used in our experiments to show the feasibility of self-supervision from unlabeled data. We believe a deeper network trained on a bigger unlabeled dataset will further improve the quality of learned representations for the semi-supervised setting.

Evaluating Knowledge Transferability

We have shown that representations learned by the self-supervised TPN consistently achieve the best performance as compared to other unsupervised/supervised techniques and also in a semi-supervised setting. As we have utilized the unlabeled data from the same data source for self-supervised pretraining, a next logical question that arises is can we utilize a different (yet
similar) data source for self-supervised representation extraction and gain a performance improvement on a task of interest (also in a low-data regime)? In Table 3.3, we assess the performance of our unsupervised learned features across datasets and tasks by fine-tuning them on HAHR, UniMiB, UCI HAR, WISDM, and MotionSense datasets. For self-supervised feature learning, we utilized the unlabeled MobiAct dataset as it is collected from a diverse group of users that performed twelve activities, highest among other considered datasets both in terms of the number of users and activities. This makes MobiAct a suitable candidate to perform transfer learning as it encompasses all the activity classes in other datasets. Of course, we do not utilize activity labels in MobiAct for self-supervised representation learning. We begin by pretraining a network on MobiAct dataset and utilize the learned weights for initialization of an activity recognition model. Moreover, the latter model is trained in a fully-supervised manner on an entire training set of a particular dataset (e.g., UniMiB). In comparison with supervised training of the network (from scratch), the weights learned through our technique from a different and completely unlabeled data source improved the performance in all the cases. On WISDM and HHAR our results are 3 percentage points better in terms of kappa score. Similarly, on UniMiB we obtained 4 percentage points improvement over supervised model, i.e. kappa score increase from 0.781 to 0.821.

### Table 3.3: Task and Dataset Generalization: Quantifying the quality of transferred self-supervised network. We pretrain a TPN on MobiAct dataset with the proposed self-supervised approach. The classifier is added on the transferred model and trained in an end-to-end fashion on a particular activity recognition dataset. We chose MobiAct for transfer learning evaluation because of the large number of users and activity classes it covers. The reported results are averaged over 10 independent runs, where $P$, $R$, $F$, and $K$ refer to Precision, Recall, F-score, and Kappa, respectively.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Supervised (From Scratch)</th>
<th>Transfer (Self-Supervised)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>K</td>
</tr>
<tr>
<td>HHAR</td>
<td>0.7276±0.0297</td>
<td>0.6816±0.0371</td>
</tr>
<tr>
<td>UniMiB</td>
<td>0.8097±0.0248</td>
<td>0.7815±0.0299</td>
</tr>
<tr>
<td>UCI HAR</td>
<td>0.8981±0.0148</td>
<td>0.8789±0.0168</td>
</tr>
<tr>
<td>WISDM</td>
<td>0.8764±0.0168</td>
<td>0.8211±0.0258</td>
</tr>
<tr>
<td>MotionSense</td>
<td>0.9027±0.0085</td>
<td>0.8761±0.0111</td>
</tr>
</tbody>
</table>

Further, we determine the generalization ability in a low-data regime setting, i.e., when very few labeled data are attainable from an end-task of interest. We transfer self-supervised learned representations on the MobiAct dataset as initialization for an activity recognizer. The network is trained in a supervised manner on the available labeled instances of a particular dataset. Figure 3.6 shows average kappa score of 10-independent runs of a fully-supervised (learned from scratch) and transferred models for 2, 5, 10, 20, 50, and 100 labeled instances. For each training run, the desired instances are randomly sampled, and for both techniques, the same instances are used for learning the activity classifier. In the majority of the cases, transfer learning improves the recognition performance especially when the number of labeled instances per class are very few, i.e. between 2 to 10. In particular, on HHAR the performance of a model trained with weights transfer is slightly lower in low-data setting but improves significantly as the number of labeled data points increases. We think it may be because of the complex characteristics of the HHAR dataset as it is particularly collected to show heterogeneity of devices (and sensors) having varying sampling rates and its impact on the activity recognition performance.
mean similarity of top supervised network, we utilize singular vector canonical correlation analysis (SVCCA) [53]. Here, to better understand the similarity of the extracted representation from TPN and the networks and layers. It does so through identifying optimal linear relationships between two

Insights on Representational Similarity with Canonical Correlation

The SVCCA allows for a comparison of the learned distributed representations across different networks and layers. It does so through identifying optimal linear relationships between two sets of multidimensional variates (i.e., neuron activation vectors) arising from an underlying process (i.e., a neural network being trained on a specific task) [53]. Figure 3.7 provides a mean similarity of top 20 SVCCA correlation coefficients for all pairs of layers for a self-
supervised (trained to predict transformations) and a fully-supervised network. We averaged 20 coefficients as SVCCA implicitly assumes that all CCA vectors are equally crucial for the representations at a specific layer. However, there is plenty of evidence that high-performing deep networks do not utilize the entire dimensionality of a layer \[53, 60, 67\]. Due to this, averaging over all the coefficients underestimates the degree of representational similarity. To apply SVCCA, we train both the networks as explained earlier and produce activations of each layer. For a layer, where the number of neurons is larger than the layer in comparison, we randomly sample neuron activation vectors to have comparable dimensionality. In Figure 3.7 each grid entry represents a mean SVCCA similarity between two layers of different networks. We observe a high correlation among temporal convolution layers trained with two different methods across all the evaluated datasets. In particular, a strong grid-like structure emerges between the last layers of the networks, which is because those layers are learned from scratch with activity labeled data and result in identical representations.

**Figure 3.7**: CCA similarity between fully-supervised and self-supervised networks. We employ the SVCAA technique \[53\] to determine the representational similarity between model layers trained with our proposed approach and standard supervised setting. Each pane is a matrix of size layers × layers with each entry showing mean similarity (i.e., an average of top-20 correlation coefficients) between the two layers. Note that there is a strong relation between convolutional layers even though the self-supervised network is pretrained with unlabeled data; showing that features learned by our approach are very similar to those learned directly via supervised learning, with activity classes. Likewise, a grid-like structure appears between the last layers of the networks depicting high similarity as those layers are always (randomly initialized and) trained with activity labels.

**Visualizing Salient Regions**

To further understand the predictions produced by both models, we use saliency maps \[54\] for the highest-scoring class on randomly selected instances from the MotionSense dataset. Saliency maps highlight which time steps largely affect the output through computing gradient of the loss function with respect to each input time step. More formally, let \(x = [x_1, \ldots, x_N]\) be an accelerometer sample of length \(N\) and \(C_\theta(x)\) be the class probability produced by a
network $C_\theta()$. The saliency score of each input element $x_k$ indicating its influence on the prediction is calculated as:

$$S_k = \left| \frac{\partial L}{\partial x_k} \right|$$

where $L$ is the negative log-likelihood loss of an activity classification network for an input example $x$.

Figure 3.8 provides a saliency mapping of the same input produced by the two networks for a class with the highest score. To aid interpretability of the saliency score, we calculate a magnitude of each tri-axial accelerometer sample, effectively combining all three channels. The actual input is given in the top-most pane, the magnitudes with varying color intensity are shown in the bottom panes. The dark color illustrates the regions that contribute most to the network’s prediction. We observe that the saliency maps of both self-supervised and fully-supervised networks hint towards similar regions that are crucial for deciding on the class label.

Interestingly, for the Sitting class instance both network mainly focus on a smaller region of the input with slightly more variation in the values. We think it could be because one thing that a network learns is to find periodic variations in the signal (such as peaks and slopes). Hence, it pays more attention even to slightest fluctuation, but it decides on the Sitting label as the signal remains constant (before and after minor changes) which is an entirely different pattern as compared to the instances of other classes. This analysis further validates the point that our self-supervised network learns generalizable features for activity classification.

Visualization of High-Level Feature Space through t-SNE

t-SNE is a non-linear technique for exploring and visualizing multi-dimensional data. It approximates a low-dimensional manifold of a high-dimensional counterpart through minimizing Kullback-Leibler divergence between them with a gradient-based optimization method. More specifically, it maps multi-dimensional data onto a lower dimensional space and discovers patterns in the input through identifying clusters based on the similarity of the data points. Here, the activations from global max-pooling layers (of both self-supervised and fully-supervised networks) with 96 hidden units are projected on to a 2D space. Figure 3.9 provides the t-SNE embeddings showing high semantic relevance of the learned features for various activity classes. We notice that the self-supervised features largely correspond to those learned with the labeled activity data. Importantly, the clusters of data points across two feature learning strategies are similar, e.g. in UCI HAR, the activity classes like Upstairs, Downstairs and Walking are grouped. Likewise, in HHAR, the data points for Walking, Upstairs, and Downstairs are close-by as opposed to others in the embeddings of both networks. Finally, it is important to note that t-SNE is an unsupervised technique which does not use class labels; the activity labels are just used for final visualization.
Figure 3.8: Saliency maps [34] of randomly selected instances from MotionSense dataset. The input signal is illustrated in the top pane with the magnitude computed from the sample shown in the bottom panes for better interpretability. The strong colored intensities exhibit the regions that substantially affect the model predictions. The saliency mapping of both networks focus on similar input areas which shows that the self-supervised representations are useful for the end-task.

3.4 Related Work

Deep learning methods have been successfully used in several applications of ubiquitous computing, pervasive intelligence, health, and well-being [8, 68, 59, 73, 74, 72] and eliminate the need of hand-crafted feature engineering. Convolutional and recurrent neural networks have shown dominant performance in solving numerous high-level recognition tasks from temporal data such as activity detection and stress recognition [69, 73, 74]. In particular, CNNs are becoming increasingly popular in sequence (or time-series) modeling due to their ability of weight sharing, translation invariance, scale separation and localization of filters in space and time [62, 75]. In fact, (1D) temporal CNNs are now widely used in the area of HAR (see [73] for a detailed review), but the prior works are mostly concerned with supervised learning approaches. The training of deep networks requires a huge (carefully) curated dataset of labeled instances, which in several domains is infeasible due to required manual labeling effort or can only be possible on a small-scale in a controlled lab environment. This inherent limitation of the fully-supervised learning paradigm emphasizes the importance of unsupervised learning to leverage a large amount of unlabeled data for representation learning [33] that can be easily acquired in a real-world setting.
Unsupervised learning has been well-studied in the literature over the past years. Before the era of end-to-end learning, manual feature design strategies [76] such as those that employ statistical measures have been used with clustering algorithms to discover a latent group of activities [77]. Although deep learning techniques have almost entirely replaced hand-crafted feature extraction with directly learning rich features from data, representation learning still stands as a fundamental problem in machine learning (see [29] for an in-depth review). The classical approaches for unsupervised learning include autoencoders [78], restricted Boltzmann machines [79], and convolutional deep belief networks [80]. Another emerging line of research for unsupervised feature learning (also studied in this work), which has shown promising results and does not require manual annotations, is self-supervised learning [81, 82, 83].
These methods exploit the inherent structure of the data to acquire a supervisory signal for solving a pretext task with reliable and widely used supervised learning schemes.

Self-supervision has been actively studied recently in the vision domain, and several surrogate tasks have been proposed for learning representations from static images, videos, sound, and in robotics [35, 36, 37, 39, 41, 46, 47, 48, 49, 50, 84, 85, 86, 87]. For example, in images and videos, spatial and temporal contexts, respectively, provide forms of rich supervision to learn features. Similarly, colorization of gray-scale images [36, 46], rotation classification [35], odd sequence detection [48], frame order prediction [47], learning the arrow of time [46], audio-visual correspondence [89, 90] and synchronization [87, 88] are some of the recently explored directions of self-supervised techniques. Furthermore, multiple such tasks are utilized together in a multi-task learning setting for solving diverse visual recognition problems [45]. These self-supervised learning paradigms have shown to extract high-level representations that are on par with those acquired through fully-supervised pretraining techniques (e.g., with ImageNet labels) and they tremendously help with transfer and semi-supervised learning scenarios. Inspired from this research direction, we explore multi-task self-supervision for learning representations from sensory data through utilizing transformations of the signals.

Some earlier works on time-series analysis have explored transformations to exploit invariances either through architectural modifications (to automatically learn task-relevant variations) or less commonly with augmentation and synthesis. In [51] task-specific transformations (such as added noise and rotation) are applied to wearable sensor data to augment and improve the performance of Parkinson’s disease monitoring systems. Saeed et al. [23] utilized an adversarial autoencoder for class-conditional (multimodal) synthetic data generation for the behavioral context in a real-life setting. Moreover, Oh et al. [89] focused on learning invariances directly from clinical time-series data with specialized neural network architecture. Razavian et al. [92] used convolution layers of varying size filters to capture different resolutions of temporal patterns. Similarly, through additional pre-processing of the original data Cui et al. [91] used transformed signals as extra channels to the model for learning multiscale features. To summarize, these works are geared towards learning supervised networks for specific tasks through exploiting invariances, but they do not address the topics of semi-supervised and unsupervised learning.

To the best of our knowledge, the work presented here is the first attempt of self-supervision for sensor representation learning, in particular for HAR. Our work differs from the aforementioned works in several ways as we learn representations with self-supervision from completely unlabeled data and without using any specialized architecture. We show that when training a CNN to predict generally known (time-series) transformations [51, 52] as a surrogate task, the model can learn features that are on a par with a fully-supervised network and far better than unsupervised pretraining with an autoencoder. We also demonstrate that the learned representations from a different (but related) unlabeled data source can be successfully transferred to improve the performance of diverse tasks even in the case of semi-supervised learning. In terms of transfer learning, our approach also differs significantly from some earlier attempts [92, 93] that were concerned with features transferability from a fully-supervised model learned from inertial measurement units data, as our approach utilizes widely available smartphones and does not require labeled data. Finally, the proposed technique is also different from previously studied unsupervised pretraining methods such as autoencoders [94].
restricted Boltzmann machines \cite{95} and sparse coding \cite{96} as we employ an end-to-end (self) supervised learning paradigm on multiple surrogate tasks to extract features.

### 3.5 Conclusion

We present a novel approach for self-supervised sensor representation learning from unlabeled data with a focus on smartphone-based human activity recognition (HAR). We train a multi-task temporal convolutional network to recognize potential transformations that may have been applied to the raw input signal. Despite the simplicity of the proposed self-supervised technique (and the network architecture), we show that it enables the convolutional model to learn high-level features that are useful for the end-task of HAR. We exhaustively evaluate our approach under unsupervised learning, semi-supervised learning and transfer learning settings on several publicly available datasets. The performance we achieve is consistently superior to or comparable with fully-supervised methods, and it is significantly better than traditional unsupervised learning methods such as an autoencoder. Specifically, our self-supervised framework drastically improved the detection rate under semi-supervised learning setting, i.e., when very few labeled instances are available for learning. Likewise, the transferred features learned from a different but related unlabeled dataset (\textit{MobiAct} in our case), further improves the performance in comparison with merely training a model from scratch. Notably, these transferred representations even boost the performance of an activity recognizer in semi-supervised learning from a dataset (or task) of interest. Finally, canonical correlation analysis, saliency mapping, and t-SNE visualizations show that the representations of the self-supervised network are very similar to those learned by a fully-supervised model that is trained in an end-to-end fashion with activity labels. We believe that, through utilizing more sophisticated layers and deep architectures, the presented approach can further reduce the gap between unsupervised and supervised feature learning.

---

Various icons used in the figures are created by Anuar Zhumaev, Korokoro, Gregor Cresnar, Becris, Hea Poh Lin, AdbA Icons, Universal Icons, and Baboon designs from the Noun Project.
## Appendix

### Table 3.4: Evaluating self-supervised representation with (user-split based) 5-folds cross-validation for activity recognition. We perform this assessment based on user-split of the data with no overlap between training and test sets i.e. distinct users' data are used for training and testing of the models. The reported results are averaged over 5-folds.

(a) HHAR

<table>
<thead>
<tr>
<th>Method</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Init.</td>
<td>0.3420 ± 0.1395</td>
<td>0.302 ± 0.0465</td>
<td>0.202 ± 0.033</td>
<td>0.161 ± 0.0336</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.8401 ± 0.0349</td>
<td>0.816 ± 0.0518</td>
<td>0.807 ± 0.0612</td>
<td>0.778 ± 0.0624</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.6068 ± 0.2149</td>
<td>0.594 ± 0.1858</td>
<td>0.147 ± 0.2182</td>
<td>0.519 ± 0.2208</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.8454 ± 0.0378</td>
<td>0.823 ± 0.0462</td>
<td>0.813 ± 0.053</td>
<td>0.788 ± 0.0536</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.8439 ± 0.0753</td>
<td>0.810 ± 0.1004</td>
<td>0.803 ± 0.1072</td>
<td>0.771 ± 0.1204</td>
</tr>
</tbody>
</table>

(b) UniMiB

<table>
<thead>
<tr>
<th>Method</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Init.</td>
<td>0.416 ± 0.0307</td>
<td>0.361 ± 0.0503</td>
<td>0.287 ± 0.0722</td>
<td>0.228 ± 0.0622</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.805 ± 0.0095</td>
<td>0.789 ± 0.0153</td>
<td>0.786 ± 0.0165</td>
<td>0.759 ± 0.0181</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.598 ± 0.0313</td>
<td>0.574 ± 0.0192</td>
<td>0.494 ± 0.0227</td>
<td>0.309 ± 0.0242</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.777 ± 0.0211</td>
<td>0.761 ± 0.0191</td>
<td>0.757 ± 0.0208</td>
<td>0.724 ± 0.0218</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.8396 ± 0.0226</td>
<td>0.831 ± 0.0269</td>
<td>0.828 ± 0.0283</td>
<td>0.804 ± 0.0309</td>
</tr>
</tbody>
</table>

(c) UCI HAR

<table>
<thead>
<tr>
<th>Method</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Init.</td>
<td>0.614 ± 0.1845</td>
<td>0.301 ± 0.0999</td>
<td>0.429 ± 0.1141</td>
<td>0.387 ± 0.1277</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.9068 ± 0.0332</td>
<td>0.903 ± 0.0166</td>
<td>0.903 ± 0.0177</td>
<td>0.882 ± 0.0446</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.874 ± 0.0167</td>
<td>0.848 ± 0.0604</td>
<td>0.846 ± 0.0642</td>
<td>0.816 ± 0.0734</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.9054 ± 0.0273</td>
<td>0.891 ± 0.0388</td>
<td>0.889 ± 0.0444</td>
<td>0.868 ± 0.0472</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.9121 ± 0.0403</td>
<td>0.908 ± 0.0473</td>
<td>0.904 ± 0.049</td>
<td>0.888 ± 0.0573</td>
</tr>
</tbody>
</table>

(d) MobiAct

<table>
<thead>
<tr>
<th>Method</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Init.</td>
<td>0.481 ± 0.1405</td>
<td>0.382 ± 0.0417</td>
<td>0.306 ± 0.0422</td>
<td>0.191 ± 0.0435</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.912 ± 0.0118</td>
<td>0.903 ± 0.0116</td>
<td>0.904 ± 0.0153</td>
<td>0.876 ± 0.0398</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.748 ± 0.0402</td>
<td>0.749 ± 0.0398</td>
<td>0.732 ± 0.0408</td>
<td>0.669 ± 0.0542</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.8877 ± 0.0128</td>
<td>0.885 ± 0.0133</td>
<td>0.887 ± 0.013</td>
<td>0.882 ± 0.0216</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.918 ± 0.0056</td>
<td>0.920 ± 0.0077</td>
<td>0.917 ± 0.0067</td>
<td>0.888 ± 0.0101</td>
</tr>
</tbody>
</table>

(e) WISDM

<table>
<thead>
<tr>
<th>Method</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Init.</td>
<td>0.607 ± 0.0555</td>
<td>0.323 ± 0.0958</td>
<td>0.331 ± 0.0885</td>
<td>0.182 ± 0.0623</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.893 ± 0.0015</td>
<td>0.879 ± 0.0031</td>
<td>0.886 ± 0.00297</td>
<td>0.839 ± 0.0042</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.687 ± 0.0203</td>
<td>0.686 ± 0.0215</td>
<td>0.677 ± 0.0079</td>
<td>0.557 ± 0.0123</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.871 ± 0.0189</td>
<td>0.836 ± 0.0656</td>
<td>0.846 ± 0.0556</td>
<td>0.780 ± 0.0826</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.884 ± 0.0218</td>
<td>0.851 ± 0.048</td>
<td>0.859 ± 0.0409</td>
<td>0.799 ± 0.0645</td>
</tr>
</tbody>
</table>

(f) MotionSense

<table>
<thead>
<tr>
<th>Method</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Init.</td>
<td>0.515 ± 0.0997</td>
<td>0.427 ± 0.1066</td>
<td>0.345 ± 0.1069</td>
<td>0.271 ± 0.1246</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.933 ± 0.0144</td>
<td>0.921 ± 0.0186</td>
<td>0.924 ± 0.0172</td>
<td>0.904 ± 0.0231</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.870 ± 0.0654</td>
<td>0.819 ± 0.0734</td>
<td>0.818 ± 0.075</td>
<td>0.776 ± 0.0892</td>
</tr>
<tr>
<td>Self-Supervised</td>
<td>0.926 ± 0.0189</td>
<td>0.917 ± 0.0195</td>
<td>0.918 ± 0.019</td>
<td>0.897 ± 0.0244</td>
</tr>
<tr>
<td>Self-Supervised (FT)</td>
<td>0.947 ± 0.0174</td>
<td>0.937 ± 0.028</td>
<td>0.939 ± 0.0254</td>
<td>0.922 ± 0.0345</td>
</tr>
</tbody>
</table>
Figure 3.10: Convergence analysis of transformation recognition tasks. We plot the kappa score of self-supervised tasks (i.e. transformation prediction) as a function of the training epochs. In order to produce the kappa curves, the TPN model's snapshot is saved every second epoch until the defined number of training epochs. For each saved network, we evaluated its performance on the self-supervised data obtained through processing the corresponding test sets. Note that the TPN never sees a test set data in any way during its learning phase.
Figure 3.11: Weighted F-score: Generalization of the self-supervised learned features under semi-supervised setting. The reported results are averaged over 10 independent runs for each of the evaluated approaches, for more details see Section 3.3.3.

Figure 3.12: Weighted F-score: Assessment of the transferred self-supervised learned features from a different but related dataset (MobiAct) under semi-supervised setting. The reported results are averaged over 10 independent runs for each of the evaluated approaches, for more details see Section 3.3.3.
Chapter 4

Sense and Learn: Self-Supervision for Omnipresent Sensors

This chapter is based on the material from our paper Sense and Learn: Self-Supervision for Omnipresent Sensors under review. This work was done during an internship at Google Research, Zurich under the supervision of Victor Ungureanu and Beat Gfeller.

4.1 Introduction

In the previous chapter, we show that the emerging paradigm of self-supervised learning offers an effective way for learning semantically-meaningful representations from accelerometer data that can be used for solving a diverse set of human activity recognition tasks. In this chapter, we provide a suite of simple yet effective auxiliary tasks that are broadly useful for a spectrum of downstream tasks and learning representations from different input modalities.

The self-supervised approaches exploit the inherent structure of the input to derive a supervisory signal. The central idea is to define a pretext task, for which annotations can be acquired without human involvement (directly from the raw data) and can be solved using some form of unsupervised learning techniques. This intriguing property essentially renders a deep sensing model, that is developed based on the earlier described principle of “self-learning” in nature: a system that can be trained continuously on massive, readily-accessible data in an unsupervised manner [3]. However, in this case, the challenge lies in designing complex auxiliary tasks that can force the deep neural network to capture meaningful features of the input, while avoiding shortcuts [97] (i.e., simple unintended ways to trivially solve the auxiliary task without learning anything useful that generalizes beyond the auxiliary task).

In this chapter, we present a principled framework for self-supervised learning of multisensor representations from unlabeled data. Our objective is to have numerous tasks, with each
perhaps imposing a distinct prior on to the learning process, resulting in varying quality features that may differ across sensing datasets. Specifically, as proxy tasks and modalities could be of more or less relevance to the downstream task’s performance, it is essential to explore and compare several pretext tasks so as to discover the ones with better generalization properties. The broad aim is to have many auxiliary tasks in a user’s toolbox such that, either experimentally or based on prior knowledge, a relevant task can be selected for training deep models. Particularly, the objective is to have proxy tasks that enable learning of representations invariant to several input deformations that commonly arise in the temporal data, such as sensor noise and sampling-rate disparities, or that can be used jointly in a multi-task learning setting.

To this end, we develop eight novel auxiliary tasks that intrinsically obtain supervision from the unlabeled input signals to learn general-purpose features with a temporal convolutional network, such that the pretrained model generalizes well to the end tasks.

Our approach comprises of pretraining a network through self-supervision with unlabeled data so that it captures high-level semantics and can be used either as a feature extractor or utilized as initialization for making successive tasks of interest easier to solve with few labeled data. To develop the auxiliary tasks, we take advantage of the synchronized multisensor (or multimodal) data as it belongs to the same underlying phenomena and we exploit it to create proxy tasks that can capture broadly useful features. Specifically, it can substantially help in learning powerful representations of each modality, and ultimately learn more abstract concepts in a joint-embedding space. Thus, we use a multi-stream neural network architecture to solve proxy tasks so that it can learn modality-specific features with a distinct encoder per modality and subsequently learn a shared embedding space with a modality-agnostic encoder. The fundamental structure of our framework is illustrated in Figure 4.1. We adopt a small model architecture in this work to highlight a) effectiveness of self-supervised tasks (i.e. improvement is not due to complex architecture) and b) potential of deployment on resource-constrained devices for training and inference. The further investigation of resource usage is beyond the scope of our work and we leave it for future work.

We demonstrate that a relatively straightforward suite of auxiliary tasks results in meaningful features for diverse problems, including: activity recognition, stress detection, sleep stage scoring, and WiFi sensing. First, we show that the self-supervised representations are highly competitive with those learned with a fully-supervised model, by training a linear classifier on top of the frozen network, as it is a standard evaluation protocol for assessing the quality of self-supervised tasks [42, 98]. Second, we explore fine-tuning the last layer of the encoder to gain further improvements over training from scratch. Third, we investigate the effectiveness of the learned representations in low-data regime². Using our pretrained network as initialization, we achieve a significant performance boost with as little as 5 to 10 labeled instances per class, which clearly highlights the value of self-supervised learning. Lastly, we evaluate the transferability of the features across related datasets/tasks to show the generality of our method in an unsupervised transfer learning setting.

In summary, our main contributions are as follows:

- We propose Sense and Learn, a generalized self-supervised learning framework compris-

---

¹i.e. leveraging representations from intermediate layers of the deep neural network
²or in a semi-supervised setting
Figure 4.1: Illustration of our Sense and Learn representation learning framework. A deep neural network is pretrained with self-supervision using input modalities from large unlabeled sensory data, such as inertial measurements (or electroencephalogram, heart rate, and channel state information). The learned network can then be utilized as a feature-extractor or initialization for rapidly solving downstream tasks of interest with few labeled data.

- We extensively evaluate our self-supervised tasks on various problems (e.g. sleep stage scoring, activity recognition, stress detection, and WiFi sensing) and learning settings (i.e. transfer and semi-supervised) to significantly improve the data efficiency or lower the requirement of collecting large-scale labeled datasets.

- Our results demonstrate that self-supervision provides an effective initialization of the network (and powerful embeddings) that improves performance significantly with minimal fine-tuning, and works well in a low-data regime, which is of high importance for real-world use cases.

- The developed auxiliary tasks require an equivalent computational cost as standard supervised learning and has fewer parameters than autoencoding methods, but provide better generalization with greatly improved sample efficiency.

- We utilize a small network architecture to show the capability of self-supervision and its prospective usage on resource-constrained devices in future. In particular, the majority of our proposed tasks are designed around the principle that self-supervised data generation should not be computationally expensive; thus, it can be readily used for on-device learning.
4.2 Approach

In this section, we begin with a motivation and an overview of our self-supervised framework for learning sensory representations. Next, we provide a formalization of the auxiliary tasks and discuss an end-to-end approach for multi-modal learning. Subsequently, we describe the network architecture design, its implementation, and the optimization procedure.

4.2.1 Motivation and Overview

The key insight behind our technique is that the self-supervised pretraining acts as a prior that can give rise to varying quality representations that encode underlying signal semantics at different levels, which may or may not be useful for a downstream-task of interest. Therefore, it is vital to employ multiple auxiliary tasks to discover the suitable inductive bias necessary to obtain optimal performance on the desired end-task. This intuition is important considering that the time-series (or sensory) data shows peculiar characteristics (e.g. signal-to-noise ratio, amplitude variances, and sampling rates) depending on the nature of phenomena being recorded. Likewise, there should be an array of tasks to choose from depending on the learning problem and device type (e.g. available resources, sensor types etc.). Importantly, we want the self-supervised model to learn generic features rather than focusing on low-level input details, as a pretrained network has to provide a strong initialization for learning with limited labeled data and generalize to other related tasks. Thus, instead of relying on a single auxiliary task, we learn latent representations with a broad set of tasks based on different objective functions.

We propose a generalized framework comprising of eight pretext tasks that can be used to learn features from heterogeneous multisensor data. To achieve this, we utilize a temporal convolutional network (TCN) $F_{\theta}$ with a distinct encoder $e_m$ for each input modality $I_m$ and a shared encoder $e_s$ for multi-modal representation extraction. We choose to use TCN as an embedding network for sequence modeling due to its effectiveness in capturing long-term dependencies and parallelizability at a significantly lower cost than recurrent networks [73]. For every learning problem, we consider unlabeled multisensor (or multi-modal) data $D = \{ (u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n) \}$ consisting of $N$ examples. Here, $u_n$ and $v_n$ denote the samples of different modalities (e.g. accelerometer and gyroscope) of the $n^{th}$ example. The defined pretext tasks exploit the inherent properties of the data to obtain supervision from the input pairs without requiring any manual annotation to optimize a certain loss function. Specifically, each surrogate task employs its own loss function $L_t$ for learning $F_{\theta}$ differently. For instance, an input reconstruction task employs mean-square error loss, while another task, concerning the detection of odd segments within a signal, uses negative log-likelihood; we discuss these in detail in the subsequent section. At a high-level, we utilize these objectives as necessary proxies for sensory representation learning without focusing on how well the model performs on them but on an end-task. After pretraining, $F_{\theta}$ captures a joint embedding space of the inputs, and thus it can be utilized either as a feature extractor or as initialization for rapidly learning to solve other problems.
4.2.2 Suite of Pretext Tasks

In order to achieve self-supervised learning of disentangled semantic representations from unannotated sensory data, we develop several auxiliary tasks for the network. To solve these tasks, we assume \( \mathbf{u} = \{u_1, u_2, \ldots, u_l\} \) and \( \mathbf{v} = \{v_1, v_2, \ldots, v_l\} \) denote multi-channel signals of length \( l \) from different modalities (e.g. accelerometer and gyroscope). Let \( z_u = e_u(\mathbf{u}) \) and \( z_v = e_v(\mathbf{v}) \) be the low-dimensional embeddings computed from the corresponding input signals with respective encoders. Likewise, \( z_s = e_s(e_u(\mathbf{u}), e_v(\mathbf{v})) \) provides a shared embedding of the inputs through fusion that may capture more abstract features. A high-level illustration of the self-supervised learning procedure is shown in Figure 4.1. A self-supervised data generation module produces annotated input from unlabeled multisensor data for learning \( F_\theta \). We utilize this formulation to define the self-supervised objectives in the following subsections.

Blend Detection

To take advantage of the multisensor signals, we define an auxiliary task of detecting input blending as a multi-class classification problem. Given an unlabeled input batch \( B = \bigcup_{i=1}^{|B|} \{(\mathbf{u}, \mathbf{v})\}_i \), we generate three types of instances. First, we keep the original samples as belonging to a class \( c_a \). Second, we perform a weighted blending of an instance from one modality with another randomly selected example from a different modality as class \( c_b \). Third and last, the instances of the same modalities are blended to have instances for a class \( c_c \). The blending weight \( \mu \) is sampled from a uniform distribution, i.e. \( \mu \sim \mathcal{U}(0, 1) \). The network is trained with a negative log-likelihood loss \( L_{NL} \) for learning to differentiate between examples of blended and clean classes \( (y_k) \) on the entire training set \( D_{train} \):

\[
L_{NL} = -\frac{1}{K} \sum_{k=1}^{K} y_k \times \log(F_\theta(\mathbf{u}, \mathbf{v}))
\]  

(4.1)

Fusion Magnitude Prediction

We create a variant of the earlier defined task that uses a similar data generation strategy but differs fundamentally in terms of the objective it optimizes. Here, we task the network with predicting the magnitude \( \mu \), which defines the blending (or weighting) factor of the signals. We assign \( \mu = 0 \) to the clean examples, while assigning weight \( \mu \sim \mathcal{U}(0, 1) \) to the blended examples, as earlier. In this case, a natural choice is to adopt mean-square loss as learning objective. However, we experimentally discovered that utilizing binary cross-entropy with a logistic function in the network’s output layer results in better generalization; thus the network is trained to minimize the following loss \( L_{BCE} \) for each input modality:

\[
L_{BCE} = -(y \times \log(F_\theta(\mathbf{u}, \mathbf{v})) + (1 - y) \times \log(1 - F_\theta(\mathbf{u}, \mathbf{v})))
\]  

(4.2)
Feature Prediction from Masked Window

It is observed that networks which try to reconstruct every bit of the input waste capacity on modeling low-level details [42]. Instead, in this auxiliary task we ask the network to approximate summary statistics of a masked temporal segment within a signal. To generate the data, we randomly sample the segment length \( s_l \sim \mathcal{U}(n_{\text{low}}, n_{\text{high}}) \) and starting point \( s_p \sim \mathcal{U}(0, l - s_l) \). From the selected subsequence, we extract 8 basic features: mean, standard deviation, maximum, minimum, median, kurtosis, skewness, number of peaks; and then mask the segment with zeros. The multi-head network is trained with Huber loss \( \mathcal{L}_{HL} \) to predict statistics of a missing sequence as:

\[
\mathcal{L}_{HL} = \begin{cases} 
\frac{1}{2} \times o^2, & \text{if } |o| \leq \delta \\
\delta \times (|o| - \frac{\delta}{2}), & \text{otherwise } |o| > \delta 
\end{cases}, \quad \text{where } o = F_{\theta}(u, v) - y \tag{4.3}
\]

Transformation Recognition

The signal transformation recognition is presented in [14] and in Chapter 3 as an auxiliary task, where it is posed as a set of binary classification problems solved with a multi-task network to determine whether a signal is a transformed version or not. Here, we simplify the problem formulation and treat the task as multi-class classification, to learn a network that can directly recognize the applied transformation on an input from one out of \( K \) classes. The benefits of our formulation are that it does not require specifying weights for task-specific losses and the network can be efficiently optimized with categorical cross-entropy objective \( \mathcal{L}_{NL} \). Another key difference is that we address the problem of learning from multi-modal data as opposed to a unimodal signal. To produce task-specific data, we generate transformed versions of each instance utilizing eight transformation functions: permutation, channel shuffle, timewarp, scale, noise, rotation, flip, negation), and an identity operation while assigning the function type as the corresponding class. During network training, we feed a batch of data consisting of examples for all the classes (inclusive of originals) and optimize a separate loss function for each input signal.

Temporal Shift Prediction

This conceptually straight-forward task consists of estimating the number of steps by which the samples are circularly-shifted in their temporal dimension. We pose this problem such that it can be treated either as a classification or as a regression task. We define a range of shift intervals, depending on the input resolution. For instance, in the activity recognition task, the considered ranges are: \([(0, 5), (6, 10), (11, 20), (21, 50), (51, 100), (101, 200), (201, 300)]\). For producing shifted inputs, we first select a pair at random from the defined ranges, and second we sample a shifting factor within the defined boundary of the selected range. Last, we temporally shift the values of an input segment with the sampled factor. The network can be trained to predict either the range index (treating each entry as a class, with 7 classes in total) or regress the factor. In our experiments, we notice that solving it as a regression problem results in better generalization on the end-
task. Thus, the network is trained by minimizing mean-squared error loss $L_{MSE}$ for each sensing modality:

$$L_{MSE} = \frac{1}{m} \sum_{i=1}^{m} (y_i - F_\theta(u, v))^2$$  \hspace{1cm} (4.4)

### Modality Denoising

This task’s objective is to decompose a signal for obtaining a clean target through input reconstruction, i.e. isolating the mixed noise. It is similar in spirit to source separation in audio \cite{99,100} and a denoising autoencoder \cite{101}. The fundamental intuition here is that if the network is tasked to reconstruct the original input from corrupted or mixed modality signals, then it forces the network to identify core signal characteristics while learning usable representations in the process. In our case, instead of mixing arbitrary noise, we exploit the availability of multisensor data to generate instances that might be of sufficient difficulty for the network to denoise. Specifically, we utilize a weighted blending operation $u \times (1 - \mu) + v \times \mu$ to mix instances of different modalities, i.e. we produce samples through combining the clean instances of accelerometer with gyroscope and vice versa while keeping the original samples as additional data points. The encoder-decoder network is trained end-to-end to minimize the mean-square error loss $L_{MSE}$ between ground truth and corrupted input pairs.

### Odd Segment Recognition

The goal of odd segment recognition is to identify the unrelated subsegment that does not belong to the input under consideration, where the rest of the sequences are in the correct order. The high-level idea behind the task is that if the network can spot artifacts in the signal, it should then also learn about useful input features. Similar ideas have been employed in video representation learning \cite{48} to spot invalid frame detection in video. There are multiple ways to generate examples with odd subsegments; we approach it as an input consisting of an irregular segment of fixed length $s_o$ that is selected randomly from a different input modality. To generate proxy task examples, we begin with splitting an instance into equal-length sequences (e.g. of length 100). Then, 2 sequences from different modalities are randomly selected, that are either directly swapped or blended before applying a substitution operation. The index of the interchanged slices is used as the class, where valid inputs are assigned a distinct class. The network is asked to predict an index $id$ of the odd sequence in each input modality. For this task, we minimize a categorical cross-entropy loss $L_{NL}$ to train a multi-head network.

