Time-series data augmentation with evolutionary search methods

Li, Y.

Award date:
2019

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Time-series Data Augmentation with Evolutionary Search Methods

Master Thesis

Ye Li

Supervisors:
Dr.Tanir Ozcelebi
Aaqib Saeed

Complete version

Eindhoven, October 2019

Abstract

Data augmentation is an efficient method to produce synthetic samples when labeled samples are limited in the neural network classification task. However, current data augmentation methods for time-series are designed manually and the transformations are dataset-specific. In this project, we propose a procedure to automatically search the optimal combination of transformations (i.e. policy) for time-series.
In our implementation, we use evolutionary search algorithms to find the augmentation policy, which can promote the neural network to achieve better classification performance on different datasets. The policy search is designed in continuous search space. An augmentation policy consists of many sub-policies and a sub-policy is made up of operations, corresponding parameters and the probability of application. Each operation is a time-series transformation function such as scaling, jitter, time-warp and so on. We propose several evaluation strategies on five datasets to validate our method performance.
We enrich the transformations for time-series data. Empirical results prove our method obtains a significant improvement in classification Cohen's-kappa score, F1-score, precision and recall. Our approach implements more abundant transformations for time-series data than current research. Our augmentation policies are transferable between domain-specific datasets. Our method can implement effective classification when there are limited available instances.

Preface

This thesis is the result of the master graduation project for Mathematics and Computer Science at Eindhoven University of Technology. The research of this project is performed within the System Architecture and Networking group of the TU/e.

First of all, I appreciate my supervisor Dr.Tanir Ozcelebi for offering this interesting project, during the project, he provided effective supervision to ensure that the project is carried out in an orderly manner. Also, I would like to take this opportunity to thank Aaqib Saeed, who followed closely on my project and gave me very in-time valuable practical and improvement suggestions. I learned a variety of knowledge from Aaqib at every stage of the project, and he always encouraged me when I lost confidence.

I would like to thank my family for their unconditional financial and spiritual support during my master's project. At the same time, I would like to thank my friends for communication and encouragement during the whole process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we first introduce the motivation for this project. Next, we briefly summarize the goals and challenges in time-series data augmentation. In accordance with the challenges, we describe our approach and results as well. In the final section of this chapter, we give the outline of this thesis.

1.1 Motivation

Recently, neural networks have been widely applied in various scenarios adopted for time-series data and achieve near-human accuracy performances, in tasks such as speech recognition[10] and human activity recognition (HAR)[22]. The advantages of deep neural networks in the classification task can be fully demonstrated when enough labeled samples are available. The time-series data requires long-term detection and recording, therefore, time-series data usually need to be collected by professional institutions or special experiments. Meanwhile, there are few publicly available time-series datasets due to privacy concerns. Therefore, it is difficult to have access to valid labeled data for neural networks training generally. The different sensor devices also may lead to the different characters of data, even with the same purpose of collection such as data collected for HAR. These data cannot be used together to train one neural network. How can we improve classification performance when faced with insufficient data in training a neural network? One viable solution is data augmentation.
Data augmentation has achieved significant improvement of performance in supervised deep neural networks, especially on the image data. But little work has been done on augmenting data for time-series classification. At present, the transformations of time-series data augmentation are all implemented manually, and associated parameters are all fixed in advance. Cubuk et al.[5] have proposed a method that realizes automatically learning data augmentation policies by reinforcement learning for image datasets. Therefore, this project is motivated by their research. We will explore evolutionary search methods for automatically finding time-series data augmentation policies.

1.2 Goals and challenges

The overall goal for this thesis is to develop an automatic approach of searching augmentation methods for the time-series data, and the augmentation method can be used to improve the classification performance of the deep model.
The overall challenge is in constructing a good set of augmentation transformations and associated parameters, and finding the best combination of transformations and parameters. Augmentation functions for time-series data are various, and each function is related to at least one parameter that needs to be tuned. Therefore the number of possible augmentation function combinations is enormous. Meanwhile, time-series data is sensitive to the transformation. How to control the

generated data owning new features without changing the label of the original data is another problem. There is very limited literature on data augmentation applied to time-series data, with a small number of transformations. Therefore, to enrich the augmentation method, we propose more transformation functions for time-series data. How to find the optimal combination of transformations and select their associated parameters have become another problem that needs to be solved.

1.3 Approach and Results

1.3.1 Approach

After a deep understanding of the method of Cubuk et al., we propose our method. The specific procedure is described roughly here corresponding to the changes:

- **Establish policy search space**, which is formulated by a set of augmentation functions and associated parameters. Augmentation function and associated parameters constitute augmentation policy. To enrich the search space, we propose six more augmentation functions (i.e. operation) in addition to adopting the current proposed six operations.
- **Establish neural networks** is an important part of the whole procedure because both obtaining the updating direction of the search algorithm and evaluating the final performance of data augmentation are based on the results of the neural network classification.
- **Evolutionary search strategy** is the optimization techniques we used to search augmentation policy, including random search, covariance matrix adaptation evolution strategy (CMA-ES) and particle swarm optimization (PSO).

1.3.2 Results

We implement a variety of experiments to evaluate our method. We demonstrate that our approach significantly improves deep model performance on classification. The network trained with augmentation policy achieves better F1-score by 1.5 points than the baseline model. The obtained augmentation policy can be highly transferred across the similar datasets and improve the classification performance when few samples available.