### Metric Learning with Triplet Loss

As we are interested in learning from multisensor data, we take advantage of multiple input modalities to formulate a metric learning objective. For this purpose, we utilize a symmetric triplet loss \cite{102}, which encourages the representations of similar inputs but different modalities to be closer, while the representations of dissimilar inputs to be further apart. To optimize the specified loss, we need to generate input triplets consisting of an anchor, which can be an original instance, a positive sample that should be related (i.e. provides a complementary
Figure 4.2: A multistream neural network architecture for learning representations from multiple sensory inputs. A distinct stream (with an identical architecture) is used for each modality, as depicted on the right.

view of the input) to the anchor, and a negative sample which must be entirely different from the former pair. The loss then minimizes the distance between the anchor and the positive samples, while maximizing the distance of the negative samples from the anchor and the positive samples. For metric learning under this formulation, we generate the examples as follows: the actual instances are treated as anchors, and positive instances are generated by applying selected transformations at random on each anchor; whereas the negative instances are sampled from a different modality (i.e. for accelerometer, we treat samples from gyroscope as negatives). We then optimize $F_\theta$ with triplet loss $L_{TL}$ to produce a smaller distance on associated samples and a more considerable distance on unrelated ones:

$$L_{TL} = \max[0, \ D(z_a, z_p) - \frac{1}{2} \times (D(z_a, z_n) + D(z_p, z_n)) + \alpha], \quad (4.5)$$

where $z_a, z_p, z_n$ are the embeddings of anchor, positive and negative samples respectively, $\alpha$ represents the distance margin, and $D$ denotes squared-euclidean distance.

### 4.2.3 Network Architecture Design

We implement the learning network $F_\theta$ as a multi-stream temporal convolutional model (TCN). The part of the motivation to use TCN came from [75] where it has been shown that convolutional networks perform remarkably well on sequence modeling tasks. Likewise, they have a low footprint for training and inference as compared to other methods and can be pruned easily to further compress the network [103]. Our model consists of a distinct learning stream for each input to extract modality-specific features. The subnetworks share the same architecture, which is followed by a modality-agnostic network that fuses and learns a shared representation from the multimodal input. Jointly, we refer to these modules as encoder $e$, which is embedded within $F_\theta$. Importantly, we add an extra block connected to $e$, which is discarded after self-supervised pretraining. The intuition behind this strategy is that the model’s last layers capture features that are primarily task-specific and do not generalize well on the end-task of interest. Therefore, the additional layers allow the base encoder to capture more generic features, while solving the auxiliary tasks.
Figure 4.2 illustrates the architecture design by precisely highlighting these main building blocks. The modality-specific encoder consists of three 1D convolutional layers with 32, 64, and 96 feature maps and a kernel size of 24, 16, and 8, respectively. The max-pooling layer, with a pooling size of 4 and a stride of 2, is added after the initial convolutional layers. A dropout is used with a rate of 0.1 at the end of the block. The shared encoder consists of a single convolutional layer with 128 feature maps and a kernel size of 4, which takes concatenated features as input. The supplementary layers in the pretraining block consist of a convolutional layer with 64 feature maps and a kernel size of 4 and a dense layer having 512 hidden units. Importantly, a separate output layer is used for each input modality for all the surrogate tasks except ‘sensor blend,’ which, based on its formulation, does not require this. Likewise, we use global pooling as the last layer in the representation learning network that aggregates discriminative features. L2 regularization with a rate of 0.0001 is applied to the weights of all the layers to avoid overfitting. Moreover, we employ SELU as non-linearity except on the output layer; the network is trained with a learning rate of 0.0001 for a maximum of 30 epochs unless stated otherwise.

We utilize a fixed network architecture for all the considered tasks (both auxiliary and downstream), the intuition behind this choice being threefold. Firstly, we want to minimize the architectural differences to discover the true potential of self-supervision, i.e. it can be used with minimal effort on architecture tuning to extract semantic representations across diverse datasets. Secondly, our aim is to show that self-supervision has a huge prospect to be utilized for on-device learning. Having a smaller architecture and given the annotation-free nature of the proposed approach opens several exciting avenues in learning and inference with devices having limited processing capabilities. However, the further investigation of this is beyond the scope of our work and we leave it for future work. Lastly, our multi-modal architectural specification provides the flexibility to incorporate other modalities effortlessly. Furthermore, we highlight that in this work our focus is on individual task proposal and evaluation, but the framework can be used for jointly solving proxy tasks (i.e. in multi-task learning setting) as they share the same architecture, but differ fundamentally in terms of the loss function being optimized.

Given an unlabeled data $D_U$ and a specified auxiliary task $A_t$, we optimize $F_\theta$ with task-specific data that is generated on-the-fly, as described in the preceding section. Once pretraining converges, the layers specific to self-supervised learning are discarded, and the encoder $e$ is saved. Then, the second round of training on a downstream task of interest begins with labeled data $D_L$. Depending on the evaluation criteria, the following can be done: a) the network is either kept frozen and used as a generic feature extractor for learning a linear classifier, b) the modality-agnostic encoder $e_s$ is fine-tuned during learning an end-task, or c) the self-supervised network is used as initialization for rapidly solving the final-task, e.g., fine-tuning a model with little labeled data. The encoder network shown in Figure 4.2 represents the module that is kept frozen, while depending on the learning setting the shared layers are further fine-tuned.

$^3$logistic regression
4.3 Experiments

We perform a comprehensive evaluation of our framework on four different application domains: a) activity recognition, b) sleep-stage scoring, c) stress detection, and d) WiFi sensing. For every area, we train the self-supervised networks with each proposed task and determine the quality of the learned representation with either a linear classifier or by fine-tuning with few labeled instances. Furthermore, we also examine the knowledge transferability between related datasets. In the following, we describe the utilized datasets, pre-processing steps, and assessment strategy, including the baselines.

4.3.1 Datasets

We assess the performance of Sense and Learn on 8 publicly available multisensor datasets from diverse domains. The brief description of each utilized data source is provided below, with Table 4.1 summarizing their major characteristics.

Table 4.1: Key characteristics of the datasets used in the experiments.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Subjects</th>
<th>#Classes</th>
<th>Task</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHAR</td>
<td>9</td>
<td>6</td>
<td>Activity/Context Recognition</td>
<td>Accelerometer &amp; Gyroscope</td>
</tr>
<tr>
<td>MobiAct</td>
<td>66</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MotionSense</td>
<td>24</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCI HAR</td>
<td>30</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAPT</td>
<td>30</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sleep-EDF</td>
<td>20</td>
<td>5</td>
<td>Sleep Stage Scoring</td>
<td>EEG &amp; EOG</td>
</tr>
<tr>
<td>MIT DriverDb</td>
<td>17</td>
<td>2</td>
<td>Stress Detection</td>
<td>Heart Rate &amp; Skin Conductance</td>
</tr>
<tr>
<td>WiFi CSI</td>
<td>6</td>
<td>7</td>
<td>Activity (Behavior) Recognition</td>
<td>CSI Amplitude</td>
</tr>
</tbody>
</table>

Activity Recognition

For smartphone-based human activity recognition, we select 5 datasets containing accelerometer and gyroscope signals, namely: HHAR, MobiAct, UCI HAR, MotionSense, and HAPT. The Heterogeneity Human Activity Recognition (HHAR) dataset \[56\] is collected from 9 participants, each performing 6 basic activities (i.e. sitting, standing, walking, stairs-up, stairs-down and biking) for 5 minutes. A broad range of devices is used for the systematic analysis of sensor, device, and workload-specific heterogeneities across manufacturers. Specifically, each user carried 8 smartphones on different body locations that were selected from a pool of 36 devices of different models and brands. Likewise, the sampling rate differs considerably across phones with values ranging between 50Hz-200Hz. The MotionSense dataset \[61\] is recorded with the aim of inferring personal attributes, such as physical and demographics, in addition to the activities. The iPhone6s is placed in the users’ front pocket during the collection phase, while they performed 15 trials of 6 activities in the same experimental setting. In
total, 24 subjects of varying height, weight, age and gender performed the following 6 activities: walking, jogging, sitting, standing, downstairs and upstairs. We use this data only for the detection of activities without concerning with the identification of other attributes. UCI HAR \[18\] comprises data obtained from 30 subjects with waist-mounted Samsung Galaxy S2 devices sampling at 50Hz. Each participant completed 6 activities of daily living (i.e. standing, sitting, lying down, walking, downstairs and upstairs) during 2 trials with a 5 seconds resting condition in-between. The MobiAct \[104\] contains inertial sensors data collected from 66 participants with Samsung Galaxy S3 phones through more than 3200 trails. The subjects freely placed the device in their trouser's pocket to mimic real-life phone usage and placement. We utilize the data of 61 subjects for whom data of any of the following 11 activity classes is available: walking, jogging, jumping, upstairs, downstairs, sitting, stand to sit, sit to stand, sitting on a chair, car step-in and car step-out. The Human Activities and Postural Transitions (HAPT) dataset \[105\] is collected from a group of 30 volunteers with Samsung Galaxy S2 devices sampling at 50Hz. The phone was mounted on the waist of each subject who completed 3 dynamic activities (walking, upstairs, downstairs), 3 static posture activities (lying, sitting, standing), and 6 postural transitions (sit-to-lying, lie-to-sitting, stand-to-sitting, sit-to-standing, stand-to-lying, and lie-to-standing); resulting in 12 classes.

Sleep Stage Scoring

We use the PhysioNet Sleep-EDF\[^4\] dataset \[106, 107\] consisting of 61 polysomnograms (PSGs) from 20 subjects. It is comprised of participants from 2 different studies: a) effect of age on sleep and b) Temazepam effect on sleep. We use the 2 whole-night PSGs sleep recording sampled at 100Hz from the former study. Each record contains 2 electroencephalogram (EEG) signals from Fpz-Cz and Pz-Oz electrode locations, electrooculography (EOG), electromyography (EMG) and event markers. Some instances also have oro-nasal respiration and body temperature. The hypnograms (30-seconds1 epochs) were manually annotated by sleep expert with one of the 8 sleep classes (Wake, N1, N2, N3, N4, Rapid Eye Movement, Movement, Unknown), based on the R&K standard. We utilize EEG (Fpz-Cz) and EOG signals in our evaluation. Following previous work \[108\], we merged N3 and N4 into a single class N3 and discarded Movement and unscored samples, to have 5 sleep stages.

Stress Detection

For physiological stress recognition, we utilize the MIT DriverDb dataset \[107, 109\], which is collected during a real-world driving experiment in a city, on a highway and in a resting condition. The publicly-available version on PhysioNet consists of 17 drives out of 24, each lasted between 1-1.5 hours. The following physiological signals are recorded: EMG, electrocardiography (ECG), galvanic skin response (GSR) from hand and foot, heart rate (HR; derived from ECG), and breathing rate. The signals were originally sampled at different rates but downsampled to 15.5Hz. The 'marker' signal provided in the dataset is used to derive the binary ground truth, indicating a change-of-drive (i.e. resting, city or highway driving), which is found to be correlated with distress level through post-driving video analysis by ex-

\[^4\]version 1
perts. We use the following 10 drives 04, 05, 06, 07, 08, 09, 10, 11, 12 and 16 in our experiments, which have HR and GSR (from hand), given collection of other signals in real-life is quite problematic.

**WiFi Sensing**

Device-free context recognition with WiFi is an emerging area of research. To show the robustness of our self-supervised methods on this task, particularly on a unimodal signal, we utilize the WiFi channel state information (CSI) dataset for activity recognition. This dataset is collected in a controlled office environment, where the transmitting (router) and receiving (Intel 5300 NIC) devices were 3m apart, and the channel state information (CSI) was recorded at 1kHz. The 6 subjects performed 20 trials for each of the following 7 activities: lying down, falling, walking, running, sitting down, standing up and picking something up. The ground truth was obtained from videos recorded during the data collection process, and CSI amplitude is used for learning a model.

### 4.3.2 Pre-processing and Evaluation

To prepare the data for sequence modeling with a temporal convolutional network, we utilize a sliding window approach to segment the signals into fixed-sized inputs. In the case of the activity recognition task, we choose a window size of 400 samples with a 50% overlap, except for the HAPT dataset where a segment size of 200 samples is used, due to the short duration of posture-transition activities. We found these windows sizes to be optimal based on earlier experiments, as each activity dataset has a different sampling rate. We did not perform resampling as the sampling rate differences among phones does not vary significantly and 1D convolutional layers with wide kernel sizes learn to adapt to the specific characteristics of the input signal. However, if the sampling rate varies considerably it might be essential to do resampling. For Sleep-EDF, we applied minimal pre-processing based on existing work to formulate the problem as a 5-stage sleep classification and used the 30 seconds epochs as model input.

In the WiFi sensing task, we process the input in the same way as the original work that open-sourced the data and utilize a downsampled CSI signal of 500Hz as, which corresponds to an input window of 1 second. The heart rate and skin conductance signals from MIT DriverDb are processed to remove artifacts and these signals are mean normalized using the ‘mean’ and ‘standard deviation’ calculated from the baseline (or resting phase) of the data collection following for each subject. We use a window size of 30 seconds with 50% overlap to generate input segments for the model. We randomly split the datasets based on subjects into train and test sets withholding 70% users for training and the rest 30% for testing. We further divide the training set to obtain a validation set of size 20%, which is used for hyper-parameter tuning and early stopping. Most importantly, we also perform 5-fold cross-validation for thorough performance analysis whenever it is applicable. Furthermore, we z-normalize the samples with mean and standard deviation calculated from the training set. For self-supervision, we pretrain the models using only the training set, including for the

56
Table 4.2: Performance evaluation (weighted F-score) of self-supervised representations with a linear classifier. The unsupervised pretrained networks achieve competitive performance with the fully-supervised networks. In WiFi-CSI sub-table, the entries with hyphen indicate auxiliary tasks that cannot be applied to unimodal signals. See Table 4.6 in the appendix of this chapter for kappa scores.

<table>
<thead>
<tr>
<th>Method</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>MotionSense</th>
<th>UCI HAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.794±0.014</td>
<td>0.934±0.005</td>
<td>0.952±0.007</td>
<td>0.962±0.006</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.218±0.062</td>
<td>0.383±0.109</td>
<td>0.246±0.090</td>
<td>0.221±0.079</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.777±0.003</td>
<td>0.726±0.001</td>
<td>0.675±0.019</td>
<td>0.782±0.042</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.823±0.006</td>
<td>0.912±0.001</td>
<td>0.911±0.009</td>
<td>0.902±0.010</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.848±0.005</td>
<td>0.905±0.001</td>
<td>0.925±0.011</td>
<td>0.895±0.010</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.817±0.005</td>
<td>0.902±0.001</td>
<td>0.849±0.010</td>
<td>0.899±0.010</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.854±0.005</td>
<td>0.911±0.002</td>
<td>0.869±0.013</td>
<td>0.906±0.011</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.834±0.008</td>
<td>0.909±0.003</td>
<td>0.851±0.016</td>
<td>0.747±0.027</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.807±0.006</td>
<td>0.817±0.004</td>
<td>0.675±0.019</td>
<td>0.798±0.035</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.835±0.006</td>
<td>0.901±0.001</td>
<td>0.869±0.012</td>
<td>0.888±0.010</td>
</tr>
<tr>
<td>Tripet Loss</td>
<td>0.773±0.005</td>
<td>0.841±0.002</td>
<td>0.910±0.008</td>
<td>0.905±0.011</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>HAPT</th>
<th>Sleep-EDF</th>
<th>MIT DriverDb</th>
<th>WiFi CSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.899±0.009</td>
<td>0.825±0.005</td>
<td>0.824±0.029</td>
<td>0.964±0.007</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.119±0.041</td>
<td>0.149±0.127</td>
<td>0.321±0.198</td>
<td>0.153±0.04</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.669±0.003</td>
<td>0.679±0.012</td>
<td>0.876±0.002</td>
<td>0.767±0.005</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.818±0.006</td>
<td>0.779±0.004</td>
<td>0.890±0.002</td>
<td>-</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.815±0.004</td>
<td>0.782±0.006</td>
<td>0.892±0.004</td>
<td>-</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.822±0.002</td>
<td>0.671±0.022</td>
<td>0.866±0.000</td>
<td>0.837±0.005</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.841±0.003</td>
<td>0.778±0.006</td>
<td>0.908±0.001</td>
<td>0.768±0.007</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.782±0.004</td>
<td>0.707±0.012</td>
<td>0.883±0.005</td>
<td>0.731±0.011</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.738±0.002</td>
<td>0.784±0.002</td>
<td>0.902±0.001</td>
<td>-</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.790±0.003</td>
<td>0.772±0.003</td>
<td>0.885±0.002</td>
<td>0.774±0.008</td>
</tr>
<tr>
<td>Tripet Loss</td>
<td>0.815±0.002</td>
<td>0.775±0.003</td>
<td>0.891±0.001</td>
<td>0.749±0.009</td>
</tr>
</tbody>
</table>

Transfer learning experiments. The self-labeled examples are generated for each task on-the-fly during the learning phase, as defined earlier in Section 4.2.2.

For each recognition problem, we treat a fully-supervised model directly trained (in an end-to-end manner) with the annotated data of an end-task as a ‘baseline.’ Likewise, we also compare self-supervised tasks against pretraining with a autoencoder as a baseline. As explained earlier, we assess the quality of the self-supervised representation (including in the transfer-learning setting) through training a linear classifier or fine-tuning the last convolutional layer of the encoder on the downstream tasks. For learning in the low-data regime, we use a self-supervised network as initialization to quickly learn a model with few labeled examples. In all the cases, we assess the network performance with a weighted version of F-score and Cohen’s kappa (see appendix of [15]); as these metrics are robust to unbalanced class distributions while being sensitive to misclassifications.
4.3.3 Results and Discussion

**Linear separability and effects of fine-tuning the shared encoder**

For assessing the quality of the self-supervised embeddings, we conduct experiments with a linear classifier on the end-tasks. Linear separability is a standard way of measuring the power of self-supervised-learned features in the literature, i.e., if the representations disentangle factors of variations in the input, then it becomes easier to solve subsequent tasks. Here, we train a linear classifier (i.e. logistic regression) 10-times on top of a frozen network (pretrained with self-supervision) using annotated data of the downstream task. Table 4.2 summarizes the results on eight benchmark datasets from four application domains. We compare the performance against a fully-supervised network that is trained in an end-to-end manner (directly with annotated data). We also consider unsupervised pretraining with a standard autoencoder to analyze the improvements of self-supervision. Likewise, a linear model is also trained with random features (i.e. from a randomly initialized frozen network) to estimate its learning capacity.

On the activity recognition problem, the self-supervised features achieve very close results on multiple benchmarks to training an entire network with annotated instances. On the HHAR dataset, the transformation and fusion magnitude prediction tasks improve the F-score by 7 points. On other datasets with a large number of classes, such as HAPT and MobiAct, our simple proxy tasks learn features that are generalizable to end-tasks. In the case of sleep stage scoring, linear layers trained with features from the modality denoising and the fusion magnitude tasks achieve a kappa of 0.70, which is impressive given that the representations are learned from completely unlabeled data. Similarly, in a stress classification problem, the self-supervised networks outperform a fully-supervised model with a large margin. The transformations and modality denoising tasks achieve kappa scores of 0.80 and 0.79, respectively. We believe it is because pretraining results in generic features, whereas a model trained directly on the end-task suffers from overfitting. Lastly, we evaluate on the device-free sensing problem using the amplitude of WiFi CSI. Although we designed the auxiliary tasks for multisensor input, we find a subset of these to be applicable for self-supervision with a unimodal input. We achieve good results with self-supervised features even though the dataset size is relatively small, and input is noisy, complex and high-dimensional. The linear layer trained on top of the feature-prediction task representations achieves an F-score of 83% compared to the end-to-end training F-score of 96%.

In Table 4.3, we notice a substantial improvement on the downstream tasks if the last convolutional layer of the encoder (see Figure 3.2) is fine-tuned while training the linear classifier. Comparing with the results given in Table 4.2, it can be seen that the recognition rate of the models improved significantly, achieving similar results as the fully-supervised baselines; while features learned by input reconstruction with an autoencoder scored low compared to our proposed surrogate tasks even after fine-tuning, except for the WiFi sensing task. On the MobiAct dataset, transformations and sensor blend tasks gain 2 points improvement in kappa. Likewise, for MotionSense, HAPT and UCI HAR, we bridge the gap between fully-supervised and self-supervised models. Interestingly, fine-tuning did not help much with MIT DriverDb compared to training a linear classifier. These results agree with our intuition.
Table 4.3: Improvement in recognition rate (weighted F-score) by fine-tuning the shared layers of the encoder while training on the end-task. We observe a significant increase in performance across datasets with self-supervised networks, either surpassing or achieving results on-par with the baseline. See Table 4.7 in the appendix of this chapter for kappa scores.

<table>
<thead>
<tr>
<th>Method</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>MotionSense</th>
<th>UCI HAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.794±0.014</td>
<td>0.934±0.005</td>
<td>0.952±0.007</td>
<td>0.961±0.008</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.218±0.062</td>
<td>0.383±0.109</td>
<td>0.246±0.090</td>
<td>0.221±0.079</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.835±0.003</td>
<td>0.927±0.003</td>
<td>0.938±0.002</td>
<td>0.943±0.004</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.841±0.009</td>
<td>0.943±0.004</td>
<td>0.937±0.004</td>
<td>0.956±0.003</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.831±0.006</td>
<td>0.938±0.005</td>
<td>0.945±0.002</td>
<td>0.946±0.002</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.840±0.007</td>
<td>0.937±0.002</td>
<td>0.951±0.003</td>
<td>0.943±0.003</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.828±0.006</td>
<td>0.946±0.004</td>
<td>0.951±0.005</td>
<td>0.954±0.006</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.831±0.008</td>
<td>0.939±0.002</td>
<td>0.934±0.006</td>
<td>0.909±0.008</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.840±0.003</td>
<td>0.938±0.002</td>
<td>0.928±0.006</td>
<td>0.941±0.001</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.826±0.003</td>
<td>0.938±0.005</td>
<td>0.935±0.006</td>
<td>0.953±0.003</td>
</tr>
<tr>
<td>Triplet Loss</td>
<td>0.835±0.013</td>
<td>0.912±0.006</td>
<td>0.955±0.003</td>
<td>0.950±0.002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>HAPT</th>
<th>Sleep-EDF</th>
<th>MIT DriverDb</th>
<th>WiFi CSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.899±0.009</td>
<td>0.825±0.005</td>
<td>0.824±0.029</td>
<td>0.964±0.007</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.119±0.041</td>
<td>0.149±0.127</td>
<td>0.321±0.198</td>
<td>0.153±0.048</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.881±0.002</td>
<td>0.805±0.008</td>
<td>0.877±0.002</td>
<td>0.898±0.025</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.895±0.003</td>
<td>0.809±0.003</td>
<td>0.881±0.014</td>
<td>-</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.898±0.002</td>
<td>0.813±0.003</td>
<td>0.882±0.011</td>
<td>-</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.893±0.003</td>
<td>0.748±0.006</td>
<td>0.859±0.003</td>
<td>0.812±0.037</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.808±0.002</td>
<td>0.822±0.005</td>
<td>0.890±0.005</td>
<td>0.823±0.028</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.876±0.007</td>
<td>0.779±0.005</td>
<td>0.883±0.005</td>
<td>0.731±0.063</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.885±0.003</td>
<td>0.819±0.002</td>
<td>0.889±0.001</td>
<td>-</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.899±0.003</td>
<td>0.804±0.003</td>
<td>0.853±0.023</td>
<td>0.860±0.030</td>
</tr>
<tr>
<td>Triplet Loss</td>
<td>0.887±0.005</td>
<td>0.805±0.003</td>
<td>0.884±0.002</td>
<td>0.755±0.022</td>
</tr>
</tbody>
</table>

that training on an end-task directly in this case results in overfitting.

In summary, the evaluation with a linear classifier trained on top of a pretrained (self-supervised) feature extractor highlights that the representations learned with auxiliary tasks are broadly useful and better than autoencoding-based approaches. It also confirms our hypothesis that general-purpose representations can be learned directly from raw input without any strongly (task-specific) labeled data. It is important to note we did not aim to surpass fully-supervised approaches in this setting. Supervised methods will be better because they have direct access to task-specific labels, while self-supervised objectives train a network without any foresight of the end-task. It can also be seen from the results of fine-tuning the encoder, as presented in Table 4.3, that the network performance matches the supervised methods or improves upon, when shared layers are further trained on the downstream tasks. Likewise, it might be possible to improve generalization of self-supervised models through pretraining on larger unlabeled datasets in a real-world setting.
Impact on learning in low-data regime

We next investigate the performance of our approach in a semi-supervised (or low-data) setting. For this purpose, we pretrain an encoder using unlabeled instances for each self-supervised task and utilize it as initialization for efficiently learning with few labeled instances on the end-task; for the end-task, we add a randomly-initialized dense layer with 1024 hidden units before a linear output layer. The non-linear classifier is then learned and the encoder is fine-tuned with the specified number of instances per class. Specifically, for the defined auxiliary tasks and datasets, we use 5 and 10 examples for each category. We want to highlight that in real-life setting, a few labeled instances can be pooled from multiple users quite easily (e.g. 2-3 examples per user) as compared to accumulating several hundred for learning fully-supervised models. Likewise, personalization can also be achieved through precisely asking for a few labels for targeted classes. In Figure 4.3, we provide an average weighted F-score of 10 independent experiment runs, comparing training from scratch (FS) with the pretraining as an effective initialization for learning a robust classifier. We show that in contrast to the purely supervised approach, leveraging unlabeled data for learning network parameters improves the performance on the end-task. Specifically, our self-supervised models greatly improve the F-score in the low-data setting, in some cases achieving F-scores nearly as good as networks trained with the entire labeled data. Similarly, the self-supervised trained models perform better than the autoencoder, which shows that, despite the simplicity, our proposed auxiliary tasks force the network to learn highly-generalizable features. For each experiment run, we randomly sample the stated number of annotated instances and use these to train all the networks, including fully-supervised baselines.

On activity recognition, our methodology significantly improves the performance in low-data; for example, on the HHAR dataset with 5 and 10 instances, temporal shift and transformations tasks gain 4 and 7 points over the fully-supervised models’ F-score of 0.60 and 0.68, respectively. Similarly, for MobiAct, pretraining with the temporal shift task helps achieve an F-score of 0.75 (5 instances) and 0.82 (10 instances), compared to 0.61 and 0.73 respectively for networks learned from scratch. Furthermore, we achieve identical improvements on UCI HAR, HAPT, and MotionSense with 5 instances per class. The attained F-scores are 0.91, 0.77 and 0.83 in contrast to 0.90, 0.59, and 0.77 of fully-supervised models, respectively. Our method represents a 26 points increase in F-score on the challenging problem of sleep stage scoring. Likewise, on physiological stress detection and device-free sensing problems, the benefit of pretraining with auxiliary tasks is further apparent, where the presented methods achieve 12 points improvement in F-score over the baseline. These results suggest that self-supervision can greatly help with learning general-purpose representations that work well in the low-data regime. We also want to highlight that although the selection of an equal number of instances results in a balanced training set, we use the full test sets (as in earlier experiments) for evaluation, which could be imbalanced. Importantly, utilizing even bigger unlabeled datasets and combining weak-supervision methods can boost the quality of the learned representations.

We emphasize that the broader objective of self-supervised methods is to learn high-level semantic features that can be used to solve an array of downstream tasks with minimal labeled data. The evaluation of our presented auxiliary tasks clearly highlights the benefit of pretraining the network with unlabeled data to achieve better generalization on the tasks of
interest, with very few labeled instances. To the best of our knowledge, we, for the first time, evaluate self-supervised methods in a semi-supervised setting for problems involving multisenor data as earlier work developed fully-supervised network architectures or used classical autoencoding-based approaches for pretraining, followed by network fine-tuning with the entire labeled data. Overall, our approach provides a base for future work in developing sensing techniques that can achieve on-device personalization and perform continual, and few-shot learning, as the presented framework considerably reduces the requirement of labeled data from human annotators to learn the end-task models.

Effectiveness in a transfer learning setting

In a real-world learning setup, there is a high chance that we are interested in a different dataset and downstream task than the one originating from the unlabeled data accessible for pretraining. A broadly useful auxiliary task is thus one that produces generalizable representations that transfer well to other related end tasks. To examine the transferability property of the features learned with our proxy tasks, we evaluate their performance on the activity recognition datasets. To this end, we pretrain the feature extractor with each self-supervised objective (i.e. by discarding the semantic class labels) for all the five datasets (see Section 4.3.1 and investigate their performance through a) training a linear classifier with the entire target annotated data and b) fine-tuning it end-to-end with few labeled data (i.e. learning an activity classifier with 5 and 10 instances of each class from target dataset). Figure 4.4 provides the results of the source-to-target transfer of self-supervised models trained with nine different auxiliary losses. The diagonal entries of each subplot represent the F-scores when the source and target datasets are the same. In comparison with autoencoder pretraining, features learned with our tasks transfer well between datasets. We observe that even leveraging smaller unlabeled datasets produces useful features, as with sensor-blend-task-learned features on UCI HAR scored 0.91 F-score on the HHAR dataset. On the HAPT dataset of low input resolution (i.e. a segment size of 200 samples) and complex postural activities, transfer learning improves the performance with approximately 8 percentage points in F-score over pretraining on the same dataset. Importantly, our results are also competitive with the fully-supervised baselines on the respective datasets.

We further examine if the transferred self-supervised models are beneficial in learning from low-data; i.e. few labeled instances are available from the target data, but separate unannotated data is available for pretraining. We utilize the same network configuration as discussed earlier for low-data experiments and we fine-tune the model end-to-end. We randomly sample a specified number of instances and perform experiments 10 times while utilizing the same instances for both types of networks (i.e. pretrained and baseline) and report average F-score. In Figure 4.5, we present the results of optimal auxiliary tasks for each combination of the source to target transfer, where gray-colored bars show a fully-supervised baseline. Our experiments show that the features learned from different but related datasets do transfer well and improve the recognition rate even when as little as 5 examples per class are available. On the MobiAct dataset, our approach with HAPT as source data results in an F-score of 0.68 and 0.78 compared to the training from scratch F-score of 0.61 and 0.73, respectively. Similarly, with HAPT as a target, transferring from the UCI HAR using the sensor blend task, the F-score improved from 0.59 to 0.68 and 0.72 to 0.78. Interestingly, on UCI HAR and Mo-
Figure 4.3: Cont'd
Figure 4.3: Cont'd
tionSense, the performance attained with our approach is very close to the purely supervised models trained with entirely labeled data (see Table 4.2).

Learning generalizable representations that can be reused for solving related tasks is an important property to have in a learning system. Our investigation of transferring unsupervised pretrained models consistently highlights substantial performance improvements, indicating that the self-supervised features are broadly useful across different subjects, devices, environments and data collection protocols. In particular, the data efficiency enabled by our method in a low-data regime provides further evidence of semantic feature learning without merely
over-fitting on the source dataset. It is also important to note that compared to earlier work which focuses on supervised transfer or joint-training on source and target datasets, we provide evaluation of unsupervised transfer and its ability to boost performance even with few-labeled data. Likewise, self-supervised learning has other benefits as it has been shown to improve adversarial robustness and uncertainty of deep models as compared to purely supervised methods [111]. Although we did not study these aspects explicitly in this work, the results of transfer learning across domains hint that our auxiliary tasks also enhance the model’s robustness; we leave an in-depth study for future work.

Cross-validation to determine robustness against subject variations

To validate the stability of our methodology against variations in subjects’ data utilized for pre-training and downstream task evaluation, we perform 5-fold cross-validation based on user split (i.e. the train and test division (80 – 20) is based on users with no overlap among them;
train/test users are entirely independent); and we follow the same experimental setup as earlier. For each fold’s data and surrogate task, we pretrain the models and train a linear classifier on top of the frozen network. The fully-supervised baseline is trained in an end-to-end manner,

![Figure 4.5: Cont’d](image)
directly with the semantic labels. Table 4.4 summarizes the results averaged across 5 folds on eight considered datasets. We observe that the results achieved with self-supervision are consistent with earlier experiments. This highlights that our approach for sensory representation learning works well with different users’ data and it is robust to subjects’ differences. On the MobiAct dataset, the feature prediction and transformation recognition tasks achieve 0.90 F-score, which is very close to a fully-supervised model’s F-score of 0.91. Likewise, on MIT DriverDb, self-supervision provides an impressive improvement over training from scratch.

To summarize, these results suggest that the learned representations with unlabeled data learn useful features that can be used to a large extent for solving the end-task with a simple linear layer. Furthermore, we explore fine-tuning the last convolutional layer of the encoder while training a linear layer on downstream tasks. In Table 4.5, we show that fine-tuning a shared layer leads to a better performance than the fully-supervised model training from scratch on most of the datasets. The feature prediction task on the HHAR dataset achieved an F-score of 0.87, which is 5 points above the baseline. Likewise, on other datasets and tasks, our technique either bridges the gap or achieves broadly similar results as the supervised models. We think that careful fine-tuning of the architecture and related hyper-parameters could further improve the recognition rate of self-supervised networks. We note that a direct comparison of our approach with existing methods is not feasible as we learn representations from unlabeled data and evaluate through training a linear classifier, whereas, prior methods focus on fully-supervised learning with different architectures and evaluation strategies. However, to be comparative, we summarize related results here, which are only indicative. On MotionSense, our sensor blend task achieves an F-score of 0.92 compared to 0.95 and 0.86 accuracy for trial- and subject-wise evaluation in [61]. For SleepEDF, our fusion magnitude task scores a kappa of 0.72 compared to 0.76 of a sophisticated fully-supervised model [108]. Likewise, on WiFi sensing task, feature prediction proxy task results in an F-score of 0.85 compared to the 0.90 accuracy of an LSTM-based model [110] over six classes.

We wondered whether pretraining with our auxiliary tasks is invariant to utilized subjects’
data, as it is critical for learning in a real-world setting due to the non-curated nature of the data. We found that proxy tasks are highly stable and result in a similar performance as earlier, when a linear classifier is trained on top of self-supervised feature extractors. This analysis further shows that the self-supervised features are not necessarily subject-specific, but are general in nature. Moreover, our evaluation demonstrates there is a room for improvement through selecting problem- or task-specific network architectures and using larger unlabeled datasets for unsupervised learning. Specifically, it would be valuable to explore unifying supervised and self-supervised objectives in a multi-task setting to personalize or adapt sensing models directly on user devices.

Table 4.4: Comparison of self-supervised representation learning to fully-supervised approach with 5-fold cross-validation based on user-split. We pretrain the feature extractors for each fold's data and learn a linear classifier for the end-task as usual. We report weighted F-score averaged over the 5 folds, highlighting the robustness of our method to subject variations. See Table 4.4 in the appendix of this chapter for kappa scores.

<table>
<thead>
<tr>
<th>Method</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>MotionSense</th>
<th>UCI HAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.844±0.090</td>
<td>0.917±0.017</td>
<td>0.960±0.007</td>
<td>0.951±0.025</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.199±0.047</td>
<td>0.394±0.086</td>
<td>0.284±0.086</td>
<td>0.268±0.208</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.722±0.085</td>
<td>0.736±0.021</td>
<td>0.752±0.050</td>
<td>0.831±0.041</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.829±0.061</td>
<td>0.886±0.010</td>
<td>0.920±0.019</td>
<td>0.913±0.038</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.841±0.040</td>
<td>0.889±0.014</td>
<td>0.924±0.025</td>
<td>0.899±0.049</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.820±0.068</td>
<td>0.900±0.016</td>
<td>0.900±0.025</td>
<td>0.896±0.043</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.822±0.059</td>
<td>0.900±0.011</td>
<td>0.898±0.013</td>
<td>0.916±0.018</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.811±0.057</td>
<td>0.890±0.017</td>
<td>0.889±0.027</td>
<td>0.793±0.030</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.798±0.077</td>
<td>0.834±0.029</td>
<td>0.780±0.058</td>
<td>0.829±0.056</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.812±0.079</td>
<td>0.890±0.015</td>
<td>0.901±0.014</td>
<td>0.861±0.015</td>
</tr>
<tr>
<td>Tripet Loss</td>
<td>0.749±0.065</td>
<td>0.822±0.013</td>
<td>0.917±0.022</td>
<td>0.893±0.036</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>HAPT</th>
<th>Sleep-EDF</th>
<th>MIT DriverDb</th>
<th>WiFi CSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.897±0.053</td>
<td>0.822±0.025</td>
<td>0.789±0.122</td>
<td>0.959±0.005</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.155±0.061</td>
<td>0.072±0.021</td>
<td>0.206±0.015</td>
<td>0.214±0.044</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.818±0.064</td>
<td>0.701±0.026</td>
<td>0.850±0.054</td>
<td>0.793±0.014</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.855±0.044</td>
<td>0.788±0.014</td>
<td>0.824±0.106</td>
<td>-</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.840±0.040</td>
<td>0.795±0.025</td>
<td>0.859±0.061</td>
<td>-</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.859±0.040</td>
<td>0.777±0.033</td>
<td>0.843±0.045</td>
<td>0.855±0.024</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.863±0.045</td>
<td>0.788±0.028</td>
<td>0.860±0.060</td>
<td>0.770±0.032</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.837±0.042</td>
<td>0.753±0.027</td>
<td>0.844±0.082</td>
<td>0.729±0.015</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.835±0.050</td>
<td>0.797±0.029</td>
<td>0.864±0.061</td>
<td>-</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.821±0.043</td>
<td>0.767±0.037</td>
<td>0.839±0.071</td>
<td>0.793±0.018</td>
</tr>
<tr>
<td>Tripet Loss</td>
<td>0.845±0.044</td>
<td>0.789±0.027</td>
<td>0.860±0.059</td>
<td>0.769±0.022</td>
</tr>
</tbody>
</table>

4.3.4 Impact and Limitations

Our Sense and Learn framework shows that it is possible to use unlabeled data, in addition to smaller amounts of labeled data, when learning features for varied classification problems. We believe our method is useful in practice, where obtaining labeled data is difficult and costly. Since the same approach, with a fixed neural network structure, provides gains for quite different application areas, ranging from activity recognition to sleep stage scoring, we also believe
Table 4.5: The effect of fine-tuning modality-agnostic encoder while learning downstream task under 5-folds cross-validation as evaluated through weighted F-score. See Table 4.9 in the appendix of this chapter for kappa scores.

<table>
<thead>
<tr>
<th>Method</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>MotionSense</th>
<th>UCI HAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.844±0.090</td>
<td>0.917±0.017</td>
<td>0.960±0.007</td>
<td>0.951±0.025</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.199±0.047</td>
<td>0.394±0.086</td>
<td>0.284±0.086</td>
<td>0.268±0.208</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.891±0.049</td>
<td>0.914±0.019</td>
<td>0.961±0.010</td>
<td>0.936±0.051</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.893±0.062</td>
<td>0.919±0.011</td>
<td>0.964±0.011</td>
<td>0.949±0.036</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.885±0.054</td>
<td>0.918±0.011</td>
<td>0.961±0.013</td>
<td>0.942±0.039</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.894±0.050</td>
<td>0.930±0.014</td>
<td>0.962±0.003</td>
<td>0.943±0.047</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.893±0.052</td>
<td>0.933±0.0126</td>
<td>0.968±0.007</td>
<td>0.949±0.033</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.885±0.055</td>
<td>0.920±0.014</td>
<td>0.941±0.012</td>
<td>0.915±0.050</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.886±0.061</td>
<td>0.929±0.015</td>
<td>0.966±0.011</td>
<td>0.933±0.054</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.894±0.067</td>
<td>0.927±0.011</td>
<td>0.962±0.004</td>
<td>0.951±0.030</td>
</tr>
<tr>
<td>Tripet Loss</td>
<td>0.856±0.055</td>
<td>0.904±0.020</td>
<td>0.957±0.006</td>
<td>0.944±0.044</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>HAPT</th>
<th>Sleep-EDF</th>
<th>MIT DriverDb</th>
<th>WiFi CSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.897±0.053</td>
<td>0.822±0.025</td>
<td>0.789±0.122</td>
<td>0.959±0.005</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.155±0.061</td>
<td>0.072±0.021</td>
<td>0.206±0.015</td>
<td>0.214±0.044</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.883±0.059</td>
<td>0.764±0.028</td>
<td>0.804±0.132</td>
<td>0.911±0.032</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.892±0.052</td>
<td>0.801±0.020</td>
<td>0.793±0.149</td>
<td>-</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.884±0.051</td>
<td>0.808±0.023</td>
<td>0.788±0.148</td>
<td>-</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.893±0.055</td>
<td>0.794±0.031</td>
<td>0.795±0.143</td>
<td>0.857±0.040</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.896±0.051</td>
<td>0.801±0.029</td>
<td>0.806±0.127</td>
<td>0.805±0.051</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.890±0.052</td>
<td>0.781±0.027</td>
<td>0.805±0.133</td>
<td>0.758±0.048</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.882±0.051</td>
<td>0.796±0.028</td>
<td>0.858±0.051</td>
<td>-</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.888±0.048</td>
<td>0.778±0.035</td>
<td>0.849±0.050</td>
<td>0.854±0.032</td>
</tr>
<tr>
<td>Tripet Loss</td>
<td>0.888±0.056</td>
<td>0.792±0.031</td>
<td>0.806±0.128</td>
<td>0.765±0.022</td>
</tr>
</tbody>
</table>

the method is applicable in practice. While it is true that a practitioner cannot be certain which self-supervised task will work best for a new application, the range of experiments we present should provide a valuable starting point as to which tasks are most promising. Moreover, our fine-tuning experiments (Table 4.3) show that e.g. the Transformations task provides significant gains across all datasets even when using all available supervised data. Finally, self-supervised tasks do not need any labels while learning the representations, which opens up the possibility of using our framework for on-device Federated Learning [112], where the sensor data never leaves the users’ device (e.g., smartphone).

Self-supervised learning provides a scalable, inexpensive, and data efficient way to learn high-level features with deep neural networks without requiring strong labels, which could be unclear, noisy or limited for many real-world problems. However, there are limitations of these approaches which are also applicable to our methodology. First, deep neural networks are prone to learning via shortcuts through exploiting low-level cues in the input e.g. object textures and other local artifacts in image classification [97]. The unintended cue learning is not limited to supervised methods, but is a problem for self-supervised methods too, as networks can use shortcuts to solve proxy task without learning anything useful (e.g. chromatic aberration in vision models [113]). For time-series or multisensor inputs discovering, a model relying on shortcuts is an unsolved problem and could be challenging to detect. Second, as
getting access to large unlabeled and labeled sensory datasets is difficult, evaluating how auxiliary tasks will perform on non-curated data or learning in an open-world environment needs further exploration. Third and last, interpretability and understanding the decision mechanism of deep models is another open area of research to address issues of model uncertainty, bias and fairness. The features learned with deep network could be non-interpretable, but we think that unifying shallow models using hand-crafted features with deep networks consuming raw input through knowledge distillation [114] might shed light on the importance of certain features.

### 4.4 Related Work

#### Unsupervised and Self-Supervised Learning

Deep learning has revolutionized several areas of research with an intuitive property of learning discriminative features directly from the raw data and eliminating the need of manual feature extraction [8, 74, 113, 116]. The success of deep learning is largely attributed to the massive labeled datasets apart from other factors, such as availability of computational power and better neural architectures. Obtaining semantically labeled data required for training supervised models is an expensive and time-consuming process. Therefore, unsupervised learning has seen growing interest in the last couple of years as unlabeled data is available in huge quantities, especially on decentralized edge devices. A classical illustration of unsupervised feature learning is the autoencoder, which learns to map an input onto a lower-dimensional embedding so that reconstructing the original input from such a space incurs a lower error. However, the decoding-based strategies deplete the network capacity through attending to low-level details instead of capturing semantically meaningful features. Therefore, the focus of recent studies is on providing an alternative form of supervision, where annotations can be intrinsically extracted from the data itself.

The field of self-supervised learning exploits the natural supervision available within the input signal to define a surrogate task that can force the network to learn broadly-useable representations. To that end, numerous pretext tasks are proposed in different domains. [57] established the task of predicting the relative position of randomly cropped image patches. [117, 118] inferred color values for grayscale pictures. [119] utilize time-contrastive loss as a way to minimize the embedding distances of the same scene recorded from multiple viewpoints, while maximizing the distances for those captured at different timesteps. A similar technique is proposed in [120] to learn from multiple views of the data. [58] defined self-supervised tasks for audio, inspired by word2vec [121]. [58] showed that video representations could be learned by exploiting audio-visual temporal synchronization. Time-contrastive learning is suggested in [122] for extracting features from time-series, in an unsupervised manner, through predicting segment IDs. Likewise, autoregressive modeling has been combined with predictive coding to learn compact latent embeddings for various domains [42].

For natural language modeling, self-supervised objectives, such as predicting masked tokens from surrounding ones and predicting the next sentence, turn out to be powerful methods for learning generic representations of text [58]. Similarly, for learning inertial sensory fea-
tures, [14, 123, 124] presented a signal transformation recognition task. Lately, self-supervised learning has been shown to be beneficial for semi-supervised learning, through jointly optimizing supervised and self-supervised losses [125]. In this work, we develop several self-supervised tasks for learning representations from a wide range of sensory data such as electroencephalography, electrodermal activity and inertial signals. We show that pretraining with self-supervision using unlabeled data helps in learning highly generalizable features that improve data efficiency and transfer well to a related set of tasks.

Learning Sensing Models with Machine Learning

An understanding of human contexts, activities and states is an important area of research in ambient computing and pervasive sensing due to the fact that it can play a central role in several application domains including: health, wellness, assistance, monitoring, and human computer interaction. To achieve the earlier described objective, the data is collected from users through wearables or other sensors, under varied environments, for learning a task-specific model. For instance, prior work on activity recognition explored various methodologies with inertial sensors embedded in smartphones or smartwatches [56, 74, 126]. Emotional state recognition is widely achieved with physiological signals, such as skin conductance and heart rate variability [24, 115, 127]. Similarly in sleep analysis, the electrical brain activity is captured with an electroencephalogram to classify sleep into different stages [108, 128, 129]. Importantly, for device-free sensing systems, channel state information from WiFi is utilized to infer participants’ activities in a non-intrusive manner [110]. Earlier developed methods for these problems heavily relied on manual feature extraction from sensory data to infer a user's activity, emotional state or sleep score and these methods were limited depending on the domain knowledge available to extract discriminative features. With the tremendous progress in end-to-end supervised learning via deep networks, it has been shown that the features can be learned directly from data instead of hand-crafting them based on domain knowledge [3, 74, 115, 116].

Consequently, 1D convolutional and recurrent neural networks have become standard techniques for achieving state-of-the-art performance on problems involving temporal data [24, 74, 108, 116]. Nevertheless, these approaches have heavily relied on the availability of large-annotated datasets, which are notoriously difficult to acquire in the real-world. Due to this, in recent years, few work explored unsupervised feature learning to exploit the availability of vast amounts of unlabeled data, while mainly focusing on input reconstruction via autoencoders and related variants, such as restricted Boltzmann machines and sparse coding [124, 125, 126, 119]. There has also been work on utilizing generative adversarial networks for modeling data distributions without supervision [130, 131] and in semi-supervised learning for sensing models [72]. Furthermore, transfer learning has also been leveraged to improve neural network generalization in domains where large labeled data is difficult to obtain, but focused on transfer from supervised models [132, 133].

More recently, [14] proposed a self-supervised task of signal transformation recognition for feature learning that achieved significant improvement in activity recognition over autoencoding, though focusing only on unimodal input and the activity recognition problem. As opposed to earlier works, we present a general framework for learning multimodal represen-
tations from a diverse set of sensors in a self-supervised way and compared to [14] we simplify the problem formulation of transformation recognition (see Section 4.2.2); our novel proxy tasks work on-par and can be used when transforming the input is not desirable or when it may lead to unintended outcomes (e.g. ECG signals). Furthermore, pretraining models with our auxiliary tasks significantly lower the amount of labeled data required to achieve better generalization and opens up the possibility of on-device learning from decentralized unlabeled data.

4.5 Conclusion

We proposed a self-supervised framework for multisensor representation learning from unlabeled data, produced by the omnipresent sensors. To realize the vision of unsupervised learning for sensing systems and IoT in general, we developed eight novel auxiliary tasks that acquire their supervision signal directly from the raw input, without any human involvement. The defined proxy objectives are utilized to learn general and effective deep models for a wide variety of problems. Through extensive evaluation on eight publicly available datasets from four application domains, we demonstrate that the self-supervised networks learn useful semantic representations that are competitive with fully-supervised models (i.e. trained end-to-end with labeled data). In summary, we demonstrated that the straight-forward and computationally-inexpensive surrogate tasks perform well on downstream tasks of interest by learning a linear classifier on top of frozen feature extractors. We further showed that fine-tuning a pretrained modality-agnostic encoder further improved the detection rate of a network. As the key objective of leveraging unannotated data is to reduce the labeled data required for the end-tasks, we have also shown that our approach significantly improves the performance in the low-data regime. In particular, with as few as 5 to 10 labeled examples per class, the self-supervised initialized networks achieve an F-score between 0.70-0.80. Furthermore, we examined the effectiveness of learned representations in an unsupervised transfer setting with linear separability analysis and semi-supervised learning, achieving much better results than training from scratch.

Various icons used in the figure are created by Sriramteja SRT, Berkah Icon, Ben Davis, Eucalyp, ibrandify, Clockwise, Aenne Brielmann, Anuar Zhumaev, and Tim Madle from the Noun Project.
### Table 4.6: Performance evaluation of self-supervised representations with a linear classifier. See Section 4.3.3 for more details.

<table>
<thead>
<tr>
<th>Method</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>MotionSense</th>
<th>UCI HAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.758 ± 0.019</td>
<td>0.915 ± 0.007</td>
<td>0.941 ± 0.010</td>
<td>0.955 ± 0.007</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.115 ± 0.069</td>
<td>0.254 ± 0.122</td>
<td>0.153 ± 0.086</td>
<td>0.157 ± 0.104</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.732 ± 0.004</td>
<td>0.696 ± 0.002</td>
<td>0.654 ± 0.011</td>
<td>0.749 ± 0.041</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.785 ± 0.007</td>
<td>0.890 ± 0.001</td>
<td>0.890 ± 0.011</td>
<td>0.881 ± 0.013</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.815 ± 0.006</td>
<td>0.880 ± 0.002</td>
<td>0.907 ± 0.014</td>
<td>0.874 ± 0.013</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.780 ± 0.007</td>
<td>0.878 ± 0.002</td>
<td>0.824 ± 0.012</td>
<td>0.878 ± 0.012</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.826 ± 0.006</td>
<td>0.890 ± 0.002</td>
<td>0.838 ± 0.016</td>
<td>0.888 ± 0.013</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.801 ± 0.010</td>
<td>0.884 ± 0.004</td>
<td>0.818 ± 0.019</td>
<td>0.708 ± 0.027</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.771 ± 0.007</td>
<td>0.789 ± 0.004</td>
<td>0.656 ± 0.017</td>
<td>0.758 ± 0.043</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.801 ± 0.008</td>
<td>0.877 ± 0.002</td>
<td>0.837 ± 0.015</td>
<td>0.871 ± 0.010</td>
</tr>
<tr>
<td>Triplet Loss</td>
<td>0.727 ± 0.006</td>
<td>0.802 ± 0.002</td>
<td>0.888 ± 0.011</td>
<td>0.888 ± 0.012</td>
</tr>
</tbody>
</table>

### Table 4.7: Improvement in recognition rate by fine-tuning the shared layers of the encoder while training on the end-task. See Section 4.3.3 for more details.

<table>
<thead>
<tr>
<th>Method</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>MotionSense</th>
<th>UCI HAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.883 ± 0.011</td>
<td>0.760 ± 0.007</td>
<td>0.637 ± 0.054</td>
<td>0.955 ± 0.009</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.041 ± 0.039</td>
<td>0.026 ± 0.068</td>
<td>0.077 ± 0.206</td>
<td>0.011 ± 0.042</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.646 ± 0.004</td>
<td>0.166 ± 0.014</td>
<td>0.736 ± 0.005</td>
<td>0.713 ± 0.005</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.792 ± 0.007</td>
<td>0.691 ± 0.005</td>
<td>0.766 ± 0.004</td>
<td>-</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.789 ± 0.007</td>
<td>0.700 ± 0.008</td>
<td>0.771 ± 0.010</td>
<td>-</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.800 ± 0.002</td>
<td>0.418 ± 0.011</td>
<td>0.715 ± 0.001</td>
<td>0.798 ± 0.006</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.820 ± 0.003</td>
<td>0.696 ± 0.008</td>
<td>0.804 ± 0.003</td>
<td>0.715 ± 0.009</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.753 ± 0.004</td>
<td>0.599 ± 0.014</td>
<td>0.751 ± 0.011</td>
<td>0.670 ± 0.013</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.717 ± 0.003</td>
<td>0.702 ± 0.002</td>
<td>0.792 ± 0.003</td>
<td>-</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.758 ± 0.004</td>
<td>0.689 ± 0.004</td>
<td>0.758 ± 0.004</td>
<td>0.722 ± 0.009</td>
</tr>
<tr>
<td>Triplet Loss</td>
<td>0.789 ± 0.003</td>
<td>0.690 ± 0.005</td>
<td>0.769 ± 0.003</td>
<td>0.690 ± 0.012</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>HAPT</th>
<th>Sleep-EDF</th>
<th>MIT DriverDb</th>
<th>WiFi CSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.758 ± 0.019</td>
<td>0.915 ± 0.007</td>
<td>0.941 ± 0.010</td>
<td>0.955 ± 0.007</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.115 ± 0.069</td>
<td>0.254 ± 0.122</td>
<td>0.153 ± 0.086</td>
<td>0.157 ± 0.104</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.732 ± 0.004</td>
<td>0.696 ± 0.002</td>
<td>0.654 ± 0.011</td>
<td>0.749 ± 0.041</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.785 ± 0.007</td>
<td>0.890 ± 0.001</td>
<td>0.890 ± 0.011</td>
<td>0.881 ± 0.013</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.815 ± 0.006</td>
<td>0.880 ± 0.002</td>
<td>0.907 ± 0.014</td>
<td>0.874 ± 0.013</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.780 ± 0.007</td>
<td>0.878 ± 0.002</td>
<td>0.824 ± 0.012</td>
<td>0.878 ± 0.012</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.826 ± 0.006</td>
<td>0.890 ± 0.002</td>
<td>0.838 ± 0.016</td>
<td>0.888 ± 0.013</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.801 ± 0.010</td>
<td>0.884 ± 0.004</td>
<td>0.818 ± 0.019</td>
<td>0.708 ± 0.027</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.771 ± 0.007</td>
<td>0.789 ± 0.004</td>
<td>0.656 ± 0.017</td>
<td>0.758 ± 0.043</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.801 ± 0.008</td>
<td>0.877 ± 0.002</td>
<td>0.837 ± 0.015</td>
<td>0.871 ± 0.010</td>
</tr>
<tr>
<td>Triplet Loss</td>
<td>0.727 ± 0.006</td>
<td>0.802 ± 0.002</td>
<td>0.888 ± 0.011</td>
<td>0.888 ± 0.012</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>HAPT</th>
<th>Sleep-EDF</th>
<th>MIT DriverDb</th>
<th>WiFi CSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.883 ± 0.011</td>
<td>0.760 ± 0.007</td>
<td>0.637 ± 0.054</td>
<td>0.955 ± 0.009</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.041 ± 0.039</td>
<td>0.026 ± 0.068</td>
<td>0.077 ± 0.206</td>
<td>0.011 ± 0.042</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.646 ± 0.004</td>
<td>0.166 ± 0.014</td>
<td>0.736 ± 0.005</td>
<td>0.713 ± 0.005</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.792 ± 0.007</td>
<td>0.691 ± 0.005</td>
<td>0.766 ± 0.004</td>
<td>-</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.789 ± 0.007</td>
<td>0.700 ± 0.008</td>
<td>0.771 ± 0.010</td>
<td>-</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.800 ± 0.002</td>
<td>0.418 ± 0.011</td>
<td>0.715 ± 0.001</td>
<td>0.798 ± 0.006</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.820 ± 0.003</td>
<td>0.696 ± 0.008</td>
<td>0.804 ± 0.003</td>
<td>0.715 ± 0.009</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.753 ± 0.004</td>
<td>0.599 ± 0.014</td>
<td>0.751 ± 0.011</td>
<td>0.670 ± 0.013</td>
</tr>
<tr>
<td>Modality Denoise.</td>
<td>0.717 ± 0.003</td>
<td>0.702 ± 0.002</td>
<td>0.792 ± 0.003</td>
<td>-</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.758 ± 0.004</td>
<td>0.689 ± 0.004</td>
<td>0.758 ± 0.004</td>
<td>0.722 ± 0.009</td>
</tr>
<tr>
<td>Triplet Loss</td>
<td>0.789 ± 0.003</td>
<td>0.690 ± 0.005</td>
<td>0.769 ± 0.003</td>
<td>0.690 ± 0.012</td>
</tr>
</tbody>
</table>
Table 4.8: Comparison of self-supervised representation learning to fully-supervised approach with 5-fold cross-validation based on user-split. See Section 4.3.5 for more details.

<table>
<thead>
<tr>
<th>Method</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>MotionSense</th>
<th>UCI HAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.820±0.098</td>
<td>0.891±0.024</td>
<td>0.950±0.008</td>
<td>0.941±0.030</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.107±0.072</td>
<td>0.272±0.084</td>
<td>0.202±0.082</td>
<td>0.190±0.223</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.672±0.104</td>
<td>0.703±0.029</td>
<td>0.719±0.038</td>
<td>0.805±0.041</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.796±0.074</td>
<td>0.853±0.014</td>
<td>0.902±0.024</td>
<td>0.898±0.048</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.809±0.047</td>
<td>0.819±0.018</td>
<td>0.906±0.030</td>
<td>0.877±0.063</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.787±0.083</td>
<td>0.876±0.020</td>
<td>0.878±0.010</td>
<td>0.875±0.051</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.789±0.071</td>
<td>0.876±0.015</td>
<td>0.873±0.017</td>
<td>0.900±0.022</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.776±0.069</td>
<td>0.859±0.022</td>
<td>0.863±0.032</td>
<td>0.756±0.038</td>
</tr>
<tr>
<td>Modality Denoise</td>
<td>0.762±0.092</td>
<td>0.802±0.036</td>
<td>0.750±0.065</td>
<td>0.799±0.059</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.777±0.090</td>
<td>0.863±0.019</td>
<td>0.877±0.017</td>
<td>0.843±0.012</td>
</tr>
<tr>
<td>Triplet Loss</td>
<td>0.707±0.077</td>
<td>0.777±0.018</td>
<td>0.897±0.027</td>
<td>0.873±0.043</td>
</tr>
</tbody>
</table>

Table 4.9: The effect of fine-tuning modality-agnostic encoder while learning downstream task under 5-folds cross-validation.

<table>
<thead>
<tr>
<th>Method</th>
<th>HAPT</th>
<th>Sleep-EDF</th>
<th>MIT DriverDb</th>
<th>WiFi CSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Supervised</td>
<td>0.820±0.063</td>
<td>0.760±0.037</td>
<td>0.577±0.219</td>
<td>0.949±0.006</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.075±0.046</td>
<td>0.004±0.006</td>
<td>0.02±0.00</td>
<td>0.042±0.049</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.879±0.077</td>
<td>0.601±0.021</td>
<td>0.677±0.118</td>
<td>0.743±0.019</td>
</tr>
<tr>
<td>Sensor Blend</td>
<td>0.832±0.053</td>
<td>0.715±0.026</td>
<td>0.636±0.203</td>
<td>-</td>
</tr>
<tr>
<td>Fusion Magnitude</td>
<td>0.816±0.049</td>
<td>0.724±0.033</td>
<td>0.696±0.132</td>
<td>-</td>
</tr>
<tr>
<td>Feature Prediction</td>
<td>0.837±0.048</td>
<td>0.701±0.014</td>
<td>0.661±0.100</td>
<td>0.821±0.029</td>
</tr>
<tr>
<td>Transformations</td>
<td>0.842±0.051</td>
<td>0.712±0.026</td>
<td>0.698±0.134</td>
<td>0.716±0.039</td>
</tr>
<tr>
<td>Temporal Shift</td>
<td>0.812±0.047</td>
<td>0.664±0.035</td>
<td>0.667±0.170</td>
<td>0.669±0.019</td>
</tr>
<tr>
<td>Modality Denoise</td>
<td>0.810±0.060</td>
<td>0.728±0.046</td>
<td>0.708±0.134</td>
<td>-</td>
</tr>
<tr>
<td>Odd Segment</td>
<td>0.793±0.053</td>
<td>0.690±0.049</td>
<td>0.655±0.151</td>
<td>0.745±0.023</td>
</tr>
<tr>
<td>Triplet Loss</td>
<td>0.820±0.052</td>
<td>0.715±0.037</td>
<td>0.698±0.128</td>
<td>0.713±0.027</td>
</tr>
</tbody>
</table>
Chapter 5

Federated Self-Supervised Learning of Multi-Sensor Representations

This chapter is based on our paper Federated Self-Supervised Learning of Multi-Sensor Representations for Embedded Intelligence published in IEEE Internet of Things Journal 2020 [17] and it was a joint work with Flora D. Salim.

5.1 Introduction

In Chapters 3 and 4, we have introduced self-supervised auxiliary tasks for learning models from wide variety of sensory data. Here, we introduce a general contrastive approach based on multi-view inputs for learning representations without any hand crafted pretext tasks. We use it especially for unifying self-supervision with federated learning in this chapter.

The wealth of sensory data from Internet of Things (IoT) devices are only recently being leveraged for tackling important problems in understanding context, user monitoring, health, and other predictive analytics tasks e.g., for emotional well-being [134, 135], sleep tracking [108], and physical activity detection [14]. The success is mainly attributed to the supervised methods that utilize labeled datasets for training models in a central environment, while learning models from unlabeled decentralized data still presents a major challenge. Obtaining large, well-curated sensory data from edge devices is especially difficult owing to issues like user privacy, the prohibitive cost of labeling, bandwidth limitations, network connectivity and the diversity of device types. These factors make it significantly challenging to harness abundant data on remote devices for learning semantic features with standard supervised approaches.

We hypothesize that the fusion of self-supervision with federated learning could result in an effective method for learning from unlabeled, private and diverse types of sensory data which is
crucial for several embedded (personalized) machine learning tasks. To achieve this objective, we develop a novel auxiliary task based on a wavelet transform, which we call *scalogram-signal correspondence learning* (SSCL). A deep temporal convolution network is trained to solve the specified task so as to learn representations from a variety of sensory inputs (e.g., electroencephalography, inertial measurement unit’s sensors (IMUs), and WiFi channel state information). We name it a *scalogram contrastive network* (SCN). Specifically, the self-supervised scheme is designed to contrast between a raw signal (time-series) and its complementary view, which in our case, is a scalogram, extracted with continuous wavelet transform $\textstyle \mathcal{W}_a$. However, we note that other views, such as spectrogram derived with fast Fourier transform can also be used in combination but in this work, we opt only for wavelet transformation due to its better capability at localizing time-frequency properties $\textstyle \mathcal{W}_a$ of the signal.

The core idea behind our pretext task is to determine if a given pair of scalogram-signal inputs are aligned or misaligned, i.e., whether a scalogram is the transformation of a given signal. The presented auxiliary task can formally be seen as a binary classification problem, and we employ a contrastive objective inspired by [43] for optimizing it (see Figure 5.1 for an overview) in both central and federated settings without involving a human in the data labeling process. Importantly, we would like to highlight that for the model to successfully solve the defined task, it should learn the core semantics in shared input views through possibly relating frequency, scale, and other information present in the signal. The network captures important latent relationships through correlating scalogram-signal inputs in the embedding space. Mainly, the representations that could emerge from the learning process are form of invariances (such as sensor noise, subject-specific variations), which are essential in several tasks involving sensory data, e.g., stress detection with physiological signals.

The key contributions of this work are three-fold: First, we propose a scalogram-signal correspondence learning framework for self-supervised learning from diverse sensory data. Second, to the best of our knowledge, we for the first time propose to unify federated learning with self-supervision to learn from unlabeled and private data on edge devices. Third, we extensively assess the proposed method on several publicly available datasets from different domains with linear classification protocol in central and federated contexts, low-data regime (i.e., semi-supervised setting), and transfer learning including cross-validation. The SCN achieves competitive performance in comparison with fully-supervised networks that are trained entirely on labeled data and works significantly better than other approaches. Particularly, SCN fine-tuning with few labeled data, e.g., five or ten instances per class, improves the F-score by as much as 5%-6% than training from scratch. Our approach also works better than transferring supervised features, learned from the source data, between the related tasks.

5.2 Background

We consider learning sensory features from raw unlabeled data with a deep neural network $\mathcal{F}_\theta$ (parameterized by $\theta$), which transforms an input from $\mathcal{X}$ into an output in $\mathcal{Z}$. Here, we refer to a vector obtained through applying a mapping function $\mathcal{F} : \mathcal{X} \mapsto \mathcal{Z}$ from an arbitrary intermediate or penultimate layer of the network as ‘representation’ or ‘feature.’ Our objective is to learn general-purpose representations that can make subsequent tasks of interests easier
to solve. To this end, numerous unsupervised methods are developed to leverage a large amount of unlabeled data for achieving better generalization. Moreover, the data required for model development could not only be unannotated but also distributed, without the option to accumulate it in a centralized repository due to privacy concerns and its ever-increasing size. To tackle the issue of learning models from decentralized user data, the field of federated learning \cite{137} is rapidly gaining momentum. Our work is intended to unify self-supervision with federated learning to realize the vision of on-device learning, with a focus on multi-sensor inputs. We provide an overview of federated learning and wavelet transform in the subsequent sections, the other essential background information can be found in Chapter \ref{chapter:background}.

### 5.2.1 Federated Learning

The number of Internet of Things (IoT) devices embedded with sophisticated sensors is growing at an unprecedented rate. On the one hand, the distributed devices are producing a massive amount of data about the environment, daily-living, health, well-being, manufacturing, and more; at the same time, the computational power of edge devices is significantly improving. As the work in the thesis is on learning representations for sensors, it is of high value to address a class of methods that allow us to learn models in a collaborative manner on what is usually called edge devices, or the edge of the network. This refers to fairly powerful devices nearby where the data is generated. We briefly introduce an exciting sub-field of machine learning that enables training over decentralized data residing on devices that could be dispersed over the globe but without the need to move such data to a centralized location; hence, effectively addressing issues of data ownership and privacy.

Traditionally, machine learning models are developed with data located in a centralized (or controlled) environment, such as a data center, where the data is divided across machines in a class-balanced and independent and identically distributed (i.i.d) manner. Due to the factors mentioned earlier, the idea of learning models on distributed devices without aggregating data in a central repository is rapidly gaining momentum. The nascent area of federated learning (FL) \cite{138} explores developing methods to achieve the goal of learning from highly distributed and heterogeneous data through aggregating locally trained models on remote devices (such as wearables, smartphones, and other ambient sensors) in cooperation with the central server. Optionally, edge device can collect data from user devices with embedded sensors and serves as federate: compute and communicate local model to the server. We can think of FL as an instance of distributed learning to move computation closer to the data instead of the other way around. It provides an exciting opportunity to harness the power of edge devices that otherwise merely sense and transmit the data to the server for further analysis. Nevertheless, FL poses numerous challenges in system design, asynchronous or synchronous training, unreliable communication, limited storage, device availability, non-i.i.d (and unbalanced) data, secure model aggregation, importantly, a general lack of annotations for learning supervised models.

For our purpose, we use a synchronous model update strategy which is a central element of the federated averaging \cite{138} algorithm. Considering the supervised learning formulation with data \( D \in \{(x, y)\}_{m=1}^{M} \), a neural network \( f_\theta(.) \) with parameters \( \theta \), a loss function \( L \), we assume \( \ell_m(\theta) = L(x_m, y_m; \theta) \) the loss of prediction on an instance \( m \) made with a
model $f_{\theta}(\cdot)$. We further assume in a federated setting data is partitioned over $\mathcal{C}$ clients, each having $m_c$ local instances with $m_T = \sum_c m_c$ being the set of total instances. Then, the federated optimization objective can be specified as follows:

$$ \min_{\theta} \ell(\theta), \text{ where } \ell(\theta) \overset{\text{def}}{=} \sum_{c=1}^{\mathcal{C}} \frac{m_c}{m_T} \times F_c(\theta) $$

with

$$ F_c(\theta) = \frac{1}{m_c} \sum_{i=1}^{m_c} \ell_i(\theta) $$

In a nutshell, the federated learning proceeds for $t \in \{1, \ldots, T\}$ communication rounds as follows: At the beginning, a model is randomly initialized with parameters $\theta_0$ and a fixed architecture on a central server. A set of $c$ clients are selected from a pool $\mathcal{C}$, which could be based on certain parameters or conditions, e.g., battery power. The server shares a model $f_{\theta_{t-1}}$ with the chosen devices for training round $t$, which perform model updates locally using stochastic gradient descent with local learning rate, batch size, and a number of epochs. The devices send updated models back to the server once local training is finished. Finally, the server computes an update for the global model through aggregation as:

$$ \theta_{t+1} \leftarrow \theta_{t-1} - \sum_{c=1}^{\mathcal{C}} \frac{m_c}{m_T} \theta_{c,t} $$

This process is repeated for several training rounds until convergence. We use federated averaging in conjunction with self-supervised learning to tackle the issue of learning from unlabeled distributed data in Chapter 5. For a detailed treatment of the FL, associated challenges, and open problems, we refer to [139].

### 5.2.2 Wavelet Transform

While the Fourier Transform (FT) sheds light on the frequency properties of the transformed signal, the input signal’s time properties are not directly accessible from the Fourier representation. An alternative to this, which provides information about the time properties of the input signal (time locality of signal variations) is the Wavelet Transform (WT) [18]. The WT, similarly to the Short-term Fourier Transform (STFT), divides the input signal into time windows of a certain size and operates on each time window separately. Choosing a larger time window of WT gives better frequency resolution of the WT output signal, while this reduces the time resolution. Precisely, the Wavelet series gives individual coefficients of a set of orthonormal functions (wavelets, e.g., Morlet, Haar, Daubechies). Like its counterparts, this representation effectively decomposes the input signal into combinations of wavelets. Due to these compelling properties, WT has been widely used in a myriad of domains [136]. In particular, continuous WT gained significant popularity as compared to discrete counterpart since it is better at localizing time-frequency properties. A wavelet transform of a signal $x(t)$
is defined as follows:

$$T(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} x(t) \cdot \psi \left( \frac{t - b}{a} \right) dt$$ \hspace{1cm} (5.3)$$

where $\psi$ represents a wavelet function, $a$ and $b$ denote scaling and translation factors, respectively. It is important to note that although we utilize WT in this work but other approaches like STFT could also be used in conjunction to possibly improve the performance along with segmentation \[140].

5.3 Approach

Learning multi-sensor representations with deep networks requires a large amount of well-curated data, which is made difficult by the diversity of device types, environmental factors, inter-personal differences, privacy issues, and annotation cost. We propose a self-supervised auxiliary task whose objective at a high level is to contrast or compare raw signals and their corresponding scalograms (which are a visual representation of the wavelet transform) so that a network learns to discriminate between aligned and unaligned scalogram-signal pairs. The rationale of the proposed approach is similar in spirit to cross-view learning in the audio-visual domain \[87] but differs in a core way that we obtain aligned and unaligned views from the same modality with wavelet transform. In the absence of the semantic labels, our methodology can be leveraged to generate an endless stream of labeled data. Therefore, it can train the network without any human involvement which is particularly attractive for the on-device learning. In subsequent sections, we describe details of the correspondence learning, sample generation, preference of a loss function, and key network architectural properties.

5.3.1 Scalogram-Signal Correspondence Learning

The idea behind scalogram-signal correspondence learning (SSCL) is to learn network parameters with a self-supervised objective that determines whether a raw signal and a scalogram correspond (or align) with each other or not. Given a multi-sensor dataset with fixed-length input segments of multiple modalities $D = \{(x, \ldots, x)_1, \ldots, (x, \ldots, x)_M\}$ of $M$ instances, we train a multi-modal contrastive network to achieve the above specified objective. Specifically, a time-series is segmented into a fixed size input with a sliding window having a certain overlap between samples. Afterward, the scalogram $s$ of a signal $x$ can be generated with a specified wavelet transformation $\Psi$. This procedure results in synchronized pairs for each $x_m$ and $s_m$ of $m$-th instance. These co-occurring pairs of inputs are assigned a class label $y_m = 1$, i.e., representing in-sync examples. Likewise, for generating negative samples $y_m = 0$, for a particular $x_m$, a randomly selected $s_m$ is assigned, which in principle represents that these scalogram-signal pairs do not align with each other. Here, we sample a negative scalogram from the same input modality. However, it can also be selected from a different modality, e.g., for accelerometer, the scalogram of the gyroscope can also be utilized. Importantly, we utilize or in-sync and out-of-sync samples
Figure 5.1: Scalogram contrastive network. We design a dual-stream architecture to learn from the raw input signal and its complementary view i.e. a scalogram. We map the original signal fragments into another domain and train the network to recognize which pairs belong together. Within this work, we use a wavelet transform. The high-level overview of the method is illustrated in (a) where signal and scalogram networks are also multi-stream networks with a distinct stream for each input modality. The architecture of these modality-specific signal and scalogram networks is shown in (b) and (c), respectively.

an equal number of positive and negative instances for training the network. As described earlier, a wavelet transform provides a better multi-resolution analysis of non-stationary signals than short-time Fourier transform \[136\]. Hence, we extract a scalogram which is an absolute and squared value of a WT operation. It is achieved using a continuous Morlet WT function which is expressed as follows:

$$\psi(t) = \exp^{-\frac{t^2}{2}} \cdot \exp^{-jw_0 t} \quad (5.4)$$

where \(w_0\) denotes a central frequency of the mother wavelet.
In the broadest sense, the SSCL task requires a semantic understanding of how time-frequency information presented in a scalogram relates to a raw input signal, thus enabling the model to learn general-purpose embedding with a complementary view on the original input. We give a high-level overview of our approach in Figure 5.1. The aim here is to learn a classifier $H(\cdot)$ that can minimize an empirical loss, so $H(x_m, s_m) = y_m$. A natural choice is to cast the specified problem as a binary classification task $p(y|x, s)$ and hence, optimize a cross-entropy loss. Nevertheless, we achieve better convergence through employing a contrastive loss that pulls together embedding of positive pairs and pushes different pairs apart, as it is also shown to be improving generalization in earlier work [43]:

$$\mathcal{L} = \frac{1}{M} \sum_{m=1}^{M} (y_m) \left\| \mathcal{F}_X(x_m) - \mathcal{F}_S(s_m) \right\|^2_2 + (1 - y_m) \max(\alpha - \left\| \mathcal{F}_X(x_m) - \mathcal{F}_S(s_m) \right\|_2, 0)^2 \quad (5.5)$$

where $\alpha$ is a margin hyperparameter which is enforced between positive and negative samples, $\mathcal{F}_X$, and $\mathcal{F}_S$ are signal and scalogram networks, respectively. The contrastive loss optimization solves the proposed self-supervised task through the integration of not just different views of the same underlying signal, but it also aligns samples across multiple sensory modalities. This label-free correspondence learning approach results in rich representations that may be invariant to sensor noise, amplitude (or scale) variations, user-specific differences, and other factors.

### 5.3.2 Network Architecture

To tackle the SSCL task, we design a dual-stream architecture named *scalogram contrastive network*, as illustrated in Figure 5.1. It is composed of two distinct parts: the scalogram network and the signal network, each extracting features from its respective inputs. As the aim here is to learn representations from multiple sensors, each network consists of modality-specific and fusion layers to learn specialized and joint embedding, respectively. In particular, we utilize the same network architecture for learning on different datasets unless mentioned otherwise. Likewise, only the features from the signal network are used for evaluation; discarding the scalogram network after pretraining.

The scalogram network consists of three 2D convolution layers with kernel sizes of 5, 4, 3, and 32, 64, 96 feature maps, respectively. Dropout is applied after every layer and max-pooling after the initial two convolutional layers with a pooling size of 2. We use the same design for each input modality, followed by the fusion layer consisting of 128 feature maps with a kernel size of 3. To learn from raw signals, we use 1D convolutional network with the same structure as scalogram network but with crucial differences in kernel sizes which are 10, 8, and 6 for sensor-specific layers and 4 in case of a shared layer with a dropout layer at the end. Moreover, we use additional pretraining related layers for both networks, comprising of a convolutional layer with 128 features maps and a dense layer with 256 hidden units. These layers are discarded after the self-supervised learning phase as we hypothesize that they might learn features relevant to the auxiliary task (i.e. SSCL). We use Mish [141] activation function in all the layers except the last, which has either linear or softmax activation. Finally, the input
to our scalogram network are coefficients of the wavelet transform with a size \((h \times w \times c)\), each representing height, width, and the number of channels, respectively. The signal network directly processes raw input of size \((w \times c)\).

### 5.3.3 Implementation Details

For pretraining, we sample the non-corresponding scalogram-signal examples through randomly selecting scalograms from outside the current input batch while keeping the raw input fixed for positives and negatives. We preprocess the signals before computing scalogram or initiating network training as done in the previous works for each considered dataset; further details are provided in Section 5.4.1. We calculate summary statistics for z-normalization from the training set. We use an Adam optimizer with a fixed learning rate of 0.0001 for pretraining and 0.01 or 0.02 in case of learning a linear classifier, which could also be decayed based on performance on the validation set. The network is trained with a batch size of 24, a dropout rate of 0.1, and L2 regularization rate of 0.0001. Importantly, for federated learning simulation, we use the Tensorflow federated learning framework. In this case, the networks are trained with a batch size of 12 for 5 local epochs using data of \(n\) randomly selected users at

---

²https://www.tensorflow.org/federated
Table 5.1: Summary of datasets.

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>#Users</th>
<th>#Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep Stage Scoring</td>
<td>Sleep-EDF</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>Activity Recognition</td>
<td>HHAR</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>MobiAct</td>
<td>61</td>
<td>11</td>
</tr>
<tr>
<td>Device-Free Sensing</td>
<td>WiFi-CSI</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Stress Detection</td>
<td>WESAD</td>
<td>15</td>
<td>3</td>
</tr>
</tbody>
</table>

each training round with 30 – 50 rounds in total, depending on the dataset size. A high-level overview of federated learning is illustrated in Figure 5.2.

5.4 Experiments

We evaluate the effectiveness of our approach in multiple ways with several publicly available datasets from different domains. First, we probe the quality of representations with a linear classifier trained on-top of a frozen feature extractor in both central and federated learning settings. Second, we examine whether scalogram-signal correspondence learning could be used to improve the recognition rate in the low-data regime. Finally, we determine the transferability of features on related datasets, followed by an evaluation with cross-validation to determine robustness against subject variations.

5.4.1 Datasets and Preprocessing

We experimented with learning models on 5 datasets from the following application areas: sleep stage scoring, human activity recognition, WiFi sensing, and physiological stress detection.

The electroencephalogram (EEG) and electrooculography (EOG) signals are used from the PhysioNet Sleep-EDF dataset for classifying sleep into five stages (i.e., Wake, N1, N2, N3, and Rapid Eye Movement). We preprocess these signals, which are recorded at 100Hz, as done in earlier work and utilize 30-second epochs (segments). For activity classification with smartphones, accelerometer and gyroscope signals from HHAR and MobiAct datasets are used, which have 6 and 11 output classes, respectively. We segment the raw signals through a sliding window into a segment size of 400 samples with a 50% overlap between them. For device-free sensing of daily activities, we use the WiFi channel state information data and follow identical preprocessing steps with . Notably, the signals are resampled from 1kHz to 500Hz through uniform temporal downsampling with a rate of 2 for each of the 90 channels (i.e., 30 sub-carriers per antenna) to classify them into 7 classes.

The WESAD dataset is used for the detection of stress, normal, and amusement physiological states. Here, we use blood volume pulse, electrodermal activity, and temperature signals collected from a wrist wearable device at 64Hz, 4Hz, and 4Hz, respectively. Following , we extract 30-seconds segments and independently normalize each subject’s data before the model development phase.
Table 5.2: Performance evaluation of self-supervised representations learned in a standard central setting with a linear classifier.

<table>
<thead>
<tr>
<th></th>
<th>Sleep-EDF</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>WiFi-CSI</th>
<th>WESAD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F-score</td>
<td>Kappa</td>
<td>F-score</td>
<td>Kappa</td>
<td>F-score</td>
</tr>
<tr>
<td>Random Init.</td>
<td>0.67</td>
<td>0.54</td>
<td>0.64</td>
<td>0.58</td>
<td>0.65</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.82</td>
<td>0.76</td>
<td>0.73</td>
<td>0.69</td>
<td>0.95</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.75</td>
<td>0.66</td>
<td>0.69</td>
<td>0.63</td>
<td>0.80</td>
</tr>
<tr>
<td>SCN</td>
<td>0.78</td>
<td>0.70</td>
<td>0.82</td>
<td>0.79</td>
<td>0.91</td>
</tr>
</tbody>
</table>

5.4.2 Results

In all the cases, we use a random 70% – 30% split of the dataset (based on users such that there is no overlap in terms of user’s data) for training and evaluation, respectively. We also a pick 20% from training split as a validation set for hyperparameter tuning and model selection. Moreover, we also evaluate the performance of our approach with cross-validation based on user split i.e., leave-one-user-out. Table 5.1 summarizes the key characteristics of the datasets used in our evaluation.

Quality assessment of the learned features with separability analysis

In Table 5.2 and Table 5.3, we provide our key evaluation results in central and federated learning settings. First, we compare the performance of our approach with a) supervised network trained end-to-end, b) an autoencoder, and c) a randomly initialized network in a central setting i.e., when entire data are available for learning on a server. We measure the quality of learned representations through a linear classifier trained on-top of the frozen extractor which is a standard evaluation protocol used in earlier work. In the federated setting (table 5.3), the supervised network is learned for each user and the weights are aggregated to create a unified model. For an autoencoder and SCN, the pretraining is performed in a federated setting to learn representations, and a classifier is trained in a standard way i.e. as if the data of end-task are available on the server. In addition, we also assess the performance when unsupervised networks are kept frozen and classifier is also learned in a federated setting. In Table 5.2 these entries are represented with FC which is an abbreviation of a federated classifier.

On the evaluated datasets, we observe that the classifiers learned on-top of a fixed randomly initialized network achieves F-score above 60% in most of the cases. It highlights the representational capacity of our architecture design that, without seeing any samples, the encoder can provide reasonable embedding for a linear classifier. Notably, the SCN surpasses results of pretraining with the autoencoder and on HHAR achieves better F-score (82.7) than a supervised baseline (73.0). Particularly, we notice that the results obtained in a federated setting are close to those achieved with learning end-to-end models in a central setting which hints towards the robustness of our approach in a federated environment. Similarly, when a linear classifier is also trained in federated setting the performance of SCN is largely consistent with the centralized classifier, which is simply not the case for an autoencoder. Moreover, in Figure 5.3 we provide the t-SNE embedding of SCN on 1000 randomly selected instances from a test set of Sleep-EDF, WiFi-CSI, and HHAR. The distinct clusters of data points can be seen that are discovered entirely in an unsupervised manner. This further highlights the ability of
**Table 5.3:** Assessing performance in a federated learning setting to determine SCN's ability to learn representations from distributed data. The entries marked with FC (federated classifier) denotes metrics when both representations and classifier are learned in a federated context.

<table>
<thead>
<tr>
<th></th>
<th>Sleep-EDF</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>WiFi-CSI</th>
<th>WESAD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F-score</td>
<td>Kappa</td>
<td>F-score</td>
<td>Kappa</td>
<td>F-score</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.82</td>
<td>0.76</td>
<td>0.77</td>
<td>0.73</td>
<td>0.94</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.76</td>
<td>0.68</td>
<td>0.71</td>
<td>0.66</td>
<td>0.82</td>
</tr>
<tr>
<td>SCN</td>
<td>0.78</td>
<td>0.72</td>
<td>0.81</td>
<td>0.77</td>
<td>0.90</td>
</tr>
<tr>
<td>Autoencoder (FC)</td>
<td>0.68</td>
<td>0.61</td>
<td>0.44</td>
<td>0.34</td>
<td>0.67</td>
</tr>
<tr>
<td>SCN (FC)</td>
<td>0.77</td>
<td>0.69</td>
<td>0.48</td>
<td>0.48</td>
<td>0.82</td>
</tr>
</tbody>
</table>

SCN in learning meaningful representations.

In Figure 5.4, we compare the performance of downstream task classifiers trained on embedding from two different parts of the network. The representations from the encoder $e$ and the features from the penultimate layer of SCN $h(e)$ are used for this purpose. It can be seen that the classifier trained on the output of $e$ performs significantly better than the one learned using the last layer’s features. We think it could be because that layers at the end might learn auxiliary task-specific features which are not useful enough for the end-task.

### Improving generalization in low-data regime and transfer learning

We explore the effectiveness of the proposed technique for improving performance with a few labeled data. We pretrain a scalogram-contrastive network with the entire unlabeled data and use the model as initialization for learning a downstream task. We compare the performance with a standard supervised network trained only with certain labeled instances. Specifically, we use 5, 10, 20, and 40 labeled instances per class to learn the end-task model. Figure 5.3 and Table 5.4 show an average F-score of 100 independent runs, where, at each run, different examples are sampled to train the network. In all the cases, the results obtained with utilizing a self-supervised network are better than the baseline even when small labeled data are available. This highlights that the SCN efficiently harnesses unlabeled data to learn generalized features.

Similarly, the self-supervised networks are also evaluated in terms of their usefulness in a transfer setting. Generally, this is achieved by treating a pretrained model as a fixed feature extractor, and a linear model is trained on top of it using a different dataset. Here, we assess the performance on activity recognition tasks with HHAR and MobitAct datasets. Table 5.3 provides these results and compares with the supervised network, transfer from supervised (Sup.), and SCN trained on the same source instances. In both cases, we see that the recognition rate improves if transferred embedding is from SCN compared to a supervised network. Finally, we also assess the performance of SCN when few-labeled instances are available for fine-tuning but different unlabeled data are available for pretraining in Table 5.6. Similar to earlier semisupervised evaluation, we fine-tune a pretrained network end-to-end with 5, 10, 20, and 40 examples of each class from the target dataset. We notice a 2% – 3% improvement in F-score over the supervised network when an SCN encoder is utilized.
Figure 5.3: t-SNE embedding learned with scalogram contrastive network on a random subset of test subjects. Note, t-SNE does not utilize class labels, the colors are added during post-hoc analysis for better interpretability.

Figure 5.4: Performance comparison of linear classifiers trained on-top of representations from encoder ($e$) and penultimate layer's projection $h(e)$ of SCN denoted with $fc$ 256 in Figure 5.1.

Table 5.4: Generalization improvement in semi-supervised setting with self-supervised pretraining.

<table>
<thead>
<tr>
<th></th>
<th>Sleep-EDF</th>
<th>HHAR</th>
<th>MobiAct</th>
<th>WiFi-CSI</th>
<th>WESAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>$0.14 \pm 0.01$</td>
<td>$0.84 \pm 0.01$</td>
<td>$0.42 \pm 0.07$</td>
<td>$0.51 \pm 0.06$</td>
<td>$0.46 \pm 0.06$</td>
</tr>
<tr>
<td>SCN</td>
<td>$0.62 \pm 0.06$</td>
<td>$0.61 \pm 0.07$</td>
<td>$0.48 \pm 0.04$</td>
<td>$0.43 \pm 0.03$</td>
<td>$0.49 \pm 0.06$</td>
</tr>
</tbody>
</table>

Robustness against subject variation with cross-validation

To determine the robustness of network pretraining with the proposed approach against subject variation, we perform cross-validation (CV) based on user split. For Sleep-EDF, HHAR, and WESAD leave-one-subject-out CV is employed, whereas for MobiAct and WiFi-CSI, a 10-fold stratified CV is used due to a large number of users in the former and unavailability of subject ID’s in the latter. We follow the same evaluation strategy as earlier, i.e., training a lin-
Figure 5.5: Effectiveness of self-supervised learning in a low-data regime. The SCN is pretrained on unlabeled data and fine-tuned end-to-end with few labeled data points (i.e., 5, 10, 20, and 40 instances per class). On all the evaluated datasets, we notice a significant performance improvement over a supervised baseline network which is trained only with labeled inputs.

Table 5.5: Evaluation of self-supervised representation in a standard transfer learning setting.

<table>
<thead>
<tr>
<th></th>
<th>HHAR → MobiAct</th>
<th>MobiAct → HHAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F-score</td>
<td>Kappa</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.95</td>
<td>0.93</td>
</tr>
<tr>
<td>Source (SCN)</td>
<td>0.91</td>
<td>0.88</td>
</tr>
<tr>
<td>Transfer (Sup.)</td>
<td>0.86</td>
<td>0.83</td>
</tr>
<tr>
<td>Transfer (SCN)</td>
<td>0.87</td>
<td>0.84</td>
</tr>
</tbody>
</table>

Table 5.6: Fine-tuning transferred model with few-labeled data to improve recognition rate. We report weighted F-score averaged over 100 independent runs. T denotes a transfer learning.

<table>
<thead>
<tr>
<th></th>
<th>HHAR → MobiAct</th>
<th>MobiAct → HHAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Supervised</td>
<td>SCN (T)</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>0.51</td>
</tr>
<tr>
<td>10</td>
<td>0.56</td>
<td>0.59</td>
</tr>
<tr>
<td>20</td>
<td>0.62</td>
<td>0.65</td>
</tr>
<tr>
<td>40</td>
<td>0.68</td>
<td>0.70</td>
</tr>
</tbody>
</table>

ear classifier to assess the quality of representations as compared to the fully-supervised model and an autoencoder. Table 5.7 summarizes mean and standard deviation of metrics averaged over folds. Overall, we notice that SCN is stable despite the changes of subject data in a train-
Table 5.7: Comparison of self-supervised representations to a fully-supervised network and pretraining with autoencoder using cross-validation.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Supervised F-score ± Kappa</th>
<th>Autoencoder F-score ± Kappa</th>
<th>SCN F-score ± Kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep-EDF</td>
<td>0.83 ± 0.05</td>
<td>0.71 ± 0.08</td>
<td>0.62 ± 0.10</td>
</tr>
<tr>
<td>HHAR</td>
<td>0.82 ± 0.12</td>
<td>0.64 ± 0.13</td>
<td>0.59 ± 0.15</td>
</tr>
<tr>
<td>MobiAct</td>
<td>0.94 ± 0.02</td>
<td>0.79 ± 0.04</td>
<td>0.71 ± 0.06</td>
</tr>
<tr>
<td>WiFi-CSI</td>
<td>0.97 ± 0.00</td>
<td>0.81 ± 0.01</td>
<td>0.82 ± 0.02</td>
</tr>
<tr>
<td>WESAD</td>
<td>0.76 ± 0.11</td>
<td>0.71 ± 0.14</td>
<td>0.56 ± 0.25</td>
</tr>
</tbody>
</table>

ing set and achieves significantly better results than an autoencoder. Notably, on Sleep-EDF, our methods achieve a mean kappa score of 0.83 as compared to 0.77 of a supervised network and 0.76 as reported in [108]. Likewise, our self-supervised technique performs better than the hand-designed features from wrist physiological signals on WESAD by achieving F-score of 75.7 ± 0.13 as compared to 66.33 ± 0.36 [134]. Furthermore, we would like to highlight that a direct comparison of existing approaches on other datasets used in our study is not feasible due to the differences in reported metrics and used sensing modalities. Nevertheless, our results with cross-validation further indicate that self-supervised learning can be effectively utilized for sensor modeling tasks on a large-scale and it can be combined with active learning methods [143].

5.5 Conclusion

We propose a self-supervised method for learning representations from unlabeled multi-sensor input data, which is typical in the IoT setting. Our method utilizes wavelet transform to generate a complementary view of the input (i.e., a scalogram) to define an auxiliary task of scalogram-signal correspondence. This procedure is specifically designed to work in a federated learning setting to allow training networks with widely distributed and unannotated data as the labels can be readily extracted from the data without human-in-the-loop. We show the efficacy of the developed technique on several publicly available datasets involving diverse sensory streams, such as electroencephalogram, blood volume pulse, and IMUs. Particularly, we evaluate the quality of learned features with a linear classifier on an end-task and compare the performance with a fully-supervised network and pretraining with an autoencoder in both federated and central settings. Furthermore, we demonstrate an improved generalization in the low-data regime with self-supervision, i.e., when few labeled instances are used for fine-tuning network on the desired end-task. Our generic self-supervised approach can be used efficiently for learning general-purpose deep feature extractors entirely on-device without the need of transmitting the actual data to the server. In future work, we plan to combine self-supervision with architecture search on larger datasets and evaluate our method in a non i.i.d setting for federated learning.

---

Various icons used in the figure are created by Andrejs Kirmia, Stefan Traistaru, Graphic Tigers, Icon Lauk, Olena Panasovska and others from the Noun Project.
Chapter 6

Contrastive Learning of General-Purpose Audio Representations

This chapter is based on the material from our paper Contrastive Learning of General-Purpose Audio Representations accepted at IEEE ICASSP 2021 [18] and a joint work with David Grangier and Neil Zeghidour. This work was done during an internship at Google Research within Brain team, Paris.

6.1 Introduction

Having explored representation learning for multivariate time-series (or signals) acquired from sensors embedded in smartphones and other wearable devices in preceding chapters, in this chapter we focus on extracting useful representations from unlabeled audio to solve diverse set of sound recognition problems.

Self-supervised pretraining has recently emerged as a successful technique to leverage unlabeled data to learn representations beneficial to supervised problems. This success spans a wide range of tasks and modalities [38, 42, 87, 144]. Among these methods, Discriminative pretraining (DPT) is particularly effective. This approach learns a representation from pairs of similar inputs from unlabeled data, exploiting e.g. temporal consistency [87, 143, 146] or data augmentation [147] and trains a model to recognize similar elements among negative distractors. In contrast with generative encoder-decoder approaches [148, 149, 150, 151, 152], DPT is computationally efficient as it avoids input reconstruction entirely.

Amidst DPT models for audio, [146] used a metric learning approach with a triplet loss to minimize the distance between embeddings of anchor and positive pairs and maximize it
among the negatives. The instance generation is achieved through noise injection, shifting along time-frequency dimensions, and extracting samples in temporally close neighborhoods. Along similar lines, [153] proposed a benchmark for comparing speech representations on non-semantic tasks. Through utilizing a triplet loss as an unsupervised objective with a subset of AudioSet [154] for model training, they showed improved performance on several downstream speech classification tasks. Inspired from seminal work in NLP [155], the work in [98] adopted a similar approach to learn audio representations (i.e. Audio2Vec) along with another “pretext” task of estimating temporal distance between audio segments. The pretrained models are tested on several downstream tasks, from speaker identification to music recognition. Despite recent progress, most work on learning representations of audio focuses on speech tasks [156, 157, 158] (with the exception of [98, 146]) and ignores other audio tasks such as acoustic scene detection or animal vocalizations. Moreover, triplet-based objectives heavily rely on the mining of negative samples, and the quality of learned features can vary significantly with the sample generation scheme.

In this work, we propose COLA (COntrastive Learning for Audio), a simple contrastive learning framework to learn general-purpose representations of sounds beyond speech. We build upon recent advances in contrastive learning [42] for computer vision (SimCLR [47], MoCo [59]) and reinforcement learning (CURL [160]). We generate similar pairs by simply sampling segments from the same audio clip, which avoids exploring augmentation strategies entirely unlike SimCLR, MoCo, CURL and others [64]. Our dissimilar pairs simply associate segments from different clips in the same batch, which does not require maintaining a memory bank of distractors as in MoCo. Our approach allows us to consider a large number of negatives for each positive pair in the loss function and bypass the need for a careful choice of negative examples, unlike triplet-based approaches [146, 153]. COLA is also different from CPC [42] as it does not predict future latent representations from past ones.

We demonstrate the effectiveness of COLA over challenging and diverse downstream tasks, including speech, music, acoustic scenes, and animal sounds. After pretraining on the large-scale AudioSet database [154], we show that a linear classifier trained over a COLA embedding gets close to the performance of a fully-supervised in-domain convolutional network and exceeds it when using fine-tuning. Moreover, our system outperforms previous unsupervised approaches on most downstream tasks. These experiments demonstrate that COLA offers a simple, easy-to-implement method to learn general-purpose audio representations without supervision.

6.2 Approach

We learn general-purpose audio representations from unlabeled data by pretraining a neural network with a contrastive loss function. Our objective function maximizes an agreement between the latent embedding of segments extracted from the same audio clip while using different audio clips as negative classes, as shown in Figure 6.1. This objective pretrains a convolutional feature extractor on unlabeled audio data. After pretraining, we combine our feature extractor with an additional classification layer for solving various audio understanding tasks across several datasets.
Figure 6.1: Overview of the contrastive self-supervised learning for audio.

Contrastive learning extracts a latent space in which the similarity between an anchor example and a related example should be greater than the similarity between the same anchor and unrelated examples. In our case, an anchor and its corresponding positive are audio segments from the same clip. This contrasts with approaches that generate positives as perturbations of the anchor \[161, 162\]. For negative examples, we take segments from different audio clips in the current training batch. This strategy allows to consider a large number of negatives and is efficient since batch examples are used both as positives and negatives without additional computation.

COLA computes the similarity between audio segments in two steps. First, an encoder $f$ maps a log-compressed mel-filterbanks $x \in \mathbb{R}^{N \times T}$, with $N$ and $T$ the number of frequency bins and time frames respectively, into a latent representation $h = f(x) \in \mathbb{R}^d$. This is the representation that we will transfer to downstream classification, after pretraining. Then, a shallow neural network $g$ maps $h$ onto a space $z = g(h)$, where bilinear comparisons are performed. If we denote with $W$ the bilinear parameters, the similarity between two segments $(x, x')$ is, therefore:

$$s(x, x') = g(f(x))^\top W g(f(x')).$$  \hspace{1cm} (6.1)

Bilinear similarity has been used in the past \[42\] but is less common than cosine similarity, e.g. SimCLR and MoCo. In Section \ref{sec:ablation}, we perform an ablation study on the choice of similarity measure. Table \ref{table:comparison} shows that a bilinear similarity outperforms a simple cosine similarity \[g(f(x))^\top g(f(x'))/\|g(f(x))\|\|g(f(x'))\|\] on all downstream tasks. In the rest of this chapter, we use this method unless stated otherwise.

As an objective function, we rely on multi-class cross entropy applied to similarities, i.e.

$$\mathcal{L} = - \log \frac{\exp(s(x, x^+))}{\sum_{x^- \in X^-(x) \cup \{x^+\}} \exp(s(x, x^-))}$$  \hspace{1cm} (6.2)
where \( x^+ \) is the positive associated to anchor \( x \), while \( X^- (x) \) refers to the set of negative distractors. This loss, unlike the triplet loss [163], leverages multiple distractors at a time.

As mentioned earlier, we train our model with positive segment pairs sampled from the same audio clip. For each pair, we use one segment as the anchor and the other element as the positive. Positive segments are used as negatives for all other anchors in the batch. This strategy is more efficient than keeping a memory bank of negatives [159, 62] since the representation of an example is paired with every anchor in the batch either as a positive or as a negative segment. In particular, we experiment with batch sizes varying from 256 to 2048, as shown in Table 6.4. A large batch size allows the model to see many negative samples per anchor and helps accuracy on end tasks. It is important to note that we sample segment pairs on-the-fly and reshuffle the data at each training epoch to maximize the diversity of positive and negative pairs seen during training. The sample generation procedure is illustrated in Figure 6.1.

### 6.3 Experiments

We evaluate our method by pretraining COLA embeddings on a large-scale audio dataset and then transferring it to downstream tasks in the following ways: 1) training a linear classifier on top of a frozen embedding, used as a feature extractor and 2) fine-tuning the entire network on the end-task. Importantly, we assess the performance on several diverse datasets to determine the transferability of learned representations across audio domains and recording conditions.

#### 6.3.1 Datasets and Tasks

We pretrain COLA embeddings on the diverse, large-scale Audioset [154]. It contains 2 millions excerpts of 10 seconds audio from YouTube videos that are annotated in a multi-label fashion with over 500 classes. This dataset has been used by [98, 153, 164] for self-supervised pretraining. Since our method is self-supervised, we never use Audioset labels. As described earlier, we randomly sample audio clips to generate examples. Likewise, for the extraction of anchors and positives, segments of audio are selected uniformly at random inside a sequence.

We perform downstream evaluation on a variety of tasks, including both speech and non-speech. To allow for comparison with previous methods, we rely on datasets that have been previously used by [98, 153, 164]. For speaker identification, we use a 100-hours subset of LibriSpeech (LBS) [165] that contains audio of books read by 251 speakers, as well as the Voxceleb [166] subset used in [153], with 1,251 speakers. For keyword spotting, we use Speech Commands (SPC) [167] V1 and V2 to recognize 11 and 35 spoken commands (classes) from one second of audio, respectively. For acoustic scene classification, we use TUT Urban Acoustic Scenes 2018 (TUT) [168], consists of labeled audio segments from 10 different acoustic scenes. For animal vocalizations, we use the Bird Song Detection (BSD) dataset [169] from DCASE 2018 Challenge to solve a binary classification problem. For music recognition, we use MUSAN [170] that differentiates audio samples across 3 classes (speech, music and noise), as well as the NSynth dataset [171] of musical notes, labeled with the family of the instrument
Table 6.1: Test accuracy (%) on downstream tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Random Init.</th>
<th>Supervised Frozen</th>
<th>COLA Fine-tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speaker Id. (LBS)</td>
<td>0.4</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Speech commands (V1)</td>
<td>62.9</td>
<td>97.2</td>
<td>71.7</td>
</tr>
<tr>
<td>Speech commands (V2)</td>
<td>4.0</td>
<td>94.3</td>
<td>62.4</td>
</tr>
<tr>
<td>Acoustic scenes</td>
<td>8.6</td>
<td>98.2</td>
<td>94.1</td>
</tr>
<tr>
<td>Speaker Id. (Voxceleb)</td>
<td>0.0</td>
<td>31.7</td>
<td>29.9</td>
</tr>
<tr>
<td>Birdsong detection</td>
<td>49.6</td>
<td>79.4</td>
<td>77.0</td>
</tr>
<tr>
<td>Music, Speech and Noise</td>
<td>56.8</td>
<td>99.3</td>
<td>99.1</td>
</tr>
<tr>
<td>Language Id.</td>
<td>59.1</td>
<td>85.0</td>
<td>71.3</td>
</tr>
<tr>
<td>Music instrument</td>
<td>20.8</td>
<td>70.7</td>
<td>63.4</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>29.1</strong></td>
<td><strong>83.9</strong></td>
<td><strong>74.3</strong></td>
</tr>
</tbody>
</table>

(11 classes). For language identification, we use the Voxforge dataset [172] to categorize audio clips into six classes based on the spoken language.

Table 6.2: Test accuracy (%) of a linear classifier trained on top of COLA embeddings or baseline pretrained representations.

<table>
<thead>
<tr>
<th></th>
<th>CBoW 17, 164</th>
<th>SG 17, 164</th>
<th>TemporalGap 17, 164</th>
<th>Triplet Loss 17, 164</th>
<th>TRILL 153</th>
<th>COLA 100.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speaker Id. (LBS)</td>
<td>99.0</td>
<td>100.0</td>
<td>97.0</td>
<td>100.0</td>
<td>-</td>
<td>100.0</td>
</tr>
<tr>
<td>Speech commands (V2)</td>
<td>30.0</td>
<td>28.0</td>
<td>23.0</td>
<td>18.0</td>
<td>-</td>
<td>61.4</td>
</tr>
<tr>
<td>Acoustic scenes</td>
<td>66.0</td>
<td>67.0</td>
<td>63.0</td>
<td>73.0</td>
<td>-</td>
<td>94.1</td>
</tr>
<tr>
<td>Birdsong detection</td>
<td>71.0</td>
<td>69.0</td>
<td>71.0</td>
<td>73.0</td>
<td>-</td>
<td>77.0</td>
</tr>
<tr>
<td>Music, Speech and Noise</td>
<td>98.0</td>
<td>98.0</td>
<td>97.0</td>
<td>97.0</td>
<td>-</td>
<td>99.1</td>
</tr>
<tr>
<td>Music instrument</td>
<td>33.5</td>
<td>34.4</td>
<td>35.1</td>
<td>25.7</td>
<td>-</td>
<td>61.4</td>
</tr>
<tr>
<td>Speech commands (V1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>74.0</td>
<td>71.7</td>
<td></td>
</tr>
<tr>
<td>Speaker Id. (Voxceleb)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>17.7</td>
<td></td>
</tr>
<tr>
<td>Language Id.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>88.1</td>
<td>71.3</td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong> (TRILL tasks)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>59.9</td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong> (non-TRILL)</td>
<td>66.25</td>
<td>66.0</td>
<td>64.1</td>
<td>64.4</td>
<td>-</td>
<td>82.5</td>
</tr>
</tbody>
</table>

Table 6.3: Test accuracy (%) with different similarity functions.

<table>
<thead>
<tr>
<th></th>
<th>Cosine Similarity</th>
<th>Bilinear Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speaker Id. (LBS)</td>
<td>99.9</td>
<td>100.0</td>
</tr>
<tr>
<td>Speech commands (V1)</td>
<td>64.5</td>
<td>71.7</td>
</tr>
<tr>
<td>Speech commands (V2)</td>
<td>42.4</td>
<td>62.4</td>
</tr>
<tr>
<td>Acoustic scenes</td>
<td>87.5</td>
<td>94.1</td>
</tr>
<tr>
<td>Speaker Id. (Voxceleb)</td>
<td>15.2</td>
<td>29.9</td>
</tr>
<tr>
<td>Birdsong detection</td>
<td>76.5</td>
<td>77.0</td>
</tr>
<tr>
<td>Music, Speech and Noise</td>
<td>99.0</td>
<td>99.1</td>
</tr>
<tr>
<td>Language Id.</td>
<td>62.3</td>
<td>71.3</td>
</tr>
<tr>
<td>Music instrument</td>
<td>58.3</td>
<td>63.4</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>67.2</td>
<td>74.3</td>
</tr>
</tbody>
</table>

93
6.3.2 Model Architecture and Implementation Details

Given an audio input sequence, we extract log-compressed mel-filterbanks with a window size of 25 ms, a hop size of 10 ms, and $N = 64$ mel-spaced frequency bins in the range 60–7800 Hz for $T = 96$ frames, corresponding to 960 ms. These features are passed through an encoder $f$ based on EfficientNet-B0 [173], a lightweight and highly scalable convolutional neural network. Even though EfficientNet-B0 has been originally proposed for computer vision, the 2D structure of mel-filterbanks allows using this architecture without any adjustment. We apply a global max-pooling to the last layer of the encoder to get an embedding $h$ of size 1280. During pretraining, we pass $h$ through the projection head $g$, which contains a fully-connected layer with 512 units followed by a Layer Normalization [174] and a tanh activation. We discard the projection head for the downstream tasks and train a linear classifier on top of the encoder directly. We pretrain all our models with ADAM [11] and a learning rate of $10^{-4}$, for 500 epochs. We explore the impact of the batch size and report the results in Table 6.4. We train the downstream classifiers with a batch size of 64 and a learning rate of $10^{-3}$, on randomly selected 960ms segments, as for pretraining. However, we evaluate downstream classifiers on entire sequences using the following procedure: we split the sequence into non-overlapping 960ms segments, pass them through the encoder and linear classifier, and average the predictions.

6.3.3 Results

Table 6.1 reports the accuracy on the 9 downstream datasets. We compare our approach against multiple baselines: a linear classifier trained on a randomly initialized fixed encoder and a fully-supervised model trained directly on downstream datasets which indicates the performance achievable with EfficientNet-B0 on these datasets. First, we evaluate pretrained COLA embeddings with a linear classifier on top of frozen representations, following the same procedure as [42, 57, 147, 159]. This outperforms drastically the performance of a linear classifier trained on a random embedding (74.3% against 29.1% on average), showing that the encoder has learned useful representations. This is remarkable as we pretrain a single COLA embedding, which performs well across many tasks. Next, we also use a pretrained COLA as initialization and fine-tune one model per downstream task. Table 6.1 shows that on all tasks but language identification, initializing a supervised model with COLA improves the performance over training from scratch (85.1% against 83.9% on average), which demonstrates the benefits of transferring COLA representations even in a fully supervised setting.

We then compare COLA to prior self-supervised methods proposed in [58, 164], including a standard triplet loss, Audio2Vec (CBoW and SG) and temporal gap prediction models. Here, the CBoW and SG are generative models inspired from Word2Vec, trained to reconstruct a randomly selected temporal slice of log-mel spectrograms given the rest or vice versa. Likewise, TemporalGap trains a model to predict the temporal distance between two pairs of audio segments. Table 6.2 shows that COLA embeddings consistently outperform all these methods. In particular, on acoustic scene classification, we obtain a competitive accuracy of 94% compared to 73% achieved with a triplet loss in [58]. We also considerably improve the performance on speech commands and musical instrument classification by an absolute 30%
margin on both tasks. We also compare with the recent self-supervised learning framework TRILL \cite{153} on three speech-related tasks, benchmarking against TRILL-19 (the best self-supervised system of \cite{153}). Our general-purpose COLA embeddings are competitive with TRILL, despite the fact that TRILL is pretrained specifically on the part of Audioset that contains speech, and is evaluated only across speech tasks, while we train and evaluate COLA across speech, music, acoustic scenes, and animal sounds.

To investigate the role of the similarity measure in the quality of learned representations, we perform an ablation study to compare model pretraining with cosine and bilinear similarity. With the cosine similarity, we use a temperature $\tau = 0.2$ to normalize the scores before computing the loss. Table 6.3 reports the results obtained on downstream classifiers using encoders pretrained with each of the similarity estimation techniques. We observe that the best results are obtained using bilinear similarity in all cases. We also conduct an experiment to measure the impact of pretraining batch size, as larger batch sizes result in more negative samples and facilitate convergence \cite{147}. Table 6.4 shows that, on average, a batch size as large as 1024 provides better representations compared to smaller ones. However, increasing the batch size up to 2048 worsens the performance in most cases.

### 6.4 Conclusion

We introduce COLA, a simple, easy-to-implement, self-supervised contrastive algorithm for general-purpose audio representation learning. Our approach achieves remarkable performance improvements over earlier unsupervised methods on a wide variety of challenging downstream tasks in a linear evaluation protocol as well as significantly improves results over supervised baselines through fine-tuning. We believe that the simplicity of our system, combined with its strong transferability across audio tasks, will pose it as a go-to baseline for future work in self-supervised learning for audio.
Chapter 7

Differentiable Channel Reordering for Heterogeneous Signals

This chapter is based on the material from our paper Learning from Heterogeneous EEG Signals with Differentiable Channel Reordering accepted at IEEE ICASSP 2021 [19] and a joint work with David Grangier, Olivier Pietquin and Neil Zeghidour. This work was done during an internship at Google Research within Brain team, Paris under the supervision of Neil Zeghidour and David Grangier.

7.1 Introduction

In the previous chapters, we explored the problem of learning representations from unlabeled data acquired from different types of sensors. Here, we focus on the robustness aspect of the deep neural network in the face of inconsistent input channels i.e., a multi-channel input whose channels are permuted, missing or have noise, e.g., a electroencephalogram recording. Our approach is general-purpose and can be applied to any type of signal for handling inconsistent channel ordering in an end-to-end manner while learning the task, we use Electroencephalography (EEG) as the application domain in this chapter.

EEG is the measurement of the brain's electrical activity, which informs about neural functions and related physiological manifestations [175]. It is generally collected along the scalp in a non-invasive way for a wide array of tasks, including for clinical purposes and Brain-Computer Interface (BCI) systems. Automatic classification of EEG signals with machine learning has been widely adopted to study, diagnose, and treat neurological disorders such as seizures, epilepsy, Alzheimer's, and sleep-related problems [108, 176, 177]. In BCI tasks, EEG is used to capture motor imagery signals and recognize user’s intents [178] and event-related potential [179]. Likewise, it is also used to estimate mental workload or task complexity for monitoring cognitive stress and performance [180].
Over the last years, automatic EEG classification has moved from using hand-crafted features \[181, 182\] towards learning high-level representations from raw EEG signals with deep neural networks \[183, 184\]. In particular, Convolutional Neural Networks (CNNs) have become the standard architecture to process EEG signals and have been used for many tasks including motor imagery \[185, 186, 187\], seizure prediction \[188, 189\], Parkinson diagnosis \[190\] and sleep stage scoring \[108\]. Nevertheless, EEG measurements remain notoriously subject to intra- and inter-subject variability, which makes generalization particularly challenging, for a same subject between sessions, or between different subjects, and led to numerous works focusing on the reduction of this generalization gap \[191, 192, 193\]. A less explored problem is the variability due to differences in measuring devices: different EEG headsets have a varying number of electrodes (from a few to dozens) and different electrical specifications \[194\]. Moreover, it is not rare that headset malfunctions lead to noisy or even missing channels. Consequently, available EEG datasets are heterogeneous, and the majority of them are very small.

Scaling EEG training data seems, therefore, only feasible by aggregating heterogeneous datasets. This requires devising novel classification methods that are robust to permuted and missing channels since classical CNNs assume a fixed number of input channels, ordered consistently across data examples. With this objective, we introduce a new framework for training a single CNN across varying EEG collections that can differ both in number and location of electrodes. Our CHAnnel Reordering Module (CHARM) ingests multichannel EEG signals, identifies the location of each channel from their content, and remaps them to a canonical ordered set. After this remapping, the channels are ordered consistently and can be further processed by a standard neural network, regardless of the actual variations in the input data. We evaluate CHARM on three tasks: seizure classification, detection of abnormal EEGs and detection of artifacts (e.g. eye movement). We show that CHARM is significantly more robust to missing and permuted channels than a standard CNN. We also introduce a data augmentation technique that further improves the robustness of the model. Moreover, we show for the first time that pretraining representations of EEG on a large dataset transfers to another, smaller dataset collected with a different headset.

Figure 7.1: Illustration of various forms of inconsistencies that can arise in EEG recordings.

### 7.2 Approach

Our proposal is a differentiable reordering module that maps inputs with inconsistent channels to a fixed canonical order. It can be composed with further modules expecting consistent channel placement to be trained end-to-end on data without channel ordering information.
As inputs, we consider EEG signals with an unknown channel ordering and potentially missing channels, (Figure 7.1). Our module takes these channels and reorders them to a canonical, consistent order prior to further processing. Precisely, our reordering module outputs a soft reordering matrix \( p \).

The input signal \( \mathbf{x} \in \mathbb{R}^{N \times T} \) is a recording over \( N \) channels for a duration \( T \). Considering \( M \) canonical channels, the reordering matrix \( p(\mathbf{x}) \) is an \( N \times M \) matrix. Precisely, each canonical output is estimated as a weighted sum of the input channels, i.e.,

\[
\hat{\mathbf{x}}_{i,t} = \sum_{j=1}^{N} p(\mathbf{x})_{i,j} \mathbf{x}_{j,t}, \quad i = 1, \ldots, M, \quad t = 1, \ldots, T. \tag{7.1}
\]

\( \hat{\mathbf{x}} \in \mathbb{R}^{M \times T} \) then serves as input to a standard neural network expecting a consistent input ordering across data samples. Since \( p \) is differentiable, the reordering module parameters can be learned jointly with the rest of the architecture (Figure 7.2). Training optimizes the cross-entropy classification loss with no extra supervision on the channel order.

### 7.2.1 Learnable Channel Remapping

Using \( \text{CHARM} \) as the first layers of the model allows us to train a single deep architecture over different EEG recording headsets. We consider three variants of our reordering method.

**Convolutional Reordering**

\( \text{CHARM-base} \) represents the signal of each channel as a vector,

\[
h_{i} = m_{\text{conv}}(\mathbf{x}_{i,:}) \tag{7.2}
\]

where \( m_{\text{conv}} \) composes a 1-D convolution layer with \( d \) filters and an aggregation operation (global max-pooling) to map a single-channel temporal signal into a fixed dimensional vector of dimension \( d \). Since this step convolves channels independently, its predictions are invariant to a reordering of the input channels.

Each vector is then compared to learned embeddings of dimension \( d \) that represent each of the \( M \) canonical channels, \( c \in \mathbb{R}^{M \times d} \), yielding the matrix \( p \) for channel remapping,

\[
p_{i,j} = \text{softreorder}(c, h)_{i,j} = \frac{\exp(c_i \cdot h_j)}{\sum_{j'} \exp(c_i \cdot h_{j'})}. \tag{7.3}
\]

**Attentive Reordering with Canonical Keys and Values**

\( \text{CHARM-CKV} \) builds upon residual attention [25]. We build a query vector representing each input channel as \( q_i = m_{\text{conv}}(\mathbf{x}_{i,:}) \). Each query vector attends over canonical channels, i.e.
Inconsistent EEG Channels Placement

Learnable Channel Remapping (CHARM)

Remapped Canonical Channels

Classifier

**Figure 7.2:** Overview of CHARM with a 1D convolutional classifier for EEG classification tasks.

Each input channel query is mapped to a weighted sum of canonical channel value vectors according to their similarity to canonical key vectors,

\[ h_i = \sum_j a_{i,j} v_j \quad \text{where} \quad a_{i,j} = \frac{\exp(q_i \cdot k_j)}{\sum_j \exp(q_i \cdot k_j')}, \]  

(7.4)

and \( k, v \) are key and value embeddings representing the canonical channels. These layers can be stacked after a residual connection,

\[ q_{l+1}^i = \text{layernorm}(q_l^i + \text{mlp}_l(h_l^i)) \]  

(7.5)

where \( q^l, h^l \) are the query and attentive representations of layer \( l \). The layernorm denotes a layer normalization module \([174]\) and mlp denotes a multi-layer perceptron with a single hidden layer. We denote the residual attention compactly as

\[ q_{l+1}^i = \text{attn}(q_l^i, k, v) \]  

(7.6)

At the last layer, we compute \( p = \text{softreorder}(c, q^l) \).

Compared to our base convolutional model, this model can use many channel predictions to refine each individual prediction. For instance, it can be confident only for a few channels at the first layer and refine its decision for the other channels in the next layers.

**Attentive Reordering with Canonical Queries**

**CHARM-CQ** reverses the role of canonical and input channels, using input keys and values and relying on canonical queries. Input keys and values result from an independent channel-wise convolution with 1D filters,

\[ (k_i, v_i) = m^{\text{conv}}(x_i,:) \]  

(7.7)

and initial canonical queries \( q \) are learned embeddings. Each query can be attended over to represent each canonical channel as a weighted sum of input values, as in Equation 7.4. Several layers of attention can be stacked. Finally, \( p = \text{softreorder}(q^l, k^l) \). In our experiments, we found it beneficial to share keys and values across layers, i.e. \( (k^l, v^l) = (k^0, v^0) \).
Our main architecture is a 1D-CNN that takes raw EEG signals as input and cascades four blocks of 1D convolutions. Each block has a LayerNormalization and PReLU as activation along with a max-pooling layer for downsampling. We use a kernel size of 8 with stride 1 and pooling size of 2 and stride 2 with 256 feature maps for the first three blocks, and 512 in the last one. To aggregate the features, we use global max-pooling, which then feeds into a single linear classification layer. We apply L2 regularization with a rate of $10^{-4}$ on all the layers but the last. Given an input sequence, we first apply an InstanceNormalization, which normalizes each channel independently. Then our baseline model (Baseline) passes the output through the 1D-CNN directly. On the other hand, CHARM first passes the waveforms through the remapping module, which then feeds into the 1D-CNN.

### 7.2.3 Network Architecture Design and Implementation

Our main architecture is a 1D-CNN that takes raw EEG signals as input and cascades four blocks of 1D convolutions. Each block has a LayerNormalization and PReLU as activation along with a max-pooling layer for downsampling. We use a kernel size of 8 with stride 1 and pooling size of 2 and stride 2 with 256 feature maps for the first three blocks, and 512 in the last one. To aggregate the features, we use global max-pooling, which then feeds into a single linear classification layer. We apply L2 regularization with a rate of $10^{-4}$ on all the layers but the last. Given an input sequence, we first apply an InstanceNormalization, which normalizes each channel independently. Then our baseline model (Baseline) passes the output through the 1D-CNN directly. On the other hand, CHARM first passes the waveforms through the remapping module, which then feeds into the 1D-CNN.

### Channel Remapping Networks

CHARM uses 24 canonical channels, regardless of the actual number of channels in an input sequence (from 17 to 22 in our experiments). We represent each canonical channel with an

---

Table 7.1: Test accuracy (± std) averaged over 10-folds for model generalization to shuffled and masked input channels. The entries with Noisy-n% represent the results when a fixed ratio of n% of the channels are masked after shuffling.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Channels</th>
<th>Classes</th>
<th>Method</th>
<th>Clean</th>
<th>Noisy</th>
<th>Noisy-25%</th>
<th>Noisy-50%</th>
<th>Noisy-75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUH Abnormal</td>
<td>22</td>
<td>2</td>
<td>Baseline</td>
<td>0.830 ± 0.030</td>
<td>0.566 ± 0.025</td>
<td>0.581 ± 0.015</td>
<td>0.589 ± 0.027</td>
<td>0.548 ± 0.028</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-base</td>
<td>0.766 ± 0.016</td>
<td>0.742 ± 0.014</td>
<td>0.760 ± 0.013</td>
<td>0.749 ± 0.016</td>
<td>0.731 ± 0.014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-CKV</td>
<td>0.772 ± 0.019</td>
<td>0.741 ± 0.011</td>
<td>0.767 ± 0.013</td>
<td>0.756 ± 0.013</td>
<td>0.747 ± 0.014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-CQ</td>
<td>0.751 ± 0.014</td>
<td>0.743 ± 0.012</td>
<td>0.744 ± 0.016</td>
<td>0.744 ± 0.024</td>
<td>0.741 ± 0.028</td>
</tr>
<tr>
<td>TUH Artifact</td>
<td>19</td>
<td>6</td>
<td>Baseline</td>
<td>0.711 ± 0.009</td>
<td>0.243 ± 0.016</td>
<td>0.243 ± 0.016</td>
<td>0.176 ± 0.018</td>
<td>0.263 ± 0.022</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-base</td>
<td>0.618 ± 0.011</td>
<td>0.514 ± 0.028</td>
<td>0.566 ± 0.029</td>
<td>0.517 ± 0.028</td>
<td>0.466 ± 0.028</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-CKV</td>
<td>0.623 ± 0.016</td>
<td>0.481 ± 0.028</td>
<td>0.521 ± 0.034</td>
<td>0.491 ± 0.026</td>
<td>0.452 ± 0.019</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-CQ</td>
<td>0.607 ± 0.009</td>
<td>0.524 ± 0.044</td>
<td>0.538 ± 0.043</td>
<td>0.531 ± 0.046</td>
<td>0.505 ± 0.043</td>
</tr>
<tr>
<td>TUH Seizure</td>
<td>21</td>
<td>5</td>
<td>Baseline</td>
<td>0.950 ± 0.010</td>
<td>0.289 ± 0.040</td>
<td>0.368 ± 0.037</td>
<td>0.297 ± 0.052</td>
<td>0.171 ± 0.046</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-base</td>
<td>0.906 ± 0.021</td>
<td>0.663 ± 0.015</td>
<td>0.818 ± 0.016</td>
<td>0.693 ± 0.034</td>
<td>0.502 ± 0.027</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-CKV</td>
<td>0.912 ± 0.027</td>
<td>0.713 ± 0.030</td>
<td>0.841 ± 0.041</td>
<td>0.716 ± 0.043</td>
<td>0.591 ± 0.046</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-CQ</td>
<td>0.890 ± 0.021</td>
<td>0.770 ± 0.041</td>
<td>0.857 ± 0.028</td>
<td>0.808 ± 0.045</td>
<td>0.704 ± 0.058</td>
</tr>
<tr>
<td>CHB-MIT</td>
<td>17</td>
<td>2</td>
<td>Baseline</td>
<td>0.618 ± 0.009</td>
<td>0.371 ± 0.014</td>
<td>0.396 ± 0.014</td>
<td>0.365 ± 0.012</td>
<td>0.439 ± 0.029</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-base</td>
<td>0.554 ± 0.006</td>
<td>0.504 ± 0.010</td>
<td>0.523 ± 0.012</td>
<td>0.503 ± 0.013</td>
<td>0.487 ± 0.016</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-CKV</td>
<td>0.562 ± 0.009</td>
<td>0.518 ± 0.011</td>
<td>0.543 ± 0.009</td>
<td>0.529 ± 0.007</td>
<td>0.504 ± 0.009</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARM-CQ</td>
<td>0.576 ± 0.009</td>
<td>0.541 ± 0.009</td>
<td>0.560 ± 0.010</td>
<td>0.550 ± 0.010</td>
<td>0.530 ± 0.014</td>
</tr>
</tbody>
</table>
embedding of dimension $d = 32$. CHARM-base contains three convolutional layers with 32 feature maps and a filter size of 8 with a stride of 1. The residual attentive modules are inspired by Transformers \([20]\). Queries, keys and values have a dimension of 32, and the mlp has a single hidden layer of dimension 64. For all models, we apply $L1$ regularization onto $p$ with a weight of $10^{-4}$ to promote sparse reordering matrices.

**Training Details**

We train on 500-sample windows dynamically sampled from an entire EEG sequence. The CHARM is trained jointly with the 1D-CNN to minimize a categorical cross-entropy loss, using ADAM \([31]\) with a learning rate of $10^{-4}$ and a batch size of 64 for 100 epochs. For imbalanced datasets (see Section 7.3.1), we use a weighted cross-entropy loss to minimize error across rare and frequent classes equally.

### 7.3 Experiments

#### 7.3.1 Datasets

Our evaluation focuses on the Temple University Hospital EEG Corpus (TUH) \([197]\) and CHB-MIT dataset \([198]\). The TUH corpus is the most extensive publicly available corpus with over 15000 subjects; it comprises several datasets analyzed and annotated by expert clinicians where the majority of EEG is sampled at 250Hz \([197]\). The CHB-MIT contains intractable seizures collected from 23 pediatric subjects at a sampling rate of 256Hz. Here, we focus on the tasks of recognizing abnormal EEG (TUH Abnormal), detection of artifacts such as eye movement or chewing (TUH artifacts) as well as determining the presence and type of seizures (TUH Seizure, CHB-MIT). Each dataset has a different number of EEG channels, as shown in Table 7.1. We employ a 10-folds stratified cross-validation technique for assessing the model performance. Our evaluation metric is the accuracy averaged over ten folds, weighted for imbalanced datasets (TUH Artifacts, TUH Seizure) to account for minority classes.

#### Table 7.2: Performance when masking half of the brain, along a vertical or horizontal axis.

<table>
<thead>
<tr>
<th>Augmentation</th>
<th>Method</th>
<th>Clean</th>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Group A</td>
<td>Group B</td>
</tr>
<tr>
<td>None</td>
<td>Baseline</td>
<td>0.951</td>
<td>0.611 ± 0.010</td>
<td>0.430 ± 0.043</td>
</tr>
<tr>
<td></td>
<td>CHARM-CKV</td>
<td>0.900</td>
<td>0.739 ± 0.012</td>
<td>0.600 ± 0.023</td>
</tr>
<tr>
<td></td>
<td>CHARM-CQ</td>
<td>0.899</td>
<td>0.839 ± 0.030</td>
<td>0.707 ± 0.056</td>
</tr>
<tr>
<td>CMSAugment</td>
<td>Baseline</td>
<td>0.873</td>
<td>0.831 ± 0.037</td>
<td>0.762 ± 0.021</td>
</tr>
<tr>
<td></td>
<td>CHARM-CKV</td>
<td>0.819</td>
<td>0.810 ± 0.029</td>
<td>0.778 ± 0.034</td>
</tr>
<tr>
<td></td>
<td>CHARM-CQ</td>
<td>0.734</td>
<td>0.710 ± 0.117</td>
<td>0.702 ± 0.200</td>
</tr>
</tbody>
</table>
Table 7.3: Out of domain transfer results on CHB-MIT with a model pretrained on TUH Seizure dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Fixed</th>
<th>Fine-tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (in-domain)</td>
<td>0.891 ± 0.038</td>
<td></td>
</tr>
<tr>
<td>Baseline (transfer)</td>
<td>0.757 ± 0.004</td>
<td>0.963 ± 0.015</td>
</tr>
<tr>
<td>CHARM-CKV</td>
<td>0.805 ± 0.008</td>
<td>0.942 ± 0.020</td>
</tr>
<tr>
<td>CHARM-CQ</td>
<td>0.795 ± 0.006</td>
<td>0.915 ± 0.026</td>
</tr>
</tbody>
</table>

7.3.2 Generalizing to Shuffled and Masked Channels

Table 7.1 compares CHARM to a baseline 1D-CNN when generalizing to noisy conditions. The models are trained on clean (no masking, no shuffling) channels, but the evaluation is done under different forms of noise injection. To this end, the Noisy entries in Table 7.1 indicate performance when the test input channels are shuffled and 0 to $N - 1$ channels are uniformly masked. Similarly, Noisy-$n\%$ represents the results when a fixed ratio of $n\%$ channels are masked at random after shuffling. We first observe that when evaluating on clean inputs, the baseline model performs better. This can be explained by the fact that CHARM sees channels independently and cannot exploit ground-truth channel location, which is useful when training and evaluating on identical channels. On the other hand, CHARM-based remapping techniques perform significantly better in handling permuted and masked channels across all four datasets. Even when 50\% to 75\% channels are missing, our proposed approach maintains high accuracy. In particular, on TUH Seizure/Noisy-75\% CHARM-CQ attains 0.704 accuracy, against 0.171 for the baseline.

7.3.3 Performance in Structured Masking Conditions

Table 7.2 reports the performance on TUH Seizure when only a subset of the channels from a specific half of the brain is active, a more tangible setting than random masking. We split the electrodes along a vertical or horizontal axis and evaluate on each half separately (Group $A$ and Group $B$). We train the models either on clean inputs or with CMSAugment, and do not report results for CHARM-base since it performed worse than alternatives in previous experiments. When training without augmentation, CHARM significantly outperforms the baseline in most cases, reaching its best results when combined with CMSAugment. Interestingly, the robustness of the baseline system significantly improves when trained with CMSAugment. This shows that, independently of CHARM, data augmentation is also a promising avenue for robust EEG classification.

7.3.4 Transfer Learning

Until now, we assessed the performance individually for each task, simulating different headsets with random shuffling and masking. Now, we evaluate the proposed methods in handling inconsistent channels in a real cross-dataset transfer learning setting. In [194], authors propose an algorithm to handle transfer between headsets, for a same subject, and using the
common subset of channels shared between headsets. In contrast, CHARM does not require any knowledge about channel placement and exploits all channels of each headset. Moreover, we experiment in a more challenging setting: we transfer trained representations to a new headset, on new subjects. We pretrain the models with clean inputs on the TUH Seizure dataset in a standard way and discard the classification head. We then reuse the other layers for learning either a linear classifier on-top of a fixed network or fine-tune it entirely on the CHB-MIT dataset. Importantly, as the number of channels in CHB-MIT is lower than TUH Seizure, we pad them with zero channels. In Table 7.3, we report the results of the baseline CNN along with channel remapping modules, where the in-domain baseline is the model directly trained on the CHB-MIT dataset, i.e., no transfer is performed. We observe that the representations learned with CHARM transfer better than the baseline when freezing the transferred layers. This shows that our system allows for pretraining and transfer of embeddings across heterogeneous datasets. Interestingly, if we fine-tune the entire network, then the baseline (transfer) is able to relearn low-level representations and improves significantly over the in-domain baseline. To the best of our knowledge, this is the first time that a deep EEG classifier demonstrates out-of-domain transfer to a dataset recorded with a different headset over different subjects.

7.4 Conclusion

We introduce CHARM, a channel remapping model for training a single neural network across heterogeneous EEG data. Our model identifies the location of each channel from their content and remaps them to a canonical ordering by predicting a reordering matrix. This allows further processing by standard classifiers that expect a consistent channel ordering. Our differentiable reordering module leverages attention mechanisms and can be trained on data without information on the channel placement. We complement this model with a new data augmentation technique and demonstrate the efficiency of our approach over three EEG classification tasks, where various types of headsets can result in inconsistent channel orderings and numbers. In particular, we successfully transfer parameters across datasets with different collection protocols. This is an important result since available EEG data are currently fragmented across a wide variety of heterogeneous datasets. We believe that the robustness of our method will pave the way to training a single model across large-scale collections of heterogeneous data. Moreover, our approach is general enough to benefit other domains with varying sensor placements and numbers, including weather modeling, seismic activity monitoring and speech enhancement from microphone arrays.
Chapter 8

Synthesizing and Reconstructing Missing Sensory Modalities

This chapter is based on our paper Synthesizing and Reconstructing Missing Sensory Modalities in Behavioral Context Recognition published in MDPI Sensors 2018 [23].

8.1 Introduction

In last chapter, we explored the problem of handling inconsistent input channels with a learnable remapping module that can be trained in an end-to-end manner while learning desired task. In a real-life setting, a model can also encounter other types of noises, artifacts and in a extreme case required input modalities can be missing entirely. In this chapter, we focus on learning to reconstruct features with a generative model, i.e., an adversarial autoencoder that can impute and synthesize data samples. Our developed technique is general-purpose but we focus on user context recognition as a concrete application domain.

The automatic recognition of human activities along with inferring the associated context is of great importance in several areas, such as intelligent assistive technologies. A minute-to-minute understanding of person’s context can enable immediate support e.g. for elderly monitoring [199], timely interventions to overcome addictions [200], voluntary behavior adjustment for living a healthy lifestyle [201,202], coping with physical inactivity [203] and in industrial environments to improve workforce productivity [204]. The ubiquity of sophisticated sensors integrated into smartphones, smartwatches and fitness trackers provides an excellent opportunity to perform a human activity and behavior analysis as such devices have become an integral part of our daily lives [205]. However, context recognition in a real-life setting is very challenging due to the heterogeneity of sensors, variation in device usage, a different set of routines, and complex behavioral activities [206].
Concretely, to predict people’s behavior in their natural surroundings, a system must be able to learn from multi-modal data sources (such as an accelerometer, audio, and location signals) that are often noisy with missing data. In reality, a system is likely to encounter missing modalities due to various reasons such as a user not wearing a smartwatch, a sensor malfunction or a user not granting permission to access specific data because of privacy concerns. Moreover, due to large individual differences, the training data could be highly imbalanced, with very few (sparse) labels for certain classes. Hence, for a context recognizer to perform well in unconstrained naturalistic conditions; it must handle missing data and class imbalance in a robust manner while learning from multi-modal signals.

There are a variety of techniques available for dealing with missing data [207, 208]. Some naive approaches are, mean substitution or simply discarding instances with missing values. In the former, replacing by average may lead to bias (inconsistency would arise e.g. if the number of missing values for different features are excessively unequal and vary over time) [207]. In the latter, removal leads to a substantial decrease in the number of samples (mostly labeled) that are otherwise available for learning. It can also introduce bias in the model’s output if data are not missing completely at random [208]. Similarly, principal component analysis (PCA) approach could be to utilize through inverse transformation on the reduced dimensions of the original data to restore lost features but the downside is PCA can only capture linear relationships. Another approach might be training a separate model for each modality, where the decision can be made on the basis of majority voting from the available signals. Though in this scheme, the distinct classifiers will fail to learn the correlation that may exist between different sensory modalities. Besides, this approach is inefficient as we have to train and manage a separate classifier for every modality available in the dataset.

An autoencoder is an unsupervised representation learning algorithm that reconstructs its own input usually from a noisy version, which can be seen as a form of regularization to avoid over-fitting [29]. Generally, the input is corrupted by adding a Gaussian noise, applying dropout [209] or randomly masking features as zeros [210]. The model is then trained to learn a latent representation that is robust to corruption and can reproduce clean samples from partially destroyed features. Therefore, denoising autoencoders can be utilized to tackle reconstruction, while learning discriminative representations for an end task e.g. context classification. Furthermore, the adversarial autoencoder (AAE) extends a typical autoencoder to make it a generative model that is able to produce synthetic data points by sampling from an arbitrarily chosen prior distribution. Here, a model is trained with dual losses—reconstruction objective and adversarial criterion to match the hidden code produced via the encoder to some prior distribution [211]. The decoder then acts as a deep generative model that maps the enforced prior distribution to the data distribution. We address the issues of missing data and augmenting synthetic samples with an adversarial autoencoder (AAE) [22].

We present a framework based on AAE to reconstruct features that are likely to go missing all at once (as they are extracted from the same modality) and augment samples to enable synthetic data generation (see Figure 8.1). We demonstrate the representation learning capability of AAE through accurate reconstruction of missing values and supervised multi-label classification of behavioral context. In particular, we show AAE is able to provide a more faithful imputation as compared to techniques such as PCA and show strong predictive performance even in case of several missing modalities. We analyze the performance of the decoder trained
with supervision enabling the model to generate class conditional artificial training data. Further, we show that AAE can be extended with additional layers to perform classification; hence leveraging the complete dataset including labeled and unlabeled instances. The primary contributions of this work are the following: a) demonstration of a method to restore missing sensory modalities using an adversarial autoencoder, b) systematic comparison with other techniques to impute lost data, c) leveraging learned embedding and extending the autoencoder for multi-label context recognition, and d) generating synthetic multi-modal data and its empirical evaluation through visual fidelity of samples and classification performance on a real test set.

![Diagram of the proposed framework for robust context classification with missing sensory modalities.](image)

**Figure 8.1:** Overview of the proposed framework for robust context classification with missing sensory modalities.

### 8.2 Approach

#### 8.2.1 Autoencoder

An autoencoder is an unsupervised representation learning technique in which a deep neural network is trained to reconstruct its own input \( x \) such that the difference between \( x \) and the network’s output \( x' \) is minimized. Briefly, it performs two transformations: encoding \( f_{\theta}(x) : R^n \rightarrow R^d \) and decoding \( g_{\theta}(z) : R^d \rightarrow R^n \) through deterministic mapping functions, namely, *encoder* and *decoder*. An encoder transforms input vector \( x \) to a latent code \( z \), where, a decoder maps the latent representation \( z \) to produce an approximation of \( x \). For a single layer neural network these functions can be written as:

\[
f_{\theta}(x) : z = \sigma(W_e x + b_e) \quad (8.1)
\]

\[
g_{\theta'}(z) : x' = \sigma(W_d z + b_d) \quad (8.2)
\]

parameterized by \( \theta = \{W_e, b_e\} \) and \( \theta' = \{W_d, b_d\} \), where \( \sigma \) is a non-linear activation function (e.g. rectified linear unit), \( W \) represents a weight coefficient matrix and \( b \) is a bias vector. The model weights are sometimes tied for regularization such that \( W_d = W_e^T \).
Learning an autoencoder is an effective approach to perform dimensionality reduction and can be thought of as a strict generalization of PCA. Specifically, a 1-layer encoder with linear activation and mean squared error (MSE) loss (see Equation 8.3) should be able to learn PCA transformation \[ \mathcal{L}_{MSE}(X, X') = \|X - X'\|^2 \] (8.3)

The classical autoencoder can be extended in several ways (see for a review [29]). For handling missing input data, a compelling strategy is to train an autoencoder with artificially corrupted input \( \tilde{x} \), which acts as an implicit regularization. Usually, the considered corruption includes isotropic Gaussian noise, salt and pepper noise and masking (setting randomly chosen features to zero) [210]. In this case, a network learns to reconstruct a noise-free version \( x' \) from \( \tilde{x} \), hence called a denoising autoencoder (DAE). Formally, the DAE is trained with stochastic gradient descent to optimize the following objective function:

\[
J_{DAE} = \min_\theta \mathbb{E}_X [\mathcal{L}(x, g_\theta(f_\theta(\tilde{x})))]
\] (8.4)

where \( \mathcal{L} \) represents a loss function like squared error or binary cross entropy.

### 8.2.2 Adversarial Autoencoder

The adversarial autoencoder (AAE) [22] combines adversarial learning [211] with classical autoencoders so it can be used for both learning data embedding and generating synthetic samples. The generative adversarial network (GAN) introduced a novel framework for developing generative models by simultaneously training two networks: a) the generator \( G \), it learns the training instances’ distribution to produce new samples emulating the original samples, and b) the discriminator network \( D \), which differentiates between original and generated samples. Hence, this formulation can be seen as a minimax game between \( G \) and \( D \) as shown in Equation 8.5, where \( z \) represents a randomly sampled vector from a certain distribution \( p(z) \) (e.g. Gaussian), and \( x \) is a sample from the empirical data distribution \( p_{data}(x) \) i.e. training data.

\[
\min_G \max_D \mathbb{E}_{x \sim p_{data}} [\log D(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]
\] (8.5)

In AAE, an additional discriminator network is added to an existing autoencoder architecture to force the encoder output \( q(z|x) \) to match a specific target distribution \( p(z) \) as depicted in Figure 8.2; hence enabling the decoder to act as a generative model. Its training procedure consists of three sequential steps:

1. The encoder and decoder networks are trained simultaneously to minimize the reconstruction objective (see Equation 8.6). Additionally, the class label information with latent code \( z \) can also be provided to the decoder as supervision. Thus, the decoder then uses both \( z \) and label information \( y \) to reconstruct the input. In addition, conditioning over \( y \) enables the decoder to produce class conditional samples.

\[
J_{AE} = \min_\theta \mathbb{E}_X [\mathcal{L}(x, g_\theta(f_\theta(x)))]
\] (8.6)
The discriminator network is then trained to distinguish between true samples from a prior distribution and fake data points \( z \) generated by an encoder.

Subsequently, the encoder, whose goal is to deceive the discriminator by minimizing a separate loss function, is updated.

### 8.2.3 Context Classification

The context recognition under consideration is a multi-label classification problem, where a user’s context at any particular time can be described by a combination of various labels. For instance, a person might be in a meeting, indoor, and with a phone on a table. Formally, it can be defined as follows: \( X \in \mathbb{R}^n \) (i.e. a design matrix) is a set of \( m \) instances each being \( n \)-dimensional feature vector having a set of labels \( L \). Every instance vector \( x \in X \) has a corresponding subset of \( L \) labels, also called relevant labels; other labels might be missing or can be considered irrelevant 
[213, 214]. The goal of the learner is to find a mapping function \( f_c : \mathbb{R}^n \rightarrow \{0, 1\}^L \) that assigns labels to an instance. Alternatively, the model predicts a one-hot encoded vector \( y \in \{0, 1\}^L \), where, \( y_i = 1 \) (i.e. each element in \( y \)) indicates the label is suitable and \( y_i = 0 \) represents inapplicability.

The feed-forward neural network can be directly used for multi-label classification with *sigmoid* activation function in the last layer and binary cross-entropy loss (see Equation 8.7); as it is assumed that each label has an equal probability of being selected independently of others. Thus, the binary predictions are acquired by thresholding the continuous output at
As mentioned earlier that in real-world datasets the available contextual labels for each instance could be very sparse (i.e. few $y_i = 1$). It may happen as, during data collection phase, a user might quickly select a few relevant labels and overlook or intentionally not provide other related labels about the context. In such a setting, just considering an absence of labels as irrelevant may introduce bias in the model, and simply discarding the instance without complete label information limits the opportunity to learn from the available states. Moreover, the positive labels could be very few with a large number of negatives, resulting in an imbalanced dataset. To tackle these issues, we employ a similar instance weighting strategy to [21] while learning a multi-label classifier. In this situation the objective function becomes:

$$J_C = \frac{1}{NC} \sum_{i=1}^{N} \sum_{c=1}^{C} (\Psi_{i,c} \cdot \mathcal{L}_{CE}(\hat{y}_{i,c}, y_{i,c}))$$

(8.8)

where $\mathcal{L}_{ce}$ is the binary cross-entropy loss, and $\Psi$ is an instance-weighting matrix of size $N \times C$ (i.e. number of training examples and total labels, respectively). The instance weights in $\Psi$ are assigned by inverse class frequency. The entries for the missing labels are set to zero, to impose no contribution in the overall cost from such examples.

8.2.4 Model Architecture and Training

The multi-modal AAE is developed to alleviate two problems: a) the likely issue of losing features of the same modality all at once, and b) synthesizing new labeled samples to increase training dataset size, data augmentation might be helpful to resolve imbalance (in addition to instance weighting), facilitate better understanding of the modeling process, and enable data sharing when original dataset cannot be distributed directly, e.g. due to privacy concerns.

We start the model training process by normalizing continuous features in the range $[0, 1]$ with summary statistics calculated from the training set. Next, all the missing features are filled-in with a particular value i.e. $-1$. It is essential to represent missing data with a distinct value that could not occur in the original. After this minimal pre-processing, a model is trained to reconstruct and synthesize from the clean samples (with all the features available) to provide noise-free ground truth $X$. During reconstruction training, each feature vector $x \in X$ is corrupted with a structured noise to get a corrupted version $\tilde{x}$ as 1) masked noise is added to randomly selected $5\%$ of the features, 2) all the features from three or more randomly chosen modalities are set to $1$, hence emulating missing data, and 3) dropout is applied. The goal of the autoencoder is then to reproduce clean feature vector $x$ from a noisy version $\tilde{x}$ or in other words to predict reasonably close values of the missing features from the available ones. For example, the model may choose an accelerometer signal from the phone to interpolate smartwatch’s accelerometer features or phone states and accelerometer to approximate location features. Furthermore, for synthesizing novel (class conditional) samples, an independent supervised AAE model is trained without introducing any noise in the input and with a slightly different architecture.
After training the AAE model with clean examples for which all sensory modalities are available, it can be extended for multi-label classification. In this situation, either a separate network is learned or additional layers are connected to encoder network to classify a user's behavioral context. For latter, the error is backpropagated through the full network; including encoder and classification layers. Moreover, during the classifier training phase, we keep adding noise in the input as mentioned earlier. To leverage the entire dataset for classification, the noisy features are first reconstructed with the learned autoencoder model and combined with the cleaned data. The class weights are calculated from the combined training set (see Section 8.2.3), where zero weight is assigned to missing labels. Thus, this formulation allows us to learn from any combination of noisy, clean, labeled and unlabeled data.

We employ binary cross-entropy (see Equation 8.7) for reconstruction loss rather than MSE as it led to consistently better results in earlier exploration. Since cross-entropy deals with binary values, all the features are first normalized to lie between zero and one as mentioned earlier. We train the reconstruction network in an unsupervised manner, while the synthesizing model is provided with supervision through the decoder network as one-hot encoded vector $y$ of class labels. The missing labels $y$ are simply represented with zeros instead of $-1$ as we wanted to utilize both labeled and unlabeled instances. The supervision of decoder network also allows the model to better shape the distribution of the hidden code by disentangling label information from compressed representation $z$.

To assess the robustness of our approach for filling-in lost sensor features, we compared it with PCA reconstruction by applying inverse transformation to the reduced 75-dimensional principle components vector. In addition, we evaluated multi-label classification performance by utilizing the learned embedding, and training an extended network on top of an encoder and comparing them with four different ways of dealing with the missing data: mean substitution, filling it with a median, replacing missing values with $-1$, and using a dimensionality reduction method i.e. PCA. To facilitate fair comparison, we limit the reduction of original 166 features to 75-dimensional feature vector, it allows PCA to capture 98% of the variance.

We also experimented with a standard DAE model but found it to perform similarly to AEE for feature reconstruction.

The visual fidelity and the supervised classification task are used to examine the quality of the synthetic samples produced by the (decoder) generative model. We train a context classification model on synthetic data and evaluate its performance on the held-out real test set and vice-versa. Because the decoder is trained with supervision it enables us to generate class conditional samples. For generating labeled data, we use labels from the (real) training set and feed it together with the Gaussian noise into the decoder. Another strategy for data augmentation could be to first sample class labels and then use those for producing synthetic features. However, as we are dealing with multi-label classification, where labels jointly explain the user's context, arbitrarily sampling them is not feasible as it may lead to inconsistent behaviors and activities (such as, sleeping during running). Therefore, we straightforwardly utilize the clean training set labels to sample synthetic data points.
Our approach is implemented in Tensorflow [218]. We initialized weights with Xavier [217] technique and biases with zeros. We use Adam optimizer [31] with fixed but different learning rates for reconstruction and synthesizing models. For the former, the learning rates of 0.0003, 0.0005 and 0.0005 are used for adversarial and reconstruction and classification losses, respectively. While in the latter, 0.001, 0.001 and 0.0005 are used for reconstruction, adversarial and classification losses, respectively. We employ L2-regularization on encoder’s and classifier’s weights with a rate of 0.00001. The rest of the hyper-parameters are minimally tuned on the (internal) validation set by dividing the training folds data into a ratio of 80 – 20 to discover an architecture that gives optimal performance across users. The suitable configuration of reconstruction network is found to be 3-layers encoder and decoder with 128 hidden units in each layer and dropout [209] with a rate of 0.2 on the input layer. The classification network contains a single hidden layer with 64 units. Similarly, the synthesizing model contains 2 hidden layers with 128 and 10 units and dropout of 0.2 is applied on encoding layer z. However, during sampling from the decoder network, we apply dropout with 0.75. The LeakyReLU activation is used in all the layers except for the classifier trained on synthetic data, where ReLU performed better. Moreover, we also experimented with several batch sizes and found 64 to produce optimal results. We train the models for a maximum of 30 epochs and utilize early-stopping to save the model based on internal validation set performance.

8.3 Experiments

8.3.1 ExtraSensory Dataset

We seek to learn a representation of context and activities by leveraging massive amounts of multi-modal signals collected using smartphones and wearables. While there are a variety of open datasets available on the web, we choose to use ExtraSensory Dataset [206] because it was collected in a real-world environment when participants were busy with their daily routines. It provides a more realistic picture of a person’s life as compared to a scripted lab data collection which constrains users to a few basic activities. A system developed with data collected in lab settings fails to capture intrinsic behaviors in every day in-the-wild conditions. The data collection protocol is described in detail in [206], and we provide a brief summary in this section. The data is collected from sixty users with their personal devices using specifically designed applications for Andriod, iPhone, and Pebble-watch unit. Every minute an app collected 20 seconds of measurements from multiple sensors and asked the user to provide multiple labels that define their environment, behavior, and activities from a selection of 100 contextual labels. In total, the dataset consists of 300,000+ labeled and unlabeled measurements of various heterogeneous sensors. We utilize pre-computed features from six modalities: phone-accelerometer (Acc), phone-gyroscope (Gyro), phone-audio (Aud), phone-location (Loc), phone-state (PS), and watch-accelerometer (WAcc). Among Loc features, we only use quick location features (such as user movement) and discard absolute location as it is
place specific. By adding features from each sensing modality, we end up with 166 features, where we utilize 51 processed labels provided in the original dataset.

This dataset also naturally highlights the inevitable problem of missing data in real-world studies. For instance, the participants turned off the location service to avoid battery drain, did not wear the smartwatch continuously and sensor malfunction or other factors resulted in missing samples. In this case, even though labels and signals from other modalities are available but instances with missing features cannot be directly used to train a classifier or to make a prediction in the production setting. This either requires imputation or leads to the discarding of expensive-to-obtain labeled data. From 300K+ instances in the dataset, approximately half of them have all the features available and the rest even though labeled cannot be utilized due to missing values. Therefore, an efficient technique is required to approximate missing data and prevent valuable information from going to waste during learning a context classifier. Similarly, the data collected in-the-wild often have imperfect and imbalanced classes as some of the labels occur only a few times. It can also be attributed to the difference between participants’ routines or their privacy concerns as some classes are entirely missing from their dataset. Hence, learning from imbalanced classes in a principled way becomes crucial to correctly identify true positives. In summary, the *ExtraSensory Dataset* highlights several challenges for context recognition in real-life conditions, including complex behavioral activities, unrestrained personal device usage, and natural environments with habitual routines.

### 8.3.2 Performance Evaluation

We evaluate reconstruction and classification performance through five-folds cross-validation, where each fold has 48 users for training and 12 users for testing; with the same folds as of [206]. The cross-validation technique is used to show the robustness of our approach when the entire data of users are held-out as test-set during experiments. For hyper-parameters optimization in this setting, we randomly divide a training set into 80% training and 20% internal validation set. The same approach is employed to evaluate the quality of synthetic data points via a supervised classification task. Figure 8.3 depicts the data division for imputation and classification experiments. The entire dataset is first split-up into clean and noisy parts, where clean data is used for training and measuring the performance of restoring missing features as described in Section 8.2.4. The noisy data is then interpolated using a learned model and combined with the clean version to use for context classification task. However, we use only clean data to train and evaluate the synthesizing model, the artificial data generated from the AAE is used to train a classifier and its performance is evaluated on real test (folds) data.

The performance of approximating missing data or input reconstruction is measured with root mean square error (RMSE) same as [215]:

$$\text{RMSE} = \sqrt{\mathbb{E}[(X - \hat{X})^2]} \quad (8.9)$$

The multi-label classification is evaluated through balanced accuracy (BA) derived from sensitivity (or recall) and specificity (or true negative rate) as shown in equations below. BA
is a more robust and fair measure of performance for imbalanced data as it is not sensitive to class skew as opposed to average accuracy, precision and f-score which can over or under emphasize the rare labels [213]. Likewise, it is important to note that, the evaluation metrics are calculated independently for each label of the 51 labels and averaged afterwards.

\[
\text{Sensitivity} = \frac{tp}{tp + fn} \quad (8.10) \\
\text{Specificity} = \frac{tn}{tn + fp} \quad (8.11) \\
\text{Balanced Accuracy} = \frac{(\text{Sensitivity} + \text{Specificity})}{2} \quad (8.12)
\]

### 8.3.3 Results

#### Modality reconstruction

We first seek to validate the capability of the AAE network to restore the missing modalities. It is evaluated in comparison with PCA reconstruction, which is achieved by projecting the original 166 features onto a lower dimensional space, having a feature vector of length 75 and then applying an inverse transformation on it to get the original data space. The PCA is able to capture 98% of the variance in the clean training data and thus to set a reasonably strong baseline. However, the AAE network trained with structured noise significantly outperformed the PCA reconstruction by achieving an average RMSE of 0.227 compared with 0.937 on the clean subset of the test folds. To assess the performance of the reconstruction of all the features of each data source, the entire modality is dropped and restored with both procedures. Table 8.1 provides RMSE averaged across folds and number of features for each modality used from the original dataset. Apart from location features, the AAE network outperforms PCA on the reconstruction of every modality. For gyroscope, we noticed a performance drop on test set of _fold 4_ which can be due to relatively fewer number of instances from the participants in the testing fold. The reason for comparatively lower performance on the phone state can be attributed to these features being binary and cannot be perfectly approximated with continuous functions.

The AAE is able to learn compressed non-linear representations that are sufficient to capture
Table 8.1: RMSE for each modality averaged over 5-folds cross-validation.

<table>
<thead>
<tr>
<th>Modality</th>
<th># of Features</th>
<th>PCA</th>
<th>AAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>26</td>
<td>1.104 ± 0.075</td>
<td>0.104 ± 0.016</td>
</tr>
<tr>
<td>Gyroscope</td>
<td>26</td>
<td>1.423 ± 0.967</td>
<td>0.686 ± 1.291</td>
</tr>
<tr>
<td>Accelerometer</td>
<td>46</td>
<td>1.257 ± 0.007</td>
<td>0.147 ± 0.003</td>
</tr>
<tr>
<td>Location</td>
<td>6</td>
<td>0.009 ± 0.003</td>
<td>0.009 ± 0.003</td>
</tr>
<tr>
<td>Audio</td>
<td>28</td>
<td>1.235 ± 0.015</td>
<td>0.080 ± 0.006</td>
</tr>
<tr>
<td>Phone State</td>
<td>34</td>
<td>0.578 ± 0.000</td>
<td>0.337 ± 0.011</td>
</tr>
</tbody>
</table>

Figure 8.4: Restoration of an (phone) accelerometer feature values with the AAE and PCA. The entire modality is dropped and reconstructed using features from the remaining signals.

the correlation between different features. Hence, it provides a close approximation of the features from the lost modality through leveraging the available signals. Figure 8.4 illustrates this point, where an accelerometer signal (from phone) is dropped (mimicking a missing signal) and all of its 26 features are reconstructed by leveraging the rest of the modalities. The AAE network predicted very realistic values of the missing features that are masked with special value $-1$. On the contrary, the PCA restoration is stuck around values near zero; failing to capture the feature variance. We think, it could be because PCA does a linear transformation, while the features may have an inherent non-linear relationship that can be extracted well using autoencoders. The difference between the considered methods is also apparent in Figure 8.5 for filling-in values of features extracted from an audio signal. Here, PCA fluctuates between zero and one, failing to recover the values, whereas, AAE largely recovers values that are close to the ground truth.

Figure 8.5: Restoration of an audio (MFCC) feature values with AAE and PCA. An entire modality is dropped and reconstructed using features from the remaining signals.

Classification with AAE representations

In order to test the ability of AAE to learn a latent code irrespective of missing modalities, we also performed classification experiments with combined, noisy and clean datasets. The feature vector $x$ is passed into the learned autoencoder to get a compressed representation $z$.
of 128 dimensions. This embedding is used to train a 1-layer neural network and compared with other methods of missing data imputation such as filling with mean, median or $-1$ and a dimensionality reduction technique i.e. PCA. Figure 8.6 provides results on various metrics for cross-validation using considered procedures. We did not find a significant difference between the classifiers trained on embedding and other methods. However, the recall (sensitivity) of AAE is found to be better but somewhat close to the mean imputation. The results obtained here are in line with [215] that used an encoded representation for mood prediction and found no improvement. Similarly, in our case, the reason for unchanged performance could be that a large part of the data is clean and the extracted features are based on extensive domain-knowledge which are highly discriminative. Nevertheless, the latent encoding acquired via AAE can be seen as privacy-preserving representation of otherwise sensitive personal data. Moreover, if an autoencoder is trained with recent advancements made in combining deep models with differential privacy [218], even stronger privacy guarantee can be provided.

![Figure 8.6: Classification results of 5-folds cross-validation with combined clean and reconstructed noisy data. This resembles the situation when all the modalities are available during learning and inference phases. We notice the AAE network performs better than other technique with high recall rate of 0.703. AC, BA, SN, and SP stand for accuracy, balanced accuracy, sensitivity, and specificity, respectively.](image)

**Context recognition with several missing modalities**

For better assessment of AAE capability to handle missing data, we simulated multiple scenarios where several modalities are lost at once. These experiments reasonably mimic a real-world situation for the classifier in which a user may turn-off the location service, forget to wear a smartwatch or may be taking a call (such that the audio modality is missing). Thus, as a baseline, we employ techniques to handle missing data through dimensionality reduction and imputation as described earlier and train a classification model with the same configuration (see Section 8.2.4). The AAE model is extended by adding a classifier network on top of an encoder to directly make predictions for the user context.

We begin by investigating the effect of losing each of the six modalities one by one on the classification performance. Figure 8.7 summarizes the classification results by utilizing different techniques to handle missing features. The classifier learned through extending the
AAE network persistently achieved superior performance compared to the others as can be seen from high BA and true positive rate.

![Figure 8.7: Average evaluation metrics for 51 contextual labels with 5-folds cross-validation. All the features from the corresponding modality are dropped and imputed with all the considered techniques. BA, SN, SP, AC stands for balanced accuracy, sensitivity, specificity, and accuracy respectively.](image)

Next, we experimented with dropping three important signals i.e. Acc, Gyro, and Aud at once. Figure 8.8 shows the averaged results across labels and testing folds, when entire feature vectors of the considered modalities are restored with each method. The simplest technique of filling-in missing data with −1 performed poorly with the lowest recall rate and the same goes for PCA which fails to restore the values. However, mean and median imputation performed moderately better as compared to the these two. The AAE achieved better BA and recall rate of 0.710 and 0.700, respectively.

It is important to note that the data is highly imbalanced with few positive samples. There-
Table 8.2: Classification results for 5-folds cross-validation with different missing modalities that are restored with a specific method. The reported metrics are averaged over 51 labels and BA stands for balanced accuracy.

(a) Missing: Gyro, WAcc, Loc and Aud

<table>
<thead>
<tr>
<th></th>
<th>BA</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAE</td>
<td>0.713 ± 0.008</td>
<td>0.711 ± 0.021</td>
<td>0.716 ± 0.021</td>
<td>0.716 ± 0.024</td>
</tr>
<tr>
<td>PCA</td>
<td>0.526 ± 0.007</td>
<td>0.249 ± 0.040</td>
<td>0.802 ± 0.041</td>
<td>0.825 ± 0.034</td>
</tr>
<tr>
<td>Mean</td>
<td>0.669 ± 0.023</td>
<td>0.348 ± 0.056</td>
<td>0.791 ± 0.025</td>
<td>0.789 ± 0.022</td>
</tr>
<tr>
<td>Median</td>
<td>0.657 ± 0.015</td>
<td>0.502 ± 0.043</td>
<td>0.812 ± 0.022</td>
<td>0.808 ± 0.017</td>
</tr>
<tr>
<td>Fill -1</td>
<td>0.519 ± 0.004</td>
<td>0.175 ± 0.012</td>
<td>0.862 ± 0.004</td>
<td>0.857 ± 0.013</td>
</tr>
</tbody>
</table>

(b) Missing: WAcc, Loc and Aud

<table>
<thead>
<tr>
<th></th>
<th>BA</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAE</td>
<td>0.723 ± 0.007</td>
<td>0.729 ± 0.017</td>
<td>0.718 ± 0.013</td>
<td>0.721 ± 0.014</td>
</tr>
<tr>
<td>PCA</td>
<td>0.549 ± 0.02</td>
<td>0.253 ± 0.052</td>
<td>0.842 ± 0.013</td>
<td>0.847 ± 0.019</td>
</tr>
<tr>
<td>Mean</td>
<td>0.682 ± 0.017</td>
<td>0.567 ± 0.04</td>
<td>0.797 ± 0.014</td>
<td>0.799 ± 0.014</td>
</tr>
<tr>
<td>Median</td>
<td>0.678 ± 0.014</td>
<td>0.543 ± 0.028</td>
<td>0.814 ± 0.005</td>
<td>0.806 ± 0.004</td>
</tr>
<tr>
<td>Fill -1</td>
<td>0.547 ± 0.016</td>
<td>0.209 ± 0.087</td>
<td>0.885 ± 0.055</td>
<td>0.836 ± 0.047</td>
</tr>
</tbody>
</table>

(c) Missing: WAcc and Loc

<table>
<thead>
<tr>
<th></th>
<th>BA</th>
<th>Sensitivity</th>
<th>Specificity</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAE</td>
<td>0.722 ± 0.010</td>
<td>0.704 ± 0.029</td>
<td>0.74 ± 0.018</td>
<td>0.742 ± 0.020</td>
</tr>
<tr>
<td>PCA</td>
<td>0.568 ± 0.012</td>
<td>0.300 ± 0.038</td>
<td>0.835 ± 0.016</td>
<td>0.836 ± 0.010</td>
</tr>
<tr>
<td>Mean</td>
<td>0.735 ± 0.011</td>
<td>0.678 ± 0.028</td>
<td>0.793 ± 0.009</td>
<td>0.789 ± 0.008</td>
</tr>
<tr>
<td>Median</td>
<td>0.727 ± 0.012</td>
<td>0.653 ± 0.035</td>
<td>0.801 ± 0.020</td>
<td>0.796 ± 0.020</td>
</tr>
<tr>
<td>Fill -1</td>
<td>0.564 ± 0.026</td>
<td>0.270 ± 0.064</td>
<td>0.859 ± 0.012</td>
<td>0.840 ± 0.008</td>
</tr>
</tbody>
</table>

Therefore, only considering naïve accuracy or true negative rate provides an incomplete picture of the models’ performance. Moreover, to see the fine differences between true positive rates of each technique, Figure 8.9 presents recall rate for all 51 contextual labels. Overall, the AAE network showed superior results across the labels, highlighting its predictive power to very well handle the noisy inputs.

Next, we evaluated a scenario when four modalities, namely, Gyro, WAcc, Loc and Aud are missing together. Specifically, these sensors have high chances of not being available in real-world conditions. Table 8.2a provides results of the experiment, as earlier, the traditional imputation procedures failed to account for the correct identification of true positives. The AAE gracefully handles missing values with BA of 0.713; through learning important characteristics of data distribution on the training set. Likewise, we tested another scenario with only WAcc, Loc and Aud being missing. Table 8.2b shows that AAE maintained BA at 0.723 even when nearly half of the features from three important modalities are missing. We further assess the classifier’s behavior, in a case when a user does not provide access to location service and does not wear a smartwatch, i.e. WAcc and Loc are not available. Table 8.2c provides these results and indicates that mean/median imputations and AAE showed similar performance on BA metric but the AAE has the highest recall rate of 0.704 among the rest. It highlights the consistent predictive power of AAE based classification network for real-world context recognition applications. Moreover, regardless of the number of missing modalities,
the AAE performed superior as compared to other classical ways to handle the lost data.

Generating realistic multi-modal data

One of the key goals of this work is to build a model capable of producing realistic data points and especially features extracted from sensory data. To demonstrate the ability of AAE to generate synthetic data, we evaluate its performance through visual fidelity and classification.

**Figure 8.9:** Recall of 51 contextual labels with 5-folds cross-validation. All the features from Acc, Gyro and Aud modalities are dropped to emulate missing features and imputed with different techniques to train a classifier.
The data generated by the AAE is used to train a classifier, which is then tested on real data instances.

![Figure 8.10: Balanced accuracy of 51 contextual labels for two classifiers trained with real and synthetic samples — evaluation is done on real test data with 5-folds cross-validation.](image)

Similarly, a model is also trained on real data and evaluated on synthetic test data generated by the AAE. This requires the artificial data to have labels, we can provide these labels to the decoder (generator) as supervision, either by sampling them independently or by an additional network (added to an AAE) predict these class labels. Here, we utilized (the former method) using training or test set labels to generate the data, as applicable. This metric of evaluation is also more suitable compared to visual analysis as it determines the ability of synthetic data to be used for real applications. The results of the classification experiments are presented in Table 8.3, which compares the performance achieved for multi-label context recognition with real and artificial data. It can be seen that the model trained on synthetically generated data achieved close results (BA of 0.715 vs. 0.752) as of when a model is learned on an original data. Likewise, the performance is also optimal (BA of 0.700) when synthetic test data generated using test set labels and random noise are assessed on a classifier learned with real samples.

To get a better appreciation of these results, Figure 8.10 provides BA of each class label for models trained on real and synthetic instances — evaluated on a real test set. We notice that, for
some class labels the BA score is equal to or larger than the model learned with real data, such as for classes: Phone in bag, Singing, On beach, and At a restaurant. It indicates that the AAE generates realistic enough samples to train a classifier which then achieves high performance on real test data. Furthermore, we also validate the quality of generated samples by visual inspection. It is helpful as we can see from the generated samples if they have the similar characteristics and dynamics as the one we wish to model. Figures 8.11 illustrates both real and generated examples, the essential thing to notice is that real and synthetic values exhibit similar shift, peaks, and local correlations that are captured well by the AAE. However, binary (discrete) features belonging to phone states such as, is phone connected to Wi-Fi etc. are hard to perfectly reconstruct but they can be easily binarized by thresholding at a particular value.

### 8.4 Related Work

Previous work on behavior context recognition has evaluated fusing single-sensor classifers to handle missing input data, in addition to utilizing different combinations of sensors to develop models for each group. However, these methods do not scale well to many sensors and may fail to learn correlations that exist between different modalities. Furthermore, restoration of missing features with imputation methods remains a non-trivial task as most procedures fail to account for uncertainty in the process. In the past, autoencoders have been successfully used for unsupervised feature learning in several domains thanks to their ability of learning complex, sparse and non-linear features. To put this work into context, we review contemporary approaches to leveraging autoencoders for representation learning and handling missing input data.

Recent methods on ubiquitous activity detection have effectually used the restricted Boltzmann machine, denoising and stacked autoencoders to get compressed feature representations that are useful for activity classification. These methods performed significantly better for learning discriminative latent representations from (partial) noisy input, that is not solely possible with traditional approaches. To the best of our knowledge, no earlier works in activity recognition domain explicitly addresses missing sensors problem except that utilizes dropout for this purpose. Nevertheless, several works in different areas have used autoencoders to interpolate missing data. Thompson et al. used contractive autoencoder for the restoration of missing sensor values and showed it is generally a non-expensive procedure for most data types. Similarly, Netwamondo et al. study the combination of an autoencoder and a genetic algorithm for an
approximation of missing data that have inherent non-linear relationships.
In bioinformatics and healthcare community, denoising autoencoders (DAE) have been used to learn from imperfect data sources. Li et al. [229] used DAE for pretraining and decoding an incomplete electroencephalography to predict motor imagery classes. Likewise, Miotto et al. [230] applied DAE to produce compressed embedding of patients’ characteristics from a very large and noisy set of electronic health records. Their results showed major improvements over alternative feature learning strategies (such as PCA) for clinical prediction tasks. Furthermore, Beaulieu-Jones [228] systematically compared multiple imputation strategies with deep autoencoders on the clinical trial database and showed strong performance gains in disease progression predictions.

Autoencoders are also extensively used in affective computing to advance emotion recognition systems. Martinez et al. [115] applied a convolutional autoencoder on raw physiological signals to extract salient features for affect modeling of game players. In [231], autoencoders are utilized with transfer learning and domain adaption for disentangling emotions in speech. Similarly, Jaques et al. [215] developed a multi-modal autoencoder for filling in missing sensor data for mood prediction in a real-world setting. Furthermore, DAE has been effectively demonstrated for rating prediction tasks in recommendation systems [232].

Generative adversarial network (GAN) [211] as a framework has shown tremendous power to produce realistic looking data samples, particularly images. It is also successfully applied in natural language processing domain to generate sequential data with a focus on discrete tokens [233]. Recently, they are also used in medical domains to produce electronic health records [234] and time-series data from an intensive care unit [131]. Makhzan et. al [22] combined classical autoencoders with GANs through the incorporation of adversarial loss to make them a generative model, hence called adversarial autoencoder (AAE). This makes AAE a suitable candidate for learning to reconstruct and synthesize with a unified model. However, to the best of our knowledge, no previous work has utilized them for synthesizing features extracted from multi-modal time-series, specifically for context and activity recognition. Hence, models capable of successful reconstruction and generation of synthetic samples can help overcome the issues of noisy, imbalanced and access problems (due to sensitive nature) to the data, which ultimately helps downstream models to become more robust.

Our work is broadly inspired by efforts to jointly learn from multi-modal data sources and it is similar to [215] in applied training strategy; though it utilizes an AAE for reconstruction, augmentation, and multi-label behavior context recognition. Besides, as opposed to [213], where a feed-forward classification model is directly trained with dropout [209] to handle missing modalities, here, the model first learn to reconstruct the missing features by employing both dropout and structured noise (see Section 8.2.4). Then, we extend this model with additional layers for multi-label classification through either directly exploiting the encoder or training a network from scratch with learned embedding. In this manner, the AAE based network will not just be able to reconstruct and classify but it can also be used for class conditional data augmentation.
8.5 Conclusion

We proposed a method utilizing an adversarial autoencoder (AAE) for synthesizing and restoring missing sensory data to facilitate user context detection. The signals loss commonly happens during real-world data collection and in realistic situations after model deployment in-the-wild. For example, a user may prefer to not wear a smartwatch, hence, no signals (or features) from a smartwatch that are used during development will be available for inference. Our empirical results demonstrate that the AAE network trained with structured noise can provide a realistic reconstruction of features from the lost modalities as compared to other methods, such as PCA. Similarly, we show the AAE model trained with supervision to a decoder network produce realistic synthetic data, which further can be used for real applications. We have shown the data generation capability of our network through visual fidelity analysis and by comparing classification performance with real data. In the latter, we do training on the artificial data and evaluation of real instances, and training on real and validation on synthetic samples. This methodology allows researchers to develop robust models that are able to learn noise invariant representations and inherently handle several missing modalities. It also enables leveraging artificial data to increase training set size, and data sharing which is occasionally not possible due to the sensitive nature of the personal data.

The presented network has several other advantages, it allows to utilize an entire dataset for learning i.e. any combination of labeled, unlabeled noisy and clean instances. We see a consistent performance of our classifier trained by extending the encoder network, even when several modalities (i.e. more than half of the features) are dropped to emulate missing sensors. Broadly, unlike prior methods for handling missing input data, where a model failed to detect true positive correctly, AAE maintains its ability to recognize user context with high performance. This highlights an important characteristic of the described technique that even if some signals are not available e.g. when users opt-out of location service or do not wear a smartwatch, still their partial data can be used to get accurate predictions. Besides, the model developed with the proposed technique could be a very attractive feature for users concerned about their privacy concerns regarding location data. Likewise, a classifier trained on embedding provides similar performance as the original feature set, which means raw features would not have to be stored and can be shared with other researchers while preserving users’ privacy. The privacy guarantee can be further enhanced by taking advantage of recent advances made in combining deep learning with differential privacy [218].

We notice that labels reported by the users are sparse, resulting in an imbalanced dataset. To deal with this, an instance weighting strategy same as in [213] is applied. Although, we experimented with resolving imbalance through synthetic data only but results were not satisfactory (unless combined with instance weighting); we believe this requires further exploration. Likewise, AAE can be extended to do semi-supervised learning taking advantage of unlabeled examples. It can further help in the collection of a large dataset with a low mental load for the user as it reduces the need for labeling every example. Another area of potential improvement could be an ensemble of multi-layer neural networks efficiently compressed to do real-time detection on an edge device with minimum resource utilization.
Various icons used in the figures are created by Anuar Zhumaev, Tim Madle, Shmidt Sergey, Alina Oleynik, Artdabana@Design and lipi from the Noun Project.
Chapter 9

Model Adaptation and Personalization

This chapter is based on our paper Model Adaptation and Personalization for Physiological Stress Detection published in IEEE DSAA 2018 [24]. It was a joint work with Jan van Erp and Stojan Trajanovski.

9.1 Introduction

In this chapter, we provide a simple yet effective approach to personalizing deep neural networks and adapting them using unlabeled target data of a same task as source domain with a multi-task learning framework. We show efficacy of our approach on physiological stress recognition but we note that the proposed methods are generic and can be used to solve other tasks in a straightforward manner.

We experience numerous stressful situations in our daily lives, such as dealing with annual job evaluation, business failure, or illness. Stress is described as a psychophysiological response to mental, emotional and physical challenges encountered in daily life [235]. Even though the human body is capable of dealing with short-lived day-to-day stressors, the long-term exposure to unremitting stress can have destructive consequences for well-being, productivity, behavior, and self-confidence [236, 237]. Stress can also adversely affect health with implications for progression, recovery, and treatment of nearly every known disease through physiological, behavioral and cognitive changes [235]. It increases the risk of diabetes, metabolic disorders, cardiovascular diseases and (psycho) somatic complaints [238, 239]. Due to these health and performance issues, stress management becomes important. A timely detection of stress can be extremely powerful as it can empower users to take corrective and preventive measures in an informed manner [240].

The autonomic nervous system (ANS) consists of two branches, namely, sympathetic and
parasympathetic nervous system which are both influenced by (amongst others) physiological stress and emotional arousal. The activity of the sympathetic part results in an increase in heart rate, blood pressure, respiration, and blood flow to the muscles. An activity of the parasympathetic division results in an increase in blood flow to the organs and the skin, a decrease in heart rate and respiration, and an increase in heart rate variability. The ANS responds to stress by stimulating specified target organs via efferent neuron tracts, initiated in the locus coeruleus of the brain stem [241] resulting in a release of noradrenaline and norepinephrine. The immediate effect thereof is an increase in sympathetic and a decrease in parasympathetic activity, resulting in a measurable change in physiological parameters, such as an increased heart rate (HR) and skin conductance (SC) level.

Assessing stress levels has a wide area of applications, from increasing resilience of military personnel to enhancing athletes’ performance and improving workforce productivity. Several techniques have been proposed in the past to detect stress in pilots [242], car and truck drivers [69, 109, 243], computer users [244], call center employees [240] and in surgeons [245]. In addition to audio-visual modalities, most approaches use numerous physiological signals, such as respiration rate, electrocardiography (ECG), blood pressure, and electromyography (EMG). The collection of these data in natural conditions is very difficult and usually not consumer friendly enough for practical applications. In contrast, SC and HR can be reliably acquired in a non-invasive and non-obtrusive way from wearable sensors placed on the wrist. Currently, the key challenge is the reliable and personalized classification of stress-states based on these easy to obtain SC and HR signals. In the present work, we focus on personalization and unsupervised model adaptation to improve stress assessment both inside and outside controlled lab environments (domains) using HR and SC signals.

The development of wearable sensors for electrodermal activity and heart rate monitoring has boosted the interest in using these for stress assessment over the last decade. Several recent works have shown their successful application with machine learning algorithms to detect stress in different (mostly controlled) conditions [246, 247, 248]; see [249] for a detailed survey. The aspect that is evident from the overview of earlier work is that current methods do not address issues of end-to-end representation learning, covariate shift, personalization, and domain adaptation. The traditional supervised learning algorithms are not robust to dataset bias [250] and may perform poorly when the data distribution of training instances (of a source domain) differs from test instances (of a target domain). For example, a model trained on a data collected in a simulated (constrained) environment may not be able to perform well in a real-world (unconstrained) setting. Hence, these methods require a collection of ground-truth data (in real-world) for model retraining and are unable to leverage unlabeled data directly to perform cross-domain stress classification. Similarly, physiological signals tend to vary in people and are influenced by age, gender, diet or sleep [127]. Due to this fact, stress responses can differ from person to person. The global (or one-fits-all) models, often do not generalize well to unseen test subjects and hence need extensive fine-tuning.

To address the aforementioned issues, we propose an end-to-end representation learning framework based on a deep reconstruction classification network [251] (DRCN) and multitask learning (MTL). We focus on personalization and domain adaptation together as DRCN can be seen as an extension of MTL. The objective of DRCN is to improve predictive performance on the target domain through joint training on labeled and unlabeled data points. It
performs shared feature extraction through supervised source label predictions and unsupervised target data reconstruction. Specifically, the reconstruction phase enables the network to adapt the label prediction function for the target domain, which is similar to learning an auxiliary task in MTL setting to improve the performance on the actual task \cite{252}. Likewise, model personalization can be achieved with MTL, if subjects are treated as tasks \cite{253}. In this case, the multi-task neural network has hard (or soft) parameter sharing of mutual representations along with distinct layers for each subject (or task) to account for bodily interpersonal differences.

We demonstrate the versatility of the proposed methods via three datasets from a representative application area i.e. stress detection in a driving context. Our approach makes no assumption about the sensor types, sampling frequency, and structure of the physiological time series. It is important to note that these methods are flexible, they can be applied to a variety of neural network architectures and can be used for a variety of different time series classification tasks with minimal changes. Additionally, DRCN and MTL models learned in an end-to-end fashion match or improve results obtained through ad-hoc feature extraction procedures, achieving promising predictive accuracy without any input from domain experts.

The primary contributions of this work are:

- Using multi-modal physiological time-series data from real-world and simulated driving environments to develop a stress recognition model with end-to-end representation learning on the one hand and manual feature engineering on the other.
- Demonstration of an unsupervised model adaptation for cross-domain transfer using deep reconstruction classification networks.
- Presenting a robust approach for personalizing a model with deep multitask neural networks.

9.2 Approach

9.2.1 Problem Definition

The stress detection (classification or recognition) can be framed as a sequence (time-series) classification task which takes physiological signals as input and outputs a label (generally binary) for each sequence. The raw input signals of different modalities are divided into segments of fixed length; with sliding window to avoid semantic segmentation. This process produces \( m \) input-output \( \{(x_i, y_i)\}_{i=1}^{m} \) pairs, where \( y_i \) is taken to be the mode of context window. The \( x_i \) is either used directly for learning representations with deep networks or high-level features are extracted from it manually to learn a classification model.
We formulate model adaptation as a cross-domain and cross-user transfer learning problem. Here, a model trained on a dataset collected in a specific setting or source domain has to be adapted to perform the same task in a different situation or target domain. The key challenges, in this case, are a) unavailability of ground-truth for the target domain, b) expensive process of acquiring a large number of labels, and c) dynamic shift in data distribution. Therefore, target data cannot be directly used for fine-tuning an existing model in a supervised manner. However, the unlabeled target data provide auxiliary training information that can be leveraged to improve model generalization on the target domain than using only source data. This learning setting resembles MTL in the sense that learning an auxiliary task can help improve performance for the actual task using a shared representation.

Our goal is to transfer knowledge from labeled source data $S$ to improve classification performance on unlabeled target data $T$. Let, $X_S$ represent data instances and let $Y_S$ be stress labels for the source. Likewise, $X_T$ denotes data points from the target without any labels, $Y_T$. In domain adaptation case, the marginal probability distribution of input data i.e. $P(X_S)$ and $P(X_T)$ are different but the set of classes are the same $Y_S = Y_T$. We used an extension of deep reconstruction classification network to jointly model distribution of $S$ and $T$ with a combination of supervised and unsupervised objectives. The model is based on temporal convolution and recurrent layers (see Figure 9.1). There are two distinct stages of the source and target feature learning by having a shared encoding representation. The

![Figure 9.1: Unsupervised (cross-domain) model adaptation architecture. The network consists of three main blocks, encoder, decoder and label classifier, where encoder is shared between autoencoder and label classifier. The target data is reconstructed with encoder/decoder part of the network, represented by E and D. Similarly, source labels are predicted with the encoder and classifier, showed by E and L. The model is trained end-to-end with back-propagation using gradient descent (Adam). During optimization, first the weights of classification network (C) are updated followed by weight optimization of autoencoder (R). Concretely, the labeled source data flow through lower part of the model whereas, the unlabeled target data passes through upper part of the network.](image-url)
The encoding phase of the architecture consists of 2 temporal convolution layers each followed by a max pooling operation with a pool size of 5. The convolution layers all have 90 feature maps and a filter length of 10 with rectified linear activation. The decoder architecture is similar except that the output is upsampled at the same rate as the input is downsampled in the encoder. The classification network shares the same encoder but has an additional bidirectional recurrent layer with 80 units. It is followed by a standard sigmoid layer to get a binary output. The Gaussian noise with a standard deviation of 0.1 is added to both source and target instances and l2-regularization is applied on the encoder’s weights. The model is jointly optimized with binary classification ($L_C$) and reconstruction ($L_R$) losses for $S$ and $T$, respectively. Given $m_S$ source labeled instances $\{(x_i, y_i)\}_{i=1}^{m_S}$ and $m_T$ unlabeled target samples $\{(x_i)\}_{i=1}^{m_T}$, the objective functions are then defined as follows:

$$L_C(\hat{y}, y) = -\sum_{i=1}^{m} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$ (9.1)

$$L_{MSE}(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$ (9.2)

$$L_C = L_C(X_S; \{\Theta_E, \Theta_L\}, Y_S)$$ (9.3)

$$L_R = L_{MSE}(X_T; \{\Theta_E, \Theta_D\}, X_T)$$ (9.4)

$$L_{CL} = \alpha L_C + (1 - \alpha) L_R$$ (9.5)

where $\Theta_E$, $\Theta_D$, $\Theta_L$, are an encoder, decoder and label prediction network weights, respectively. Note that $\Theta_E$ is shared between classification network $C$ and autoencoder $R$. Likewise, $0 \leq \alpha \leq 1$ is a trade-off hyper-parameter to control the contribution of classification and reconstruction losses.

### 9.2.3 Personalization

A subject-independent global model for stress detection may perform poorly due to large interpersonal variations in physiological parameters \cite{127} e.g. due to age, gender, sleep, and diet. In order to take these disparities into account, we personalize a model by applying deep multi-task learning (MTL) with the subjects-as-tasks approach \cite{253}. MTL involves finding a unified model for solving more than one task with a shared representation of the tasks. Consequently, a multi-task neural network (MT-NN) consists of common layers across tasks
as well as task-specific layers. Besides, the last layer contains a separate output unit and a loss function for each task. The optimization of loss functions is done at the same time by alternating between different tasks at random.

![Multi-task convolutional neural network architecture](image)

**Figure 9.2:** Multi-task convolutional neural network architecture. The network consists of two temporal convolution and an average pooling layers along with a dropout which acts as the shared feature extractors. The separate convolution and dense layers are used as private layers for each participant with a sigmoid unit in the last. The model input is a 3d tensor of raw physiological signals of fixed length. It is trained end-to-end with back-propagation and gradient descent by alternating between tasks (subjects) at random (see Section 9.3).

We use two model architectures for the MTL setting, one based on the temporal convolutional neural network for end-to-end representation learning (see Figure 9.2) and another feed-forward neural network trained with manually extracted features. In the former, the first three layers act as the shared feature extractors among the tasks (see Figure 9.3). They have 96 features maps with a kernel size of 8 and average pooling of size 5. A separate convolution and fully connected layers are employed as subject-specific layers to learn personalized features. The private convolution layer has the same configuration as shared ones but the dense layer has 512 hidden units with \( \tanh \) activation. In the latter, a fully-connected layer with 200 units is used as a shared layer, whereas a separate hidden layer with 100 units is used as a private layer for each subject. In both cases, the last part contains a sigmoidal layer with standard binary cross-entropy loss function for each user. We use rectified linear activation in every layer (unless mentioned otherwise) and apply \( l_2 \)-regularization and dropout with a rate of 0.0001 and 0.2, respectively, to avoid over-fitting.

This model architecture will be able to take interpersonal variations in physiological signals into account through person-specific layers, rather than; having a mutual global representation. Likewise, we perceive personalization for new unseen user straight-forward through adding randomly initialized layers to an existing model. In this case, our architecture can be seen as an instantiation of Progressive Neural Networks [254]. The newly-added layers can be attached to existing shared layers; while dropping or chaining the private layers for knowledge transfer. The user-specific layers can then be optimized while keeping weights of shared layers frozen or tuning them separately with very small learning rate. This training strategy provides an additional benefit as the data from earlier users/tasks are not necessarily required to train a personalized model from scratch.
9.3 Dataset and Pre-Processing

We use skin conductance and heart rate signals from real-world and simulator driving datasets. The details are discussed below:

MIT Driver Stress (M)

The MIT Driver Stress dataset [255] consists of physiological signals recorded from 17 drives in a real-life experiment; when participants drove in a city, on a highway and rested in a garage. The collected signals comprise of EMG, ECG, galvanic skin response (GSR) from hand and foot, HR derived from ECG and respiration rate. The signals provided in the dataset are all down-sampled to 15.5 Hz. We used the ‘marker’ signal (a button press) to derive the ground-truth annotation for binary stress levels. The peaks are detected in the signal to capture the button push event; indicating a new trial of the experiment is commencing, e.g. the start or end of a rest period. The data points before and after the first and last markers (peaks) are removed as they correspond to the time when subjects were equipped with sensors. Likewise, 4 minutes of data after resting and before the beginning of the post driving baseline are removed. These steps are taken to avoid feeding signals with ambiguous labels, as it is hard to determine if subjects are stressed or recuperated. The artifacts are removed from HR and GSR signals following [256] as values fluctuated to unreasonably high and low levels. Likewise, ECG, GSR from foot and respiration rate are not used as collecting them in real-world situations is very problematic. Lastly, the following 10 drives’ dataset having valid HR, GSR from hand, and marker signals are used for further experimentation: 04, 05, 06, 07, 08, 09, 10, 11, 12 and 16.

Distracted Driving (D)

The multi-modal Distracted Driving dataset [257] is acquired on a simulator in a controlled environment. The dataset includes data from 68 volunteers (35 male/33 female) that drove the
same highway under four different conditions: a) no interruption, b) cognitive distraction, c) emotional distraction, and d) sensorimotor distraction. In addition to the driving indicators (such as speed, brake force, and steering) and eye tracking; several physiological signals were recorded. These include palm electrodermal activity (EDA), HR, breathing rate, and perinasal perspiratory signal. We normalize EDA (dividing by 1000) as a pre-processing step to ensure the same range of variability compared to other data sources. In this research as our focus is on detecting cognitive load stressors, we used only the EDA and HR data (provided with a sampling rate of 1 Hz) from drive under normal and cognitive mental load. During a cognitive load drive, the stressors were induced by mathematical and analytical questions posed verbally by the experimenter. We used 40 participants for our analysis and dropped the rest due to corrupted or missing signals either during a normal or a stressful drive.

**Cognitive Load Driving (C)**

We collected heart rate and skin conductance (SC) data from 19 professional truck drivers using wrist-worn devices. The SC signal was recorded at a frequency of 10 Hz and HR was derived from Photoplethysmogram sensor data with a frequency of 1 Hz; it was upsampled to match the frequency of SC. The experiment was realized with a driving simulation software and participants received standardized instructions from an audiotape. The study consisted of three major steps: 1) baseline driving, 2) moderate stress activity, and 3) high-stress task. The high stress was induced by means of a secondary arithmetic subtraction task. It is a component of widely used Trier Social Stress Test [258], where a user has to perform a serial subtraction verbally in a loud manner and has to start over from the last correct answer; if a mistake is made. Since we are interested in recognition of baseline and high stress, data points of moderate stress activity are dropped. Also, two subjects are dropped due to having bad quality signals.

**Segmentation and Features**

To prepare the data for model input, we used a sliding window approach as mentioned earlier to extract fixed-length sequences from each participant’s physiological signals. A window length of 300 samples with a fixed step size of 50 samples is used for each dataset. In the case of end-to-end representation learning, raw physiological signals are used. For traditional learning algorithms, features are extracted manually from HR and SC which is discussed below. It is important to note that raw segments and features were computed from pre-processed signals, standardized with mean normalization by baseline to compensate for individuals having different resting heart rates.

**Heart Rate** Heart rate is the number of complete cardiac cycles for instance measured as the R-R interval in an electrocardiogram. It reflects the heart activity, including autonomic nervous system activity when it accommodates the body’s demands depending on the received stimuli [109]. We obtained the following seven features from heart rate: mean, standard deviation, min, max, range, root mean square of successive differences, and standard deviation of successive differences.
Skin Conductance The skin conductance (also known as galvanic skin response or electrodermal activity) describes the autonomic variations in electrical properties of the skin or equivalently, the number of active sweat glands. It is widely used as a sensitive index of emotional processing, sympathetic activity and is a relevant indicator of the stress level of a person [259, 260]. From this signal, the following nine features are extracted: mean, standard deviation, min, max, range, number of peaks, amplitude, skewness, and kurtosis.

Table 9.1: Test set Kappa and AUROC score for unsupervised (cross-domain) model adaptation

<table>
<thead>
<tr>
<th>Methods</th>
<th>C → D</th>
<th>C → M</th>
<th>D → C</th>
<th>D → M</th>
<th>M → C</th>
<th>M → D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Only - NN</td>
<td>0.040</td>
<td>0.640</td>
<td>0.371</td>
<td>0.604</td>
<td>0.470</td>
<td>0.148</td>
</tr>
<tr>
<td>Source Only - CNN</td>
<td>0.246</td>
<td>0.648</td>
<td>0.389</td>
<td>0.566</td>
<td>0.594</td>
<td>0.401</td>
</tr>
<tr>
<td>DRCN - NN</td>
<td>0.110</td>
<td>0.215</td>
<td>0.527</td>
<td>0.192</td>
<td>0.491</td>
<td>0.186</td>
</tr>
<tr>
<td>DRCN - CNN</td>
<td>0.541</td>
<td>0.656</td>
<td>0.747</td>
<td>0.701</td>
<td>0.774</td>
<td>0.432</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>C → D</th>
<th>C → M</th>
<th>D → C</th>
<th>D → M</th>
<th>M → C</th>
<th>M → D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Only - NN</td>
<td>0.521</td>
<td>0.826</td>
<td>0.760</td>
<td>0.781</td>
<td>0.780</td>
<td>0.575</td>
</tr>
<tr>
<td>Source Only - CNN</td>
<td>0.619</td>
<td>0.827</td>
<td>0.765</td>
<td>0.755</td>
<td>0.827</td>
<td>0.701</td>
</tr>
<tr>
<td>DRCN - NN</td>
<td>0.165</td>
<td>0.628</td>
<td>0.807</td>
<td>0.610</td>
<td>0.798</td>
<td>0.598</td>
</tr>
<tr>
<td>DRCN - CNN</td>
<td>0.772</td>
<td>0.830</td>
<td>0.844</td>
<td>0.831</td>
<td>0.867</td>
<td>0.713</td>
</tr>
</tbody>
</table>

9.4 Experiments

Our experiments were conducted using physiological signals from three datasets described in section 9.3: MIT Driver Stress (M) [255], Distracted Driving (D) [257] and Cognitive Load Driving (C). The data of every subject in each dataset is randomly divided into training, validation and test sets of size 70%, 10%, and 20%, respectively. For each experiment, the networks are trained from scratch, initializing the weights with the Xavier technique [217]. We use the Adam [31] optimizer with the default parameters but used the validation set to find optimal learning rate and trade-off parameters (α). The optimal values of α are found to be between [0.2-0.7]. Finally, we employ validation based early stopping during the optimization process to further avoid over-fitting and improve the stress recognition rate.

Table 9.2: Average test set (20%) results of drives in MIT Driver Stress dataset

<table>
<thead>
<tr>
<th>Model</th>
<th>AUROC</th>
<th>Kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>0.821 ± 0.074</td>
<td>0.650 ± 0.143</td>
</tr>
<tr>
<td>SVM (L)</td>
<td>0.832 ± 0.076</td>
<td>0.675 ± 0.146</td>
</tr>
<tr>
<td>SVM (RBF)</td>
<td>0.894 ± 0.035</td>
<td>0.808 ± 0.062</td>
</tr>
<tr>
<td>ST-NN</td>
<td>0.852 ± 0.116</td>
<td>0.707 ± 0.241</td>
</tr>
<tr>
<td>MT-NN</td>
<td>0.905 ± 0.056</td>
<td>0.831 ± 0.098</td>
</tr>
<tr>
<td>MT-CNN</td>
<td>0.918 ± 0.058</td>
<td>0.841 ± 0.110</td>
</tr>
</tbody>
</table>

Table 9.3: Average test set (20%) results of participants of Cognitive Load Driving dataset

<table>
<thead>
<tr>
<th>Model</th>
<th>AUROC</th>
<th>Kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>0.880 ± 0.161</td>
<td>0.745 ± 0.314</td>
</tr>
<tr>
<td>SVM (L)</td>
<td>0.876 ± 0.141</td>
<td>0.740 ± 0.264</td>
</tr>
<tr>
<td>SVM (RBF)</td>
<td>0.923 ± 0.104</td>
<td>0.853 ± 0.252</td>
</tr>
<tr>
<td>ST-NN</td>
<td>0.935 ± 0.072</td>
<td>0.855 ± 0.142</td>
</tr>
<tr>
<td>MT-NN</td>
<td>0.960 ± 0.056</td>
<td>0.911 ± 0.114</td>
</tr>
<tr>
<td>MT-CNN</td>
<td>0.956 ± 0.080</td>
<td>0.918 ± 0.147</td>
</tr>
</tbody>
</table>
We evaluated DRCN for model adaptation on six combinations (source \( S \rightarrow \text{target} \ T \)) of the above mentioned datasets: \( C \rightarrow M \), \( C \rightarrow D \), \( D \rightarrow C \), \( D \rightarrow M \), \( M \rightarrow D \) and \( M \rightarrow C \) and report kappa and area under the receiver operating curve (AUROC) on the held-out test set. For a baseline, we used a CNN model trained only on source data with architecture similar to the encoder part of the model as discussed in section 9.2.2. Furthermore, we also experimented with feed-forward networks trained with manually extracted features for both DRCN and source-only settings. The feed-forward models consist of 3 hidden layers with 128, 64 and 32 units with \( \tanh \) activation, where the decoder network has a similar configuration but layers in opposite order to reconstruct the original input vector of 16 dimensions. The results are summarized in Table 9.1. The DRCN model trained end-to-end demonstrates a strong performance boost for the unsupervised cross-domain transfer learning problem. It achieves kappa of above 0.7 from the simulator to on-road and vice-versa from source-only baseline kappa of 0.6. It is important to note that, we used a fixed architecture for all six combinations of model adaptation tasks to show predictive performance increase via joint training on source and target. We believe further improvement can be achieved if architectural components (e.g. number of kernels, kernel size, activation) are optimized for each adaptation task separately. Likewise, convolutional models trained end-to-end outperformed those with an ad-hoc feature extraction procedure. This can be due to CNN’s capacity and ability to automatically learn general to specific features from source and target domains together. Although, when the target domain is Distracted Driving, the domain adaptation performance is comparatively low. This could be due to the relatively small size of this dataset and the recognition rate can be improved with a larger dataset.

### Table 9.4: Average test set (20%) results of users in Distracted Driving dataset

<table>
<thead>
<tr>
<th>Model</th>
<th>AUROC ±</th>
<th>Kappa ±</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>0.734 ± 0.215</td>
<td>0.473 ± 0.431</td>
</tr>
<tr>
<td>SVM (L)</td>
<td>0.735 ± 0.217</td>
<td>0.472 ± 0.429</td>
</tr>
<tr>
<td>SVM (RBF)</td>
<td>0.882 ± 0.152</td>
<td>0.760 ± 0.909</td>
</tr>
<tr>
<td>ST-NN</td>
<td>0.860 ± 0.166</td>
<td>0.715 ± 0.314</td>
</tr>
<tr>
<td>MT-NN</td>
<td>0.908 ± 0.140</td>
<td>0.814 ± 0.282</td>
</tr>
<tr>
<td>MT-CNN</td>
<td>0.871 ± 0.127</td>
<td>0.738 ± 0.257</td>
</tr>
</tbody>
</table>

In our attempt to personalize the model, we first evaluated two standard classifiers as a baseline: logistic regression (LR) and support vector machine with linear (L) and radial basis function (RBF) kernels. In addition, we also trained two layers (subject independent) feed-forward neural network with 100 hidden units and rectified linear activation in each layer. The data of each subject is randomly divided into (80/20) train and test sets. The cross-validation is performed on the training set for hyper-parameter optimization and evaluation metrics are averaged across participants on the test set. The stress recognition performance of these models is summarized in Table 9.2, 9.3 and 9.4 for real-world and simulator drivings, respectively. In MIT Driver Stress (on-road) dataset, SVM (RBF) set a strong baseline by achieving the highest results among other single-task models including ST-NN. The proposed MT-CNN model greatly improved upon that by achieving kappa of 0.84 and AUROC score of 0.91. It can be seen as an overall improvement across drivers due to subject-specific layers. Likewise, the MT-NN model which is trained with manually extracted features achieved similar results. Nevertheless, we advise caution in the interpretation of MIT Driver Stress dataset’s result as no actual ground truth annotations or subjective self-reports are publicly available. The labels
were acquired by means of a ‘marker’ signal, representing the start of next study trial (i.e. from resting to driving in a city) and assuming that driving, in general, is a stressful task.

For simulator driving datasets, the standard (one-fits-all) classifiers do not generalize as can be seen from the high standard deviation values of evaluation metrics in Table 9.3 and 9.4. The difference is particularly high for the Distracted Driving dataset, where a number of participants were comparatively large and more diverse belonging to different gender and age groups. The MT-NN notably improved the recognition rate across subjects and resulted in a better model by achieving kappa of 0.91 and 0.81 on both simulator datasets. Similarly, MT-CNN performed well apart from on Distracted Driving dataset which can be attributed to its small size as deep models require large datasets for representation learning. However, these results show that multi-task learning with reliable quality signals can be used to develop a personalized model as it generalizes well across various users and different environments i.e. real-world and simulators.

9.5 Conclusion

In this work, we proposed a solution for unsupervised cross-domain adaptation and personalization of physiological stress recognition models with deep multi-task learning (MTL). The traditional learning approaches used for stress detection mostly (see [249] for a review) rely on sensor data (such as EMG, respiration rate, facial expressions and pupil dilation) that are very hard to acquire in a real-life situation to develop practical applications. Likewise, they do not explicitly address issues of end-to-end representation learning, covariate shift, and domain adaptation. Therefore, these methods may perform poorly when data distribution (of a source domain) training instances differs from test instances (of a target domain). Similarly, global subject-independent models do not generalize well to new test subjects because of large interpersonal variations in physiological parameters of individuals which can be due to age, gender, sleep, and diet. We used skin conductance and heart rate from real-world and simulator driving tasks to show: a) how models can be adapted to improve predictive performance on target domain in an unsupervised manner with deep reconstruction and classification networks (DRCN) and b) how to utilize multi-task learning (with subjects-as-tasks) to get personalized stress models. In our experiments, we found that the convolutional neural network based DRCN model outperforms the models trained only on source data and feed-forward networks utilizing manually extracted features. Likewise, in model personalization experiments, the MTL networks either trained end-to-end or with feature extraction procedures significantly improve the recognition rate across all datasets as compared to single-task models. We believe, if a wearable device provides reliable and high-quality signals, a real-time stress detection application can be developed to improve safety and well-being. In addition to stress classification in a driving environment, a future study may involve applying and investigating the performance of these methods in a daily-life context by comparing the model's outputs against subjective self-reports.
Chapter 10

Unified Model for Cross-Domain Sensing Tasks

This chapter is based on our paper Learning Cross-Domain Sensing Tasks with a Unified Self-Supervised Model, which is under review. It was a joint work with Shkurta Gashi, Shohreh Deldari, Flora D. Salim, Daniel V. Smith, and Silvia Santini.

10.1 Introduction

In the previous chapters, we developed a broad spectrum of self-supervised tasks for learning representations from many input modalities and datasets individually, including techniques to handle various input artifacts. In this chapter, we address the subject of building a unified neural network-based model for solving multiple sensing tasks simultaneously with a single model. In particular, we show that a unified model can be trained in a supervised and self-supervised manner and is highly competitive with single-task counterparts in a variety of settings. Single-task deep learning models have been extensively investigated to recognize behavior from sensor data \[3, 108\]. While these methods achieve acceptable performance, they ignore the knowledge and structure shared between the tasks and might not generalize well \[32\]. They also require redundant effort in designing and training a separate model for each task \[261\]. These methods can be prone to overfitting in the case of few labeled data as they require curation of massive per-task dataset. Lastly, the training of multiple similar deep models require extensive computational resources in terms of exploring architectural and hyper-parameter space.

Several researchers have shown the effectiveness of multi-task learning for solving a variety of tasks using single sensor data \[14, 262, 263\]. Existing approaches, however, focus on developing models for learning to solve similar problems using data collected from the same domain and input modality. This prevents the model from considering the correlations be-
between cross-domain data of different modalities simultaneously. Furthermore, it may also increase the deployment cost, particularly on resource-constrained devices, as each task will have one model. A multi-task model with a shared backbone and multiple heads, one for each task vastly addresses the aforementioned issues while allowing tasks to learn general representations from each other.

Although multi-modal and multi-task learning has been explored to solve various tasks with sensory data, existing methods address the problem in a purely supervised manner by heavily depending on the availability of labeled data. The collection of sensory data for several applications is challenging, costly and in some cases requires domain expertise. Therefore, a large amount of unlabeled data can not be utilized for learning [15]. Self-supervised learning (SSL) aims to solve this issue and enables the learning of useful representations from unlabeled data through tasking the network to solve an auxiliary objective for which supervision can be acquired from the input itself. Given this, we aim to answer the following question:

*Can we develop a unified deep model with self-supervision for multiple sensing tasks using multi-modal data to achieve similar performance as single-task models?*

To answer the question mentioned above, we propose *UniModel*, a multi-task deep neural network to concurrently learn to solve multiple tasks with multi-modal data collected using various sensors. To evaluate *UniModel*, we train it on the following tasks: heart rate estimation, sleep stage scoring, activity recognition and abnormal electroencephalogram (EEG) detection. Our technique exploits the underlying relations and shared features between modalities and tasks to learn a specific task better than learning them individually. We propose to leverage a self-supervision to overcome the obstacle of label scarcity in mobile and wearable sensor data as SSL techniques do not require annotated data. In particular, our model learns useful representations for downstream tasks using unlabeled data collected from different sensors. Recently, several authors have effectively utilized SSL to learn features from audio and visual data [264, 265], including wearable sensor data [14, 15, 266]. In contrast to these existing approaches, we explore a multi-task formulation of contrastive predictive coding [42] to learn general-purpose representations in a label-free manner that generalizes better on downstream tasks. Additionally, we investigate the effectiveness of SSL in low-labeled data regimes to leverage a pretrained network for improved generalization. Further, to evaluate the generalizability of *UniModel* on out-of-domain data, we investigate the applicability of our pretrained self-supervised representations for transfer learning. It has been consistently shown that the transfer from the existing models significantly reduces the training time and effort needed to converge to an optimal solution on a future task through exploiting previously acquired knowledge [32, 267].

Our contributions can be summarized as follows:

- We propose *UniModel*, a single multi-task multi-modal network to learn representations with data acquired from a variety of sensors from different domains.
- To the best of our knowledge, we, for the first time, explore learning a unified (or one) model for sensing tasks using unaligned data (i.e., signals collected for different tasks).
We apply our approach to several important recognition problems, including heart rate estimation, sleep stage scoring, activity recognition and abnormal electroencephalogram detection. We show that the performance of UniModel for considered tasks is similar to, in some cases, even higher than the single-problem (or single-task) models.

We explore the effectiveness of self-supervised learning to create a general-purpose model from unlabeled data. Specifically, to the best of our knowledge, we, for the first time, utilize contrastive predictive coding (CPC) in a multi-task learning setting for pretraining a model simultaneously across tasks.

We demonstrate significant generalization improvement in low-data regimes with a UniModel trained in a self-supervised manner compared to a supervised counterpart.

We investigate whether the representations learned with UniModel are transferable across datasets, tasks, and devices. We show that, after fine-tuning, our approach outperforms standard models trained and tested on the in-domain dataset.

10.2 Approach

We present an approach to learning unified representations for multiple tasks and modalities with a unified neural network. In this section, we formally define our problem and present our learning strategy. Next, we discuss self-supervised learning approach based on contrastive predictive coding for multi-task learning. Finally, we describe an approach to leverage self-supervision for semi-supervised learning.

10.2.1 Problem Formulation

Here, we provide a formal description of learning a UniModel for solving numerous sensing problems in a supervised manner and with self-supervision, to mitigate the requirement of well-annotated data.
We consider labeled data \( D_s^\tau = \{(x_i^\tau, y_i^\tau)\}_{i=1}^{m^\tau} \) from a distribution \( \mathcal{M}^\tau \) over the domains \( \mathcal{X}^\tau \times \mathcal{Y}^\tau \), where \( \mathcal{X} \subseteq \mathbb{R}^d \) is the input space with \( d \) input channels and \( \mathcal{Y} \) is label space for a learning task \( \tau \) and \( s \) representing labeled examples. Specifically, given multiple tasks each with its own set of \( m \) examples \((x_i, y_i)\) in input-label pairs, our goal is to learn a joint predictive model \( f_\theta : \mathcal{X}^\tau \rightarrow \mathcal{Y}^\tau \) that fits \( \forall \tau \in \mathcal{T} \), where \( \mathcal{T} \) is a set comprising learning problems.

We approach the problem of learning \( f_\theta \) through extracting task representations as \( f_{e^\tau}(x) = h \in \mathbb{R}^r \) over the task-specific input. Afterwards, we learn task-agnostic representations with a shared encoder \( f_{e^s}(h) = z \in \mathbb{R}^g \) over task-related features \( h \). Finally, we learn a predictor \( y = f_c(z) \in \mathcal{Y}^\tau \). We form a unified model through the composition of encoders and predictors to use a joint-loss \( L = \sum_{\tau \in \mathcal{T}} \ell^\tau \) for training the model end-to-end. The multi-task networks are a composition of task-specific and shared layers with hard parameter sharing amongst the tasks. Each task’s input goes through its respective layers before being processed with a shared model. Our model has hard parameter sharing amongst the tasks, which may result in robust representations invariant to noise. It also produces a compact model that can be used for multiple tasks directly on-device, e.g., on a smartphone.

We are further interested in representation learning techniques, where we can learn \( f_{e^\tau} \) and \( f_{e^s} \) with the help of a pretext task using unlabeled data \( D_u^\tau = \{x_j^\tau\}_{j=1}^{n^\tau} \) from a distribution same as or different than \( \mathcal{M}^\tau \). We pose a pretext task as a self-supervised (or unsupervised) learning problem which depends solely on input \( x \). To that end, we leverage a generic self-supervision task to correctly predict the latent embeddings in future time-steps of \( x \) using an encoder and auto-regressive model to represent the encoded embedding. A noise contrastive estimation technique, such as the InfoNCE loss [42] is then used to maximize the mutual information between the features that the model learns and those extracted from future time observations. It is also helpful that we do not need to hand-design an auxiliary task to learn from multi-modal data. Further, it could be significantly challenging and does not guarantee that the learned features will be useful for the end task of interest. For this purpose, we leverage the contrastive predictive coding [43] formulation in a multi-task setting, which allows learning a range of tasks from multiple sensing modalities, simultaneously.

### 10.2.2 Unified Sensor-based Multi-task, Multi-modal Learning

We introduce unified model or UniModel for short, a temporal convolutional-based neural network applicable to a variety of sensing tasks. Figure 10.1 illustrates the high-level architecture, which comprises individual convolutional subnets, a fusion network that is shared across tasks, and output layers for the considered tasks.

The initial modeling step is to extract important features for the tasks independently and bring all sensor modalities into a similar structure to process them further with a shared set of layers. A task-specific subnet is also essential as the input size and number of modalities (or input channels) differ across problems. We use convolutional neural networks \( f_{e^\tau}(\cdot) \) to learn salient features for each task with their own set of parameters. We keep the subnet compact to control the overall network size as adding new tasks increases the parameters that the model needs to learn with an overhead for inference at run-time. Another key reason for the design
choice is to leverage the fusion network to learn high-level features shared across tasks. Our network comprises a pair of 1-D convolutional layers to process the input modalities stacked depthwise. Each of the convolutional layers is combined with a LayerNorm and PReLU non-linear activation function. The subnet also consists of an InstanceNorm operation to normalize each instance independently. We preserve the original input resolution with the same padding in the convolutions but expand features maps that the shared network uses as an input to create an embedding space useful for many sensors. Likewise, our multi-modal subnet or shared encoder is also a convolutional model with strong parameter sharing among tasks. It leverages complementary aspects of each signal to learn representations that are general and useful across multiple tasks. The rest of the implementation details of the architecture are explained in Section 10.3.2.

10.2.3 Multi-task Contrastive Predictive Coding

A central problem in machine learning is to design (or learn) features that provide discriminative information about the input data. Their quality enables a learning algorithm to efficiently solve a task at hand and generalize well on unseen data. The representations that capture high-level information about the input facilitate the model to perform well on downstream tasks while improving data efficiency. To learn sensory representations with deep neural networks, we require a large amount of labeled data, which is prohibitively expensive to acquire for many problem domains in a real-world setting. The area of self-supervised learning describes a class of methods that aim at learning useful representations from unannotated data. It tasks the network to solve an auxiliary objective for which supervision can be acquired from the data itself. Given an unlabeled dataset $D_y = \{x\}^n_{j=1}$ and network $f_\theta(\cdot)$, the aim is to pretrain the network through solving a surrogate task where, labels $y$ for the standard objective functions are extracted automatically from $x$. The learned model is then utilized as initialization for rapidly learning an end-task.

We use contrastive predictive coding (CPC) for providing self-supervision to a deep model that relies on mutual information maximization criteria. CPC lets the network learn features that can be used to make optimal estimates of future observations through maximizing mutual information between the predicted representations based on data that has been recorded so far compared to those extracted from future time-steps. The key intuition behind CPC is to extract structure from input at different levels, ultimately deciding the horizon of long-term predictions we can make with the model. For instance, slowly varying features are learned if the model predicts further in the future than making predictions on a smaller time-horizon.

CPC is applied successfully to image and speech modalities to learn features from unlabeled data for single-task models. Here, we utilize it for a multi-task setting for jointly learning representations from multiple sensing modalities, such as accelerometer, electroencephalography, and photoplethysmogram. For the sake of simplicity, we provide a formulation for a single task, and it applies straightforwardly to multiple tasks. Consider a raw multivariate signal $x = [x_1, x_2, \ldots, x_L]$ of length $L$ with $x_i \in \mathbb{R}^d$. A task-specific encoder $h = f_{e^t}(x)$ encodes the raw signal into a latent embedding related to a task. It is then processed with a shared encoder $z = f_{e^s}(z^t), z \in \mathbb{R}^r$ (with $r$ being dimension of feature vector) to pro-
duce sensor-agnostic representations. We employ recurrent network with the gated recurrent unit [269] as an auto-regressive model \( f_a \) to generate a summary of past representation as a context vector \( c \). It has hidden units equivalent to the last convolutional layer’s features maps in \( f_e^\tau \). The network consisting of these building blocks learns the representation while maximizing mutual information between context vectors \( c \) and future embeddings \( z \) with a lower-bound on mutual information with InfoNCE [42] as:

\[
\mathcal{L}_{\text{NCE}} = \frac{1}{K} \sum_{k=1}^{K} \log \frac{s_k(c_t, z_{t+k})}{\sum_{x \in \mathcal{X}_{t,k}} s_k(c_t, z_n)},
\]

\( s(c_t, z_{t+k}) = \exp(c_t^\top W_k z_{t+k}) \)

where \( W \) are learnable parameters of log-bilinear model \( s \), the \( K \) denotes the total time-steps to predict into the future while \( t \) represents a starting point from where we start estimating future states, and it is randomly selected depending on the resolution of \( z \). We selected the value of \( K \) with an initial exploration of our considered tasks. The \( \mathcal{L}_{\text{NCE}} \) is a contrastive objective that minimizes a dot product between the predicted and correct future representation while maximizing the dot product with a set of negative examples \( \mathcal{X}_{t,k} \) taken from the mini-batch. We use the same formulation to compute the loss for each task and sum them across tasks \( \mathcal{L}_{\text{aggregated}} = \sum_{\tau \in \mathcal{T}} \mathcal{L}_{\tau \text{NCE}} \) for multi-task learning.

### 10.2.4 Semi-Supervised Learning for Sensing

As mentioned earlier, the curation of a massive amount of well-annotated data for training deep models is prohibitively expensive and requires extensive effort. We can leverage a model bootstrapped with self-supervision for semi-supervised learning at no extra cost. The self-supervised pretrained network captures high-level representations that lower the bar of annotated data and improve end-task data efficiency. To that end, once the self-supervised network is trained to produce unified sensory representations, we can use \( z \) as input to further predictive models for solving the end-task, such as sleep stage scoring and heart rate estimation. The key possibilities are to either keep the encoder fixed or fine-tune it along with the end task. For learning with few labeled data, we choose to fine-tune an entire network on the downstream task with various amounts of labeled data. We keep \( f_e^\tau (\cdot) \) and \( f_e^s (\cdot) \) encoders and discard rest of the layers from the CPC model. We apply a global average-pooling to accumulate features \( z \) from the encoder, and a linear layer follows it with the hidden units equivalent to task outputs. For classification and regression problems, we use softmax and linear activation, respectively. We fine-tune the resulting model end-to-end on the downstream task.

### 10.3 Experiments

We validate the effectiveness of UniModel on several publicly available datasets from different domains. We find that learning a unified model provides similar or better performance
<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>#Channels</th>
<th>#Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity Recognition</td>
<td>HHAR</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>MotionSense</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heart Rate Estimation</td>
<td>PPG-DaLiA</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Sleep Stage Scoring</td>
<td>Sleep-EDF</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Abnormal EEG</td>
<td>TUH Abnormal</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>Seizure Detection</td>
<td>CHB-MIT</td>
<td>17</td>
<td>2</td>
</tr>
</tbody>
</table>

As compared to individual task-specific models. We show that self-supervised pretraining is highly effective in leveraging unlabeled data to improve performance in the low-data regime, even on out-of-domain data. In the rest of the section, we provide details of the tasks and datasets used to assess UniModel. We also discuss evaluation strategy and key implementation details. Finally, we provide experimental results on various sensing tasks.

### 10.3.1 Tasks andDatasets

To evaluate our approach, we focus on health, well-being, and daily living problems that require sensory data collected from smartphones and other wearables. We target human activity recognition, sleep stage scoring, heart rate estimation, and seizure detection problems. Specifically, we evaluate UniModel on seven different datasets, four of which are used for core multi-task learning (i.e., HHAR, SleepEDF, PPG-DaLiA, and TUH Abnormal EEG) and evaluation. The other three (MotionSense, HHAR-SW, and CHB-MIT) are utilized for evaluation in transfer settings due to their relatively small size.

**Activity Recognition:** For activity recognition, we choose datasets with accelerometer and gyroscope signals: HHAR, MotionSense, HHAR-Smartwatch (SW). To summarize, the HHAR dataset contains 8 smartphones and 4 smartwatches data from 9 subjects performing 6 activities, i.e., biking, sitting, standing, walking, stairs-up, and stairs-down. The sampling rate of signals varies considerably across devices with values ranging between 50 – 200Hz. We use smartphone data in core training and use smartwatch signals in assessing generalization across devices. Similarly, MotionSense comprises inertial sensors’ data acquired at a sampling rate of 50Hz with iPhone 6S from 24 users who performed 6 activities, namely, sitting, walking, upstairs, jogging, downstairs, and standing. We segmented the signals into fixed-size windows that have 400 samples with 50% overlap for all the considered datasets.

**Sleep Stage Scoring:** For sleep stage scoring with single-channel EEG, we use Physionet Sleep-EDF dataset consisting of 61 polysomnograms. It is collected from 20 subjects to study the effect of a) age on sleep in healthy individuals and b) effects of temazepam on sleep. The dataset includes 2 whole-night sleep recordings of EEGs from FPz-Cz and Pz-Oz channels, EMG, EOG, and event markers. The signals are provided at a sampling rate of 100Hz, and sleep experts annotated 30 seconds into 8 classes. The classes include Wake (W),
Table 10.2: Comparison of UniModel trained jointly on four sensing tasks and individual models for each task. We evaluate the generalization capability of our approach in both supervised and self-supervised settings. SSL refers to self-supervised learning, a model that is pretrained and fine-tuned end-to-end on the downstream tasks. We report mean±std. deviation of metric scores averaged over 10 independent runs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Heart Rate Estimation</th>
<th>Sleep Stage Scoring</th>
<th>Activity Recognition</th>
<th>Abnormal EEG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean Absolute Error</td>
<td>Accuracy</td>
<td>F-score</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Single-problem Supervised</td>
<td>7.97±0.48</td>
<td>0.787±0.018</td>
<td>0.791±0.013</td>
<td>0.842±0.014</td>
</tr>
<tr>
<td>SSL (Fine-tuned)</td>
<td>7.66±0.33</td>
<td>0.792±0.008</td>
<td>0.791±0.006</td>
<td>0.853±0.016</td>
</tr>
<tr>
<td>UniModel (Joint Network)</td>
<td>8.09±0.46</td>
<td>0.795±0.011</td>
<td>0.782±0.014</td>
<td>0.852±0.017</td>
</tr>
<tr>
<td>SSL (Fine-tuned)</td>
<td>7.78±0.33</td>
<td>0.795±0.011</td>
<td>0.796±0.010</td>
<td>0.872±0.014</td>
</tr>
</tbody>
</table>

Rapid Eye Movement (REM), N1, N2, N3, N4, Movement and Unknown (not scored). We applied standard pre-processing as proposed in [108] to merge N3 and N4 stages into a single class following American Academy of Sleep Medicine (AASM) standards and removed the un-scored and movement segments. We utilize EEG (Fpz-Cz channel) signals from an initial study to categorize sleep into 5 classes, i.e., W, REM N1, N2, and N3.

Heart Rate Estimation: The pulse oximeters sensor in contemporary wearable devices provides a low-cost, non-invasive and efficient way to estimate heart rate. We use PPG—DaLiA dataset that is acquired from 15 subjects using Empatica E4 [270] a wrist-worn device. It consists of photoplethysmogram (PPG) and accelerometer signals recorded at 64Hz and 32Hz, respectively. The data collection protocol included low, medium, and high-intensity arm movements to introduce motion artifacts that make accurate estimation cumbersome. Similarly, to generate a highly irregular heart-rate, participants completed activities of varying physical effort (e.g., cycling vs. sitting). The ground-truth is obtained from the ECG recorded with the RespiBAN device mounted on the chest. We use a window size of 8 seconds to segment the signals as in [270] where ground-truth heart rate is provided at the same resolution. We up-sample the accelerometer signal to match PPG’s sampling rate to have an input stacked depth-wise for the network. We apply z-normalization to scale the output for training with the mean and standard deviation computed from training set examples.

Electroencephalography Tasks: Electroencephalography (EEG) is a non-invasive approach to record electrical brain activities from the user’s skull and scalp. It has a wide range of applications in medicine, such as diagnosis of epilepsy, seizure detection, and in building the brain-computer interface domain. Inherently, EEG signals are prone to noise and require collecting a large amount of data either for multiple sessions to perform meaningful neurological conditions analysis. Moreover, the acquired data volume makes it significantly challenging for manual inspection; thus, automated methods are preferred. We focus on the tasks of recognizing if the recorded EEG signal is normal or abnormal to decide if it can be useful for the downstream application or not. We use TUH Abnormal EEG Corpus containing EEG signals recorded at 250Hz and annotated them as clinically normal or abnormal. It is a part of the TUH EEG Corpus [197], a most extensive publicly available corpus comprising over 15000 subjects and session recording as per the international 10 — 20 system. We use 21 common channels set across subjects and segments of approximately 8 seconds to learn and evaluate models. In a transfer setting, for detecting seizures, we use CHB-MIT data [198] containing intractable seizures acquired from 23 pediatric subjects with EEG signals sampled.
at 256Hz with 17 electrodes or channels. The varying duration of the sessions resulted in the input of different lengths. We use segments of 500 samples randomly selected (and padded with zeros if necessary) from the training sequences during the learning phase. For evaluation, we use an entire recording to generate the model’s prediction.

**Validation Procedure:** We employ a train-test split (70 – 30) of the datasets in all the cases as we can not perform a leave-one-user-out cross-validation strategy due to a varying number of users across the datasets. We split the data based on subjects whenever possible (except TUH Abnormal EEG Corpus and CHB-MIT data where we perform a standard split). We compute the weighted F1-score and accuracy metrics for our method’s performance analysis. Importantly, we use the same network architecture in all the cases along with fixed hyper-parameters. We compare the results of our approach and the baselines using a t-test, with a significance level of 0.01 as suggested in [271].

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MotionSense</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-domain Supervised</td>
<td>0.852±0.016</td>
<td>0.854±0.016</td>
</tr>
<tr>
<td>Single-problem SSL (Fine-tuned)</td>
<td>0.879±0.012</td>
<td>0.881±0.012</td>
</tr>
<tr>
<td>UniModel (Joint Network)</td>
<td>0.874±0.012</td>
<td>0.876±0.019</td>
</tr>
<tr>
<td></td>
<td>0.873±0.006</td>
<td>0.881±0.006</td>
</tr>
<tr>
<td><strong>HHAR-SW</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-domain Supervised</td>
<td>0.643±0.017</td>
<td>0.614±0.016</td>
</tr>
<tr>
<td>Single-problem SSL (Fine-tuned)</td>
<td>0.677±0.011</td>
<td>0.674±0.013</td>
</tr>
<tr>
<td>UniModel (Joint Network)</td>
<td>0.671±0.014</td>
<td>0.670±0.014</td>
</tr>
<tr>
<td></td>
<td>0.695±0.013</td>
<td>0.695±0.012</td>
</tr>
<tr>
<td><strong>CHB-MIT</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-domain Supervised</td>
<td>0.873±0.026</td>
<td>0.871±0.026</td>
</tr>
<tr>
<td>Single-problem SSL (Fine-tuned)</td>
<td>0.874±0.013</td>
<td>0.872±0.013</td>
</tr>
<tr>
<td>UniModel (Joint Network)</td>
<td>0.899±0.025</td>
<td>0.892±0.039</td>
</tr>
<tr>
<td></td>
<td>0.923±0.011</td>
<td>0.922±0.013</td>
</tr>
</tbody>
</table>

**Table 10.4:** Evaluation of self-supervised *UniModel* with a linear classifier trained on-top of a frozen network in comparison to fine-tuning.

<table>
<thead>
<tr>
<th>Method</th>
<th>Heart Rate Estimation</th>
<th>Sleep Stage Scoring</th>
<th>Activity Recognition</th>
<th>Abnormal EEG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean Absolute Error</td>
<td>Accuracy</td>
<td>F-score</td>
<td>Accuracy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frozen</td>
<td>15.89±0.461</td>
<td>0.716±0.004</td>
<td>0.712±0.005</td>
<td>0.748±0.013</td>
</tr>
<tr>
<td>Fine-tuned</td>
<td>7.79±0.193</td>
<td>0.795±0.011</td>
<td>0.796±0.010</td>
<td>0.877±0.014</td>
</tr>
</tbody>
</table>

**10.3.2 Implementation Details**

We use a temporal convolutional model to learn representations from multi-modal data of cross-domain tasks. Our model is inspired from [42] with task-specific encoder $f_{es}(\cdot)$ has a pair of convolutional layers having a kernel size of 10 and 24 filter maps. The multi-modal subnet or shared encoder $f_s(\cdot)$ consists of a stack of 6 convolution layers with 32, 64, 72, 96, 128, and 256 feature maps and kernel sizes of 10, 8, 6, 6, 4, and 4 in each layer with a stride of 1, respectively. We also use LayerNorm and PReLU activation here, same as a task-specific subnet, after the convolution layers. After every second convolution layer, we add a max-pooling layer with a pool size of 8 and stride of 2 to reduce the input size and bring translation invariance to low-level changes in the input. We adopt the same architecture for an encoder composed of task-specific and fusion networks for learning representations.
in a supervised and self-supervised manner. Importantly, the same architecture is used as a backbone network in single-task models with an addition of output layers on top. To generate task-related outputs, i.e., class-label or scalar value depending on the learning problem, we use global average pooling to aggregate the features and pass them to the dense (output) layers of each task. These layers have hidden units corresponding to the number of outputs with either softmax or linear activation.

We use an Adam optimizer \[ 3 \] with a learning rate of \( 10^{-3} \) and a batch size of 24. We train single-task networks for 50 epochs with a standard cross-entropy loss for classification and mean absolute error for regression tasks. In the case of a multi-task network, we prepare the mini-batch with uniformly sampled data points for each task and backpropagate the error jointly (simultaneously) on the tasks to update the parameters. The different tasks are learned at a different rate and can also diverge from the optimal solution; therefore, we monitor training loss and stop learning as soon as total loss increases. We notice network converges after 10 epochs on the joint data of the considered tasks. For pretraining with contrastive loss, we let the network predict \( K = 12 \) time-steps ahead in the future for all the tasks. We explored other values but did not find a significant difference in performance. We select a starting point \( t \) randomly for each task and training step (or mini-batch creation), depending on the output resolution of encoder \( f_{e^*}(\cdot) \). Once pretraining is finished, we fine-tune the model end-to-end with labeled data of end task for 50 epochs. In this case, we select the task-specific encoder depending on the task’s available modalities, including in transfer to other datasets. We use L2-regularization with a rate of \( 10^{-4} \) on all layers in the network except output. We also apply a dropout of 0.1 in the auto-regressive function in the CPC model.

### 10.3.3 Results

In this section, we discuss the key contributions to show the effectiveness of UniModel both as a supervised and a self-supervised network against single-problem (or single-task) baselines trained either in a fully supervised manner or with self-supervision. We designed learning strategies for our model to be trained jointly on all tasks simultaneously with semantic annotations in an entirely supervised manner and exploit unlabeled data to learn useful representations with a self-supervised loss. After pretraining, we fine-tune UniModel end-to-end on each task, including in low-data regimes and evaluating transferability.

#### Train One for All

Table 10.2 reports the mean and standard deviation of the performance metrics for each considered task using UniModel and single-problem (or single-task) models trained with supervised or self-supervised learning. In the supervised case, UniModel achieves an F-score and accuracy of 80% to detect abnormal EEG, which corresponds to 4 and 5 percentage points increment on top of single-task counterpart. The performance of UniModel for the rest of the tasks is similar to corresponding single-task models. These results show the effectiveness of UniModel to learn multiple tasks jointly with similar to, in some cases, even better, performance than the single-task models. Although UniModel offers a simple architecture with
Figure 10.2: Effectiveness of self-supervised UniModel in a low-data regime. We use the pretrained model as initialization and fine-tune it end-to-end with a few labeled data for each task (i.e., SSL-FT). We perform 100 independent runs, sampling different labeled examples for learning and report average metric scores. The metrics are statistically significant according to the t-test with $p<0.01$, except for HHAR-SW with 50 instances and CHB-MIT with 20 and 50 instances. The dotted line represents the upper bound performance that can be achieved with a single-task fully-supervised model trained on entire labeled data.

hard parameter sharing among different tasks, the results are competitive with the single-task baseline specifically designed to learn only one task. This outcome highlights the fact that UniModel is able to leverage the knowledge from multiple tasks efficiently. Hence, in resource-constrained devices, a unified general-purpose model could substitute multiple single-task models.

To understand the impact of self-supervised learning, we compared the performance of UniModel trained with supervised and self-supervised learning. The F1 scores of fine-tuned UniModel from Table 10.2 for sleep stage scoring, activity recognition, abnormal EEG are 79%, 87% and 80%, which are 1-2 percentage points higher than or equal to UniModel trained in a supervised approach. The mean absolute error for heart rate estimation is 7.78, which is approximately a 1 point decrease – in this case, decreasing refers to a better performance – from the supervised UniModel. These results underpin our assumption that training the joint model first on unlabeled data and then fine-tuning on labeled data helps the model to learn unified representations for many tasks. Furthermore, in Table 10.4, we compare training a linear classifier on top of a frozen feature extractor with fine-tuning the entire pretrained model. The key objective of the experiment is to study the quality of learned representations solely with CPC. We observe that the learned (fixed) features are highly useful as a linear classifier can discriminate among the underlying classes with a reasonably good recognition rate.

Leverage Knowledge Transfer

We use the model trained in the previous section to learn new tasks (including activity recognition using different devices and seizure detection) across MotionSense, HHAR-SW, and
CHB-MIT datasets. The performance of UniModel is compared against single-problem learning on in-domain examples regarding supervised, and SSL (fine-tuned) approaches. The single-problem models for HHAR-SW and MotionSense with IMU sensors and CHB-MIT dataset with EEG signals are pretrained based on HHAR datasets and TUH Abnormal EEG datasets. Specifically, to tackle a varying numbers of channels between EEG datasets, we pad instances with zeros channel-wise. We use transferred models constituting task-specific and shared encoders and add a randomly initialized layer on top to fine-tune an entire network end-to-end.

Table 10.3 shows the comparison of transfer learning performance of both variations of UniModel against in-domain and single-problem (SSL) models pretrained with the contrastive predictive coding task. The in-domain models are trained directly on target data without any transfer, while single-problem (SSL) networks are pretrained and transferred from related data as described earlier. We observe UniModel significantly outperforming its counterparts across HHAR-SW and CHB-MIT datasets for activity recognition with smartwatch and seizure detection problems, respectively. On the latter task, our SSL model achieves an accuracy of 92.5%, which is 5, and 11 percentage points better than baselines. On MotionSense data, fine-tuning UniModel can reach almost the same accuracy, and F-score as fine-tuned single-problem provides but better than training from scratch on in-domain data. In addition, UniModel shows an average accuracy and F-score improvements of 4.2% and 5.1%, respectively, across all tasks compared to in-domain training and 4.1% and 5.7% improvements against fine-tuning the single-problem model. We can conclude that UniModel provides a more robust unified feature learning strategy that can share knowledge between different problems while avoiding overfitting towards any of the consisting tasks.

Label-efficiency with Less Supervision

The labeled data are not always available; hence, we have to deal with no or a few amount of annotated data in most cases. In this set of experiments, we aim to show the data-efficiency of UniModel against its supervised and self-supervised (single-problem) baselines. In all of the studied tasks, the single-problem model fails to learn a generalizable model as the size of labeled data decreases. We use self-supervised pretrained UniModel as initialization to rapidly learn downstream tasks with few labeled examples. We perform 100 independent runs of training and evaluation with different fractions of labeled examples. For each run, we randomly sample \( m \) instances per class from a training set for fine-tuning and report the average metric score on the test set. For the regression task of heart-rate estimation, we use 10 times more instances for learning models. Notably, only sampled examples are used to perform mean-normalization of heart rate output as described in Section 10.3.4. We show that UniModel through jointly-learning to solve multiple tasks can reach a higher recognition rate in even a low-data regime.

Figures 10.2 compare the ability of UniModel against single-problem supervised and self-supervised approaches on handling limited supervision situations. For the latter, we directly train a network from scratch with available data, while for the former, we pretrain a network with self-supervision on a single problem and then fine-tune it on the downstream task. In the transfer setting for MotionSense or HHAR-SW and CHB-MIT, a single-problem self-
supervised model is learned with HHAR and TUH Abnormal EEG datasets, respectively. We notice that the performance of both single-task models substantially degrade when less supervisory data is provided. However, UniModel can benefit from the shared information between multiple tasks. Likewise, contrary to the supervised counterparts, our approach provides a significant improvement in generalization; for instance, on activity recognition task with 20 labeled examples per class UniModel achieves a weighted F-score of 70% as compared to 55% of a model trained from scratch. Similarly, in out-of-domain transfer overall, our self-supervised approach performs better and dramatically improves label efficiency.

10.4 Related Work

Multi-Task Learning (MTL) approaches aim to learn multiple separate tasks simultaneously by leveraging the knowledge and relation among tasks generally through a partially shared network [32]. The features learned with these techniques are better transferable between different tasks without overfitting any of them. MTL methods can be more data-efficient (i.e., requiring less labeled data) than learning a single task by taking advantage of the knowledge and shared aspects of representations from other tasks. In practice, MTL models are often shown to provide higher performance compared to single-task learning [267].

MTL has been applied in a variety of application domains, such as computer vision [272, 273, 274], natural language processing (NLP) [275, 276], recommender systems [277], voice recognition [278], human activity recognition [14], sentiment classification in text [279], traffic and flow prediction [280], and health and well-being [281, 282]. However, all of these methods consider single modality, non-sequential data (e.g., images), while in this work, we focus on sequential (time-series or sensory) data from sensors of different domains (multiple modalities). Recently, in contrast to multi-task models for single-modality, the models leveraging multiple modalities are especially designed to capture the relations between data from different domains, such as text, audio, and image simultaneously (i.e. Visual Question Answering [283] or sentiment analysis in videos [284]).

For sequential data, MTL has been studied mainly in NLP, [276, 285], and computer vision (video) applications [286], which only consider one type of modality (text or video, respectively). However, the closest works to ours are multi-modality multi-task learning methods, such as MultiModel [261] which focuses on learning one model for tasks involving images, audio, text, and tabular datasets in a fully supervised manner. In comparison, we focus on tasks utilizing sensory data (e.g., physiological signals), and in addition to a supervised learning setting, we also focus on self-supervised learning. Likewise, Shazeer et al. [287] propose a framework based on a Mixture of Experts and self-attention mechanisms to learn representations from different modalities, including language, image, audio, and categorical data at the same time. Other than text or video as sequential data, MoSE (Mixture of Sequential Experts) introduces a framework based on a sequential data log from G-Suite applications to predict user activity and improve decision making in Gmail [288]. As a sequential single modality approach, Spathis et al. [281] propose a method to forecast mental moods based on sequences of self-reported data. In contrast, we target heterogeneous cross-domain sensor data of tasks in ubiquitous computing and ambient sensing, which is not addressed in previous works. Like-
wise, we focus on the self-supervised learning of a unified model to simultaneously leverage data of different modalities to demonstrate data efficiency in low-data regimes.

Recently, there has been increasing research interest in jointly learning to solve multiple tasks in sensor data \([289, 290, 291]\). Taylor et al. \([289]\) proposed an MTL model for data from different sequential modalities, including survey data, weather records, wearable sensors, and smartphone to predict mood, stress, and health per individual. The Abedin et al. \([291]\) propose a deep clustering framework for human activity data based on multi-task learning. The authors defined future prediction and frame reconstruction as auxiliary tasks to train the model to learn representations in a self-supervised manner. However, the definition of tasks is different in \([69, 289, 290]\) as it proposes a multi-task learning framework for personalized human activity recognition (treating each user as an independent task).

While various complex methods and architectures have been proposed, Liu et al. \([276]\) investigate the effectiveness of three main MTL architectures with different inter-task parameter sharing strategies. They show the superiority of the simplest model, which contains task-related encoders followed by a shared module. In our work, we follow the same strategy. In addition to the complex architectures, existing MTL methods mainly focused on 1) single modality; or 2) single type of task but for multiple instances (e.g., users). In the latter case, each user (or instance) is considered as an individual task, and 3) fully-supervised models. However, our proposed method is general-purpose, self-supervised, and applicable over multiple tasks with varying sensory modalities simultaneously.

Moreover, the usage of ubiquitous sensors is prevalent these days; annotating the recorded data is prohibitively expensive and inaccurate \([271]\). This issue emphasizes the role of self-supervised learning (SSL) methods in sensor data applications, which require no manual annotation. Recently, SSL has been used for representation learning in human activity recognition \([14, 5, 7]\), time-series change-point detection \([13]\), computer vision \([292, 293]\), NLP and reinforcement learning \([28]\), audio processing \([16, 294, 295]\). Some are applicable over multi-modality data such as \([296, 297, 298]\). Mostly, these methods are based on contrastive learning that has become a powerful tool in self-supervised learning to extract useful features from unlabeled data. Hence, we employ the contrastive predictive coding approach \([25]\) to train our unified model (as compared to single-task models of prior work) to learn compact informative representations for all tasks together.

\section{Conclusion}

We introduce a unified deep neural network, termed UniModel, for learning to solve multiple sensing problems simultaneously from multi-modal data. We demonstrate the effectiveness of our simplistic and easy-to-implement joint-network with hard parameter sharing among tasks in learning general-purpose representations with both supervised and self-supervised learning. In particular, we utilize contrastive predictive coding in a multi-task setting to exploit large-scale data for pretraining sensory models. Using data acquired from several sensors embedded in wearable devices, such as accelerometer, gyroscope, photoplethysmogram and electroencephalography signals as input to UniModel we achieve an F1 score of 79.6\%, 87.6\%,
80.8% for solving *sleep stage scoring*, *activity recognition* and *abnormal electroencephalogram detection* tasks, and an MAE of 7.78 for *heart rate estimation* task. We examine the feasibility of UniModel in contrast to models created individually for each problem, including strong transferability on out-of-domain data from different devices and tasks. Furthermore, we show that the UniModel trained with self-supervised learning using unlabeled data significantly outperforms their fully-supervised counterparts in the low-data regime. To conclude, we hope this work will popularize training unified models with self-supervised learning on massive unlabeled sensor datasets in a real-world setting to improve label efficiency across various sensing domains. In future work, we aim to investigate negative transfer in a self-supervised setting and study deployment (or efficiency) related issues of a unified model for on-device inference.

---

Various icons used in the figures are created by Rahmadi Kurniawan, Vectors Point, Universal Icons and Ben Davis from the Noun Project.
Chapter 11

Conclusion

This thesis described methods for learning high-level representations from raw signals using deep neural networks to support the development of novel sensing applications and empowering self-learning Internet-of-Things (IoT). To enable a wide range of ambient and personal sensing tasks, we put a strong focus on approaches that are: a) highly practical and general-purpose, b) simple and light-weight, c) require minimal effort in data labeling, d) are reliable and robust to failures in the sensing system, and e) have an in-built mechanism to preserve privacy. The developed methods in the thesis lie on the intersection of deep learning, ambient (or pervasive) sensing, and ubiquitous computing. They provide a strong foundation for building data-driven predictive models for various domains ranging from health monitoring, well-being, elderly care, affective computing, human mobility to industrial automation and agriculture.

Our core contributions are in representation learning with self-supervision for which we design and introduce numerous auxiliary tasks that extract the intrinsic supervisory signal from unlabeled inputs to train deep models with standard objective functions. Our self-supervised approaches applicable to different types of sensors learn broadly useful features that significantly improve data efficiency, are vastly transferable, competitive with fully-supervised counterparts, and do not require data to be aggregated in a centralized repository for learning (Chapters 3, 4, 5, and 6). We further propose techniques to make neural networks-based models adaptable and reliable when encountering noisy inputs, so their performance degrades gracefully (Chapters 7 and 8). Notably, we also study the problem of learning a unified model to solve many related cross-domain tasks with a single neural network that exploits the shared structure of the data (Chapters 9 and 10).

Having described our key methods and discussion of the results in previous chapters to substantiate this thesis, we seek to provide answers to the research questions introduced in Chapter 1. We recap the contributions and provide pointers to future research directions. To summarize, we follow the structure of Chapter 1, grouping the research questions into four themes. After that, we reflect on the guiding hypothesis that unsupervised methods can approach supervised methods in performance. To that end, first, we address self-supervised
learning from a wide variety of unlabeled sensory data, which in practice could be distributed across IoT devices. Second, we focus on the robustness and reliability of the deep model on noisy inputs either due to missing modalities or channels being inconsistent. Lastly, we look into jointly learning to solve multiple tasks with a unified model and tackle personalization.

How can we utilize large-scale sensory data without semantic labels to learn high-level representations?

It is challenging to answer such a question in general due to its wider scope. We study it through exploring varied examples in four research questions below.

**Research Question 1:** Do self-supervised pretext tasks enable learning useful representations with deep neural networks from unlabeled sensor data that are competitive with the fully-supervised counterparts?

In a variety of contexts, we explored the basic idea of self-supervising the network in different ways and we show that our proposed auxiliary tasks generate meaningful representations. The *transformation prediction network* in Chapter 3 provides an important initial step towards highlighting the effectiveness of self-supervision in learning from sensory data. The pretext tasks developed within *sense and learn* framework in Chapter 4 further showed that simple objectives could be used to leverage large-scale unlabeled data for pretraining the deep models without requiring human involvement. In particular, the auxiliary tasks, such as feature prediction from masked window, blend detection, and odd segment recognition demonstrate generalization on-par with transformations while being easy-to-implement and without indulging into selection of appropriate transformation functions for learning invariant features. The *scalogram contrastive learning* approach developed in Chapter 5 further showed that multi-view learning (i.e., aligning different views of the same data sample) with a contrastive loss is a powerful way to learn high-level representations. On a broad range of tasks and sensory modalities, we observe that self-supervision provides results on-par with supervised methods, which have direct access to task labels. Furthermore, it is also better than input reconstruction techniques (e.g., autoencoder) with not just being better in performance but also not requiring a symmetric decoder, which reduces the number of parameters to learn, makes the model compact ideal for on-device learning, and offer flexibility in designing better encoder architectures. For the cases we studied, we conclude that self-supervised learning is a viable alternative to classic supervised approaches when semantic labels are infeasible and expensive to acquire, we may expect this to generalize better. The interesting open question is to explore self-supervision for continual learning where a model has to learn tasks sequentially without having access to the task identities. Likewise, another open problem is to devise ways to counter shortcut learning in neural networks [97], i.e., the model learns semantic representation as opposed to merely modeling the noise via capturing spurious correlations to solve auxiliary tasks. The shortcut learning is a well-known phenomenon in deep learning and equally applicable to supervised learning regimes as it is to self-supervision.

**Research Question 2:** Does self-supervised pretraining improve label efficiency to achieve better generalization on downstream tasks with few-labeled examples and does it induce inductive bias required for transfer across the domain?
To answer the defined research question, we analyze the quality of learned representations with self-supervision in the low-data regimes and in a transfer setting. We evaluate our proposed pretext tasks for their efficacy to improve data efficiency and transfer of the pretrained model to other related downstream tasks in Chapters 3, 4, and 5. We focus on demonstrating that leveraging unlabeled data improves performance when a model is fine-tuned with few-labeled data points (i.e., in a semi-supervised setting) compared to training from scratch. Our results across datasets and tasks highlight that we can significantly reduce labeled data requirements, and as few as five to ten instances per class are sufficient to improve generalization over training from scratch. Likewise, we achieved encouraging results in a transfer learning setting even when a linear classifier is trained on-top of frozen feature extractors (Chapters 3, 4, and 6). These results indicate that our pretext tasks enable the deep model to learn domain invariant features that are highly transferable cross-domain. Hence, we can conclude that self-supervision is a robust way to improve label efficiency and a promising competitor to supervised transfer learning, where a model is learned with labeled examples and transferred afterwards. Along these lines, follow-up work is to incorporate pseudo-labeling or a similar semi-supervised learning algorithm to iteratively assign labels to high confidence predictions and utilize them during learning of downstream tasks. Likewise, tackling distributional shift (or domain adaptation) in a real-world setting is another open area of research.

**Research Question 3:** Does learning general purpose representations from unlabeled data improve performance on various recognition tasks? This question is investigated on sound recognition tasks.

We introduce contrastive learning for audio (COLA), a discriminative pretraining framework in Chapter 6 to address this question. The key ingredient of COLA is to contrast between representations extracted from the randomly sampled segments (i.e., anchors and positives) of the same audio clip. We demonstrate that to acquire a strong supervisory signal for training deep model unlike prior methods, COLA neither relies on augmentation strategies for positive sample generation nor require maintaining a memory bank of distractors. Our approach allows us to consider a large number of negatives for each positive pair in the objective function and avoid the need to carefully curate negative examples, unlike triplet-based approaches. We use a high-capacity network, EfficientNet-Bo [173] as the encoder and train it on a massive audio database known as Audioset [154] comprising millions of unlabeled audio clips. We evaluate COLA in a cross-domain transfer setting on numerous challenging audio understanding tasks of varying difficulty. Our results with a linear classifier trained on-top of a frozen network and end-to-end fine-tuning of pretrained model demonstrates that COLA learns unified representations useful for numerous downstream tasks beyond speech, such as keyword spotting, bird sounds, spoken language detection, speaker identification more. Specifically, our method improves performance over prior self-supervised pretext tasks and provides better generalization than training models on in-domain data directly. The COLA provides a simple yet powerful means to learn audio representations from unlabeled data. Looking into the future, it would be useful to explore if our approach is useful for other tasks, such as source separation, audio retrieval, and few-shot learning.

**Research Question 4:** How to perform self-supervised federated learning to utilize decentralized unlabeled data?

In Chapter 6 and 17, we demonstrate that self-supervision can be used in a federated
setting to exploit a massive amount of unlabeled data. Generally, it is assumed in federated learning that data on the device edge (i.e., a device holding the local data) or at client-side is annotated, or it can be labeled with a minimal effort. This assumption does not hold for many modalities, particularly for sensory data, and the edge devices have no direct interaction with human users, who can provide annotations. Self-supervised (and unsupervised) learning is a viable alternative to techniques that require semantic supervision. We propose scalogram-signal correspondence learning method, a general contrastive task focusing on contrasting whether a raw signal and a scalogram (i.e., a representation computed via wavelet transform) relate to each other or not. We note that other methods could be used to generate a corresponding view of the input in lieu of a wavelet, such as via fast Fourier transform. With a simulation of the federated environment on a range of tasks, we show, our approach learns representations of similar quality as those learned in a centralized setting, hence it could be effectively used to exploit unlabeled distributed data for model pretraining. There are many directions for research in this area; a promising way is to scale federated learning in the real world with thousands of clients participating in the learning process to recognize its true potential. Likewise, different auxiliary tasks (such as those described in Chapter 4) can be selected depending on the devices’ resources to optimize computational effort and quality of learned representations.

How to avoid catastrophic failure of deep neural networks on noisy inputs with learning-based approaches?

We study catastrophic failure in neural networks from the perspective of inconsistent inputs channels and missing modalities, e.g., due to sensor failures. We define a specific research question to investigate them in particular settings.

Research Question 5: Can a learnable channel remapping be used to handle inconsistent inputs? and what is a good strategy for tackling missing input modalities for a deep model at inference-time?

The channel reordering module (CHARM) introduced in Chapter 7 and 19 addresses the problem of tackling inconsistent input channels (e.g., in an electroencephalogram or EEG) that are permuted, missing, or noisy and hence can not be fed into neural networks, since they expect consistent channel ordering. CHARM is a novel differentiable model that learns to remap multi-channel input to canonical representations. Specifically, it learns per channel embedding independently to identify each channel’s location from its content and then uses an attention-mechanism to remap channels to a canonical ordered set. After the remapping, the channels are ordered consistently and can be further processed by a standard neural network, regardless of the actual variations in the input data. CHARM can be trained in an end-to-end manner while learning a task of interest without requiring any pre-processing on the user’s end. We also propose channel masking and shuffling augmentation to improve the model’s robustness further. CHARM is highly effective on challenging tasks involving EEG signals, such as seizure classification, detection of abnormal EEGs, and detection of artifacts (e.g., eye movement). We show that our approach is significantly more robust to missing and permuted channels than a standard model, even when 50% of the input channels are missing.

To solve the problem of missing sensory modalities at inference time, we develop an adversarial autoencoder-based model in Chapter 8 and 23 that learns to restore (or reconstruct)
features given other available modalities. Our approach can provide a better imputation compared to classical techniques and shows strong predictive performance on a multi-label context recognition task, even in cases when several sensory features are missing from a multi-modal input. An added benefit of our adversarial model is that it can be used to generate synthetic data in enormous amounts from a trained decoder. In particular, we show that our model can be conditioned on a multi-label class vector to create multi-sensory features. We evaluate the effectiveness of artificial data through visual fidelity analysis, learn a model on synthetic inputs, and estimate its performance on a real test set. For future work an important topic is to study adversarial robustness [299] of the sensing model to prevent the model from generating wrong predictions on artificially corrupted inputs.

How effective is a single deep neural network at learning shared representations for multiple related tasks?

We analyze the capability of multi-task learning in terms of personalized and adaptable models as well as creating a generalist model for solving many tasks. We outline two sub-research questions to study it in specific contexts.

**Research Question 6:** Is it possible to effectively leverage multi-task learning for model personalization and adaptation?

We introduce a subjects-as-tasks approach for personalizing deep neural networks using a multi-task learning formulation where a subject is treated as a task in Chapter 3 and [24]. A subject-independent global model for sensing tasks (e.g., physiological stress detection) may perform poorly due to large interpersonal differences. To take these disparities into account, we develop a multi-task model with a shared backbone network to learn general features across subjects and use small networks (with few layers) that focus on learning representations specific to subjects. We demonstrate the personalization efficacy of our method with two model architectures, one utilizing hand-crafted features with a feed-forward network and the other learning representations in an end-to-end manner with a temporal convolutional network. Likewise, we propose a method for unsupervised domain adaptation based on jointly learning to optimize a dual-stream neural network on source and target domain data. Precisely, one stream (or set of layers) operates on source domain data, where, labels are available to train a supervised model, while the other stream use target data with unsupervised objective function. We consider a cross-domain and cross-user transfer learning setting, where, source and target tasks share the same label set. Our model comprising an encoder, decoder, and a classifier learns to solve a source domain task using an encoder and a classifier with standard loss function (i.e., cross-entropy) and uses the encoder and decoder for the reconstruction of target domain inputs where ground-truth is not available. Our simple multi-task technique adapts shared representations using an auxiliary loss to improve generalization on a target task. Promising future work could be to investigate if self-supervised pretext tasks help in domain adaptation, particularly after model deployment.

**Research Question 7:** To what degree can a unified deep neural network learn to solve multiple tasks using multi-modal sensory data?
This question focuses on a broader and challenging problem in machine learning of creating a single generalist neural network capable of solving several tasks while exploiting shared structure among them. Within this theme, we focus on developing a model for sensory data, such as accelerometer, gyroscope, photoplethysmogram, and electroencephalography. We devise a unified model in Chapter 10, a deep temporal convolutional neural network that can learn to solve many sensing tasks from cross-domain (i.e., different input modalities, sensors, users, data collection protocols and tasks) multi-modal data both in supervised and self-supervised ways. Our lightweight and simple model learns general-purpose representations that achieve performance similar as individual task-specific models while significantly reducing the number of parameters to learn and effort to design a separate model for each task. We extend contrastive predictive coding to a multi-task setting for self-supervision of the model. We highlight its generality by pretraining a model simultaneously across datasets with different modalities and show that unified model learns useful features without task-specific labels. Our model achieves significantly better generalization in low-data regimes through using a pretrained model as initialization. In particular, even with five to 10 labeled instances the unified model performs better than supervised and single-task self-supervised models. Furthermore, our approach achieves better generalization in transfer learning settings on out-of-domain data than models trained using in-domain data. With the development of this unified model, we barely scratch the surface of what is truly possible in learning one model for multiple tasks. Given these results, we can answer the defined question affirmatively that a single model can be used to solve a variety of sensing tasks. Future work can look into designing better architectures using a mixture-of-experts or an attention-mechanism with one encoder per modality instead of task-specific encoders and reusing them across tasks. Likewise, exploring different auxiliary tasks from sense and learn framework for each modality or task is also an exciting work that may help improve the representation quality.

Final Remarks

The work in this thesis is based on the idea of incorporating self-learning capabilities in the IoT and other devices, which can ultimately unleash their true potential to continually sense, learn, and adapt in a real-world setting. The ubiquity of sensor-rich IoT devices in our daily lives generates a continuous stream of high-quality data that can be exploited to build intelligent systems to solve challenging problems. This research theme has led us to important discoveries in proposing novel self-supervised tasks, learning from decentralized data, developing small-scale and powerful temporal convolutional architectures, methods for improving the robustness of neural network-based models, enhancing generalization in the low-data regimes, and beyond towards exploring a unified model for multiple tasks.

Our devised self-supervised methods instill in deep neural networks the ability to learn general-purpose representations without requiring semantic annotations from humans, which is of a high value for IoT devices as labeling is expensive, time-consuming, and requires domain expertise in several cases. Similarly, our broad range of auxiliary tasks provides an opportunity to choose a suitable one for training deep neural network based on modalities, system device resources, and performance on downstream tasks. Learning on the distributed devices is challenging due to the heterogeneity and unlabeled nature of the data; in such cases,
our pretext tasks can be effectively used to learn deep models without aggregating privacy-sensitive data in a centralized repository. The availability of better tools and frameworks for federated learning as well as large-scale sensory datasets that simulate real-world settings can further demonstrate the effectiveness of self-supervision. Moreover, the predictive models are prone to failure when encountering missing modalities or inconsistent input channels. We developed methods to improve robustness and enable graceful degradation of predictive performance that is of central importance to smart devices as sensors can break and input can be corrupted, e.g., due to system heterogeneity. Towards incorporating useful inductive biases and leveraging the shared structure of the tasks, we also focused on learning a single model for multiple tasks. A general-purpose audio model, potentially running on a resource-constrained device, can effectively solve various sound recognition tasks. We developed a self-supervised method to create a single model that performs well on a range of audio classification tasks. Our method performed significantly better than prior hand-crafted auxiliary tasks for audio models and improved the performance over supervised models through fine-tuning. Likewise, we also proposed a unified neural network-based model for sensing tasks that can utilize cross-domain and multi-modal data for learning, moving from specialists to generalist models. With a unified model for many tasks we demonstrated that a similar performance can be achieved to in-domain models. Likewise, we show that single model can be trained with unlabeled data and it improves the performance in low-data regimes as compared to training from scratch; while being highly transferable to out-of-domain tasks.

Over the course of this thesis’s work, our contributions led us to several general insights about learning deep models for sensing tasks and corresponding shortcomings of existing approaches and missing components. We found self-supervised learning to be highly effective for pretraining as compared to auto-encoding approaches and widely beneficial in low-data regimes. Self-supervision not just reduced the number of parameters to learn but provided us the liberty in designing better encoders as no symmetric decoders were needed. Likewise, we think that the success of transformation recognition as an auxiliary task may be due to its capability of making a network become invariant to various perturbations of the input, which helps incorporate useful inductive biases. We also note that the developed variants of signal transformation objective as k-way classification and a set of binary classification tasks solved in a multi-task manner leads to different optimization of model’s parameters. Specifically, the latter provides a way to the network to learn transformation invariant and non-invariant features with distinct layers while the former makes the network architecture simpler but does not provide a distinction between learned representations. We note that here the design choice can be primarily driven based on domain-knowledge or performance on the downstream task. We also did not explore combining transformation and contrastive learning, where, representational similarity of original and augmented (transformed) samples is maximized while minimizing it for the rest of the examples. We think it could be a logical extension of our proposed task.

We believe, contrastive objectives (or other tasks that do not require reconstruction) are better suited if we are solely interested in learning representations and do not want to generate synthetic data, for which generative approaches are the natural choice. Likewise, we think a fusion of self-training (i.e., noisy student and teacher paradigm) with self-supervision could be very effective to label the data automatically and learn from it. We notice that the majority of the sensory datasets in pervasive sensing are relatively smaller in size compared to datasets in...
other communities e.g., natural language processing. Having a large-scale unlabeled dataset with millions of examples could truly unlock the potential of self-supervised learning. To avoid this problem, many small datasets could be combined but it limits the evaluation in transfer learning setting. Similarly, as we designed many auxiliary tasks, there are several open questions that can be studied to improve understanding of self-supervision and deep learning in general. We suggest the crucial ones here: a) why do certain tasks work better than others in terms of performance on downstream applications?, b) does the representations differ as compared to learning with semantic labels?, c) how to combine different pretext tasks in a multi-task learning paradigm to determine if it is useful as compared to training a model with a single auxiliary task? d) how robust can self-supervised representations be for learning with imbalanced downstream data? and e) how can we establish similarity between pretext and downstream tasks for avoiding negative transfer?

The availability of stable frameworks for federated learning simulation improved over the years. We notice that to effectively utilize federated learning in embedded or pervasive intelligence and to address corresponding issues, benchmark sensory datasets with a large number of users are needed. Even accessibility of publicly available unlabeled data with clear distinction of clients can widen the horizon of problems that can be tackled with federated unsupervised learning. We show that general representations of audio can be learned that are useful for many tasks, raising a question if a multi-modal network can be used to learn simultaneously from synchronized modalities in a same manner. Moreover, learning unified model with cross-domain data provided valuable empirical results that neural networks can effectively exploit shared structures of different input modalities (e.g., accelerometer, electroencephalogram, and photoplethysmogram). We think analysis of weights and features from the earlier layers of such a model can yield valuable information about network capturing high-level patterns that are similar to diverse signals. Such an insight can guide the development of better architectural priors or inductive biases that can ultimately be used for continual learning from sensory streams. For personalization, we demonstrated that a multi-task model with hard-parameter sharing where subjects are treated as tasks provides considerable performance improvements. To improve it further, a network that can adaptively select layers via conditioning on personal attributes (such as, gender and age-group) would be more appropriate in feature reuse. Moreover, in remapping inconsistent input channels to a canonical representation our approach exhibited remarkable performance in recognizing electroencephalogram channels from its content and improving robustness to severely shuffled and missing channels. However, we notice that on clean input a standard model performed better as it has direct (implicit) access to channel ids. We think to improve the remapping module it can be augmented with learnable positional embedding and a deep attention network discovered using architecture search.

Finally, there are other important areas we did not explore in the thesis, which can be a subject of future research. Among them, adversarial examples pose a major challenge for deep models. The safe execution of a model to generate correct predictions is crucial for any system equipped with a predictive model and that relies on it to make a decision. The area of adversarial robustness concerns the development of approaches to make neural networks invariant to imperceptible perturbations of the input \[299\]. Therefore, methods that can improve the model’s resilience to adversarial inputs are of high value and can complement our methods on robustness against noise. Likewise, avoiding learning via shortcuts \[97\] in deep neural networks
is another key area that can improve our understanding of failure modes and interpretability to
develop better self-supervised algorithms, architectural priors and boost the transferability of
models trained in a controlled environment to a real-world setting. Lastly, translating models’
predictions to decisions is also an open area of research. Currently, it is left up to the system’s
designer to decide on how to act on the predictions. Here, reinforcement learning can play
a role as it provides a set of methods for solving sequential decision-making problems. For
instance, to design better interventions to improve the user’s well-being based on recognized
behavioral patterns. In particular, offline reinforcement learning \cite{300} is a suitable candidate
to bridge the gap from prediction to decision and takes us closer to the truly intelligent IoT
devices for pervasive sensing that can act on the detected events and learn from it.
Bibliography


[288] Zhen Qin, Yicheng Cheng, Zhe Zhao, Zhe Chen, Donald Metzler, and Jingzheng Qin. Multitask mixture of sequential experts for user activity streams. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), New York, NY, USA, 2020. Association for Computing Machinery.


[293] Olivier J Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, SM Es- 
lami, and Aaron van den Oord. Data-efficient image recognition with contrastive pre-
2020.

[294] Joel Shor, A. Jansen, Ronnie Maor, Oran Lang, Omry Tuval, Felix de Chau-
mont Quitry, M. Tagliasacchi, Ira Shavitt, D. Emanuel, and Yinnon A. Haviv. Towards 
learning a universal non-semantic representation of speech. *ArXiv*, abs/2002.12764, 
2020.

problem-agnostic speech representations from multiple self-supervised tasks. In IN-
TERSPEECH, 2019.

[296] Jean-Baptiste Alayrac, Adrià Recasens, Rosalia Schneider, Relja Arandjelović, Jason 
Rampuram, Jeffrey De Fauw, Lucas Smaira, Sander Dieleman, and Andrew Zisser-

models from self-supervised synchronization. In *Advances in Neural Information Pro-
cessing Systems*, 2018.

[298] Amila Silva, Shanika Karunasekera, Christopher Leckie, and Ling Luo. Meteor: Learn-
ing memory and time efficient representations from multi-modal data streams. In Pro-
cedings of the 29th ACM International Conference on Information & Knowledge Manage-
ment, 2020.

[299] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing 

[300] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforce-
ment learning: Tutorial, review, and perspectives on open problems. *arXiv preprint 
Acknowledgements

First and foremost, I would like to thank all my teachers for their support, guidance, understanding, and encouragement, as none of this work would have been possible otherwise. I am immensely thankful to them for teaching important lessons and inspiring me to have a grand vision. I am tremendously grateful to Johan Lukkien and Tanir Ozcelebi for an opportunity to pursue Ph.D. under their supervision and for their continuous mentorship, support, and above all, freedom to pursue my interest and explore different research directions. Johan, thank you for the thought-provoking discussions, advice, motivation, and supporting me in the ups and downs of navigating the Ph.D. Tanir, thank you for your unconditional support, patience, and fostering an environment where I was able to freely pursue research. I have also been fortunate to have great mentors throughout graduate life. I want to express immense gratitude to Stojan Trajanovski, Maurice van Keulen, Adrienne Heinrich, Qasim Pasta, Faraz Zaidi, Khalid Khan, and Rashid Saleem for believing in me and inspiring me to pursue a scientific career and work towards a Ph.D. I would like to thank my thesis committee - Cecilia Mascolo, Fahim Kawsar, Flora Salim, Mykola Pechenizkiy, and Mathias Funk for taking the time to give me valuable feedback.

Being part of the IRIS (formerly SAN) group was a great and humbling experience. Many thanks to all my colleagues Jeroen, Hamid, Reza, Nan, Sachin, Bram, Luis, Tianyu, Marijn, and Geert for countless funny moments during coffee breaks, work-related discussions, social events outside work and end-of-day meetings. I am also very grateful to Anjolein Gouma and Jolande Matthijssse for all their support in administrative tasks. In particular, I am immensely thankful to Richard Verhoeven for all his support in terms of computing resources, server management, and help with demonstrator setup for the SCOTT project. I am also thankful to all the partners within the SCOTT H2020 project. I would also like to thank Nirvana Meratnia for her valuable support and help in creating a pleasant work environment during the pandemic for working from home. I also have had the chance to supervise and work with several talented MSc students, including Bram, Ye, Songwa, Sandhiya, and Vasilis. It was great working with you, and I learned a lot! I am also glad to have had the opportunity to collaborate with outstanding researchers and faculty outside of the Eindhoven University of Technology: Shkurta Gashi, Shohreh Deldari, Flora D. Salim, Irina Stipanovic, Jan van Erp, Silvia Santini, and Zaharah Bukhsh.

I am immensely grateful to have had the chance to be a research intern at Google Research twice during the Ph.D. I gained valuable research and software engineering experience
within the Sense team in Google Cerebra under the supervision of Victor Ungureanu and Beat Gfeller. Thank you, Victor, and Beat for being amazing hosts, providing excellent mentorship, and helping me in navigating the Google codebase. I would also like to thank Hassan Rom, Marco Tagliasacchi, Felix de Chaumont Quitry, Dominik Roblek, Matt Sharifi, and Jeremiah Harmsen for all their support. My mentors and hosts for an internship within the Google Brain team Neil Zeghidour, David Grainger, and Olivier Pietquin have my immense gratitude for their continuous support in making a remote internship a wonderful experience and remarkably a productive one. Neil, thank you very much for the guidance, advice, patience, being extremely approachable, and stimulating discussions and above all for being an amazing mentor. It was a pleasure working with you, and I learned a lot within a short period of time from you. David, thank you for being a great mentor and collaborator; your feedback and advice have helped me grow tremendously. I also like to thank Olivier Teboul and other members of the Brain team in Paris. I am also very thankful to Google Cloud for their support in computing resources for my research.

I am very fortunate to have great friends and want to thank them for always being with me through thick and thin and in celebrating my achievements. I am particularly grateful to Inam, Nouman, Mudasir, Mutlib, and Basit for their encouragement, support, distractions, fun conversations, and being part of my life throughout these years!

Finally, I am immensely thankful to my parents for all their love, sacrifices, and countless efforts in providing me opportunities for a better future. Many thanks to my siblings for their unconditional support, love, and taking care of everything in my absence. I also like to thank Faizan and Faiza for their guidance, encouragement, wonderful trips, barbecues, and for the good times over the years in the Netherlands. A special word of gratitude to my wife, Zaharah, for enriching my life, for her patience, enduring my long working hours, countless discussions on research, always being there for me, and being emotional support in making this journey wonderful. Zaharah, thank you!
Aaqib Saeed was born in Hyderabad, Pakistan. He obtained his bachelor’s degree in Computer Science (cum laude) from PAF-Karachi Institute of Economics and Technology, Karachi, Pakistan, in December 2014. Thereafter, in 2015, he started his M.Sc. study in Computer Science at the University of Twente, Enschede, the Netherlands. During his master’s studies, he participated in the Honors Program, and he did research internships at TNO and Philips Research. In 2017, he obtained his master’s degree (cum laude), specializing in Data Science and Smart Services. Immediately after that, he started his Ph.D. in the Department of Mathematics and Computer Science at the Eindhoven University of Technology, Eindhoven, the Netherlands, under the supervision of prof. dr. J.J. Lukkien and dr. T. Ozcelebi. His Ph.D. research was funded by the H2020 SCOTT project. During his Ph.D. studies, he did two research internships in Google Research. The research focus of the Ph.D. is on learning general-purpose and robust sensory representations with minimal supervision. The results are presented in this dissertation.