1.4 Thesis Outline

The thesis is organized as follows:

- In chapter 2 (Background and Related work) we first discuss the data augmentation, which is the foundation and the focus of this project. Then we review the classification using neural networks, which creates the requirement of data augmentation. Finally, we introduce the evolutionary search algorithm, which is the main optimization algorithm we used in the entire project.
- In chapter 3 (Problem Description and Approach) we analyse the limitations of the current implementation of time-series data augmentation firstly. Secondly, we propose the framework of our data augmentation method. We also introduce some specific definitions and implementation methods in the approach, and evaluation methods used to show the performance.
- In chapter 4 (Experiments) we describe the datasets used in the project and basic preprocessing on data. Then, we describe the settings and structures of the neural network model. Next, we conduct experiments including using different datasets, using the different evolutionary search algorithms, simulating the situation of a few instances. We present and discuss the results to prove the validation of our data augmentation method.

- In chapter 5 (Conclusions) we conclude the academic contributions of the thesis. Furthermore, we discuss future work.

Chapter 2

Background and Related work

In this chapter, we will introduce some basic knowledge related to our project and some state-of-the-art research in related fields, which mainly includes data augmentation, deep learning for time-series classification, and evolutionary search algorithm.

2.1 Data Augmentation

Deep learning has advantages in many fields such as natural language processing, speech recognition, and image analysis. However, quantity and diversity of data have a significant influence on the result of deep learning. When there is not enough labeled data, it is easy to cause over-fitting for the model in the training process. Over-fitting means that the model learned excessive features from data including noises, which leads to a poor generalization to other data. There are different approaches to solve this problem, in addition to general regularization techniques, dropout[26] and batch normalization[11], data augmentation is another method to solve over-fitting. Data augmentation is a process that generates new data containing similar features with original data.

2.1.1 Data Augmentation for Time Series

Guennec et al.[12] introduced some time series-data augmentation methods which could be used in training convolutional neural networks. They introduced three methods: window slicing, window warping, and dataset mixing. Window slicing comes from [6], which split the same label original time-series data into many slices with the same segmentation size. The segmentation size is the parameter that needs adjusting. Window warping method randomly warps time intervals of a slice time-series data. Figure 2.1 shows the operation of window warping. In addition to considering the size of the slice, the ratio of warping also needs to be adjusted, which ranges from $\frac{1}{2}$ to 2. Data mixing is the method using various datasets to pre-train model to provide better performance. In the end, the authors show data augmentation methods help improve classification performance.

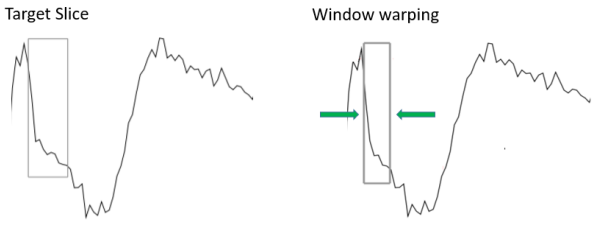


Figure 2.1: Example of the window warping data augmentation technique

In later research, Um et al.[28] proposed seven data augmentation methods for wearable sensor

data of Parkinson's disease. Jittering (Jitter) is a method for randomly adding noise to the original data. Scaling (Scale) and magnitude-warping (MagW) both change the magnitude of the input. The difference is that the former method multiplied by a random factor, the latter method convolved with a smoothly-varying noise. Rotation (Rot) method is used to change the sensor placement. Permutation (Perm) is slicing a window into different same-length segments and randomly swap the segments. Time-warping (TimeW) distorts the time intervals between samples to warp time-series. Cropping (Crop) is clipping a part of the original time-series. The authors also tried to combine different operations on one sample, and the final results concluded by the authors also show the combination of different operations can reach a better performance.
Ramponi et al. used the conditional generative adversarial network (CGAN) to augment unbalanced dataset in time-series classification problem [18]. They generate synthetic time-series data automatically. However, the synthetic data generated by GAN will be as similar to the current training data as possible, therefore, new synthetic data can not be transferable to other datasets.

2.1.2 Data Augmentation for Image Data

Compared to time series data augmentation, there are more researches on image data augmentation. Image data augmentation methods pay more attention to generating image data through a linear transformation like rotation, scaling, shearing and elastic deformation[24, 3, 25]. AutoAugment[4] was proposed by Cubuk et al., in which they automate the process of finding the best data augmentation policy for a target dataset. Augmentation policy consists of several choices and orders of augmentation operations, where the operation is made up of an image transformation function such as invert, rotate, or shear, the probability of applying the operation and the magnitude of the operation. They use reinforcement learning as the search algorithm to find the best policy such that training a neural network yields the highest validation accuracy. According to the results of [4], they achieved state-of-the-art accuracy on multiple image datasets. They stated results can be further improved if a better search algorithm is used[17].
Subsequently, Mingyang Geng et al. improved the performance of AutoAugment in ARS-Aug[9]. Their contribution lay in they covert the discrete search problem of AutoAugmnet to continuous, which could improve the searching performance and maintain the diversities of policies. In AutoAugment, they discretize the range of magnitude into 10 values and probability into 11 values so that they use a discrete search algorithm. However, in ARS-Aug they use a new method to arrange magnitude values and probability replacing discretization. In ARS-Aug, they use the random search method to find the best policy, and policies express more accurate states because of the continuous search space. Compared to AutoAugment, ARS-Aug can achieve better performance.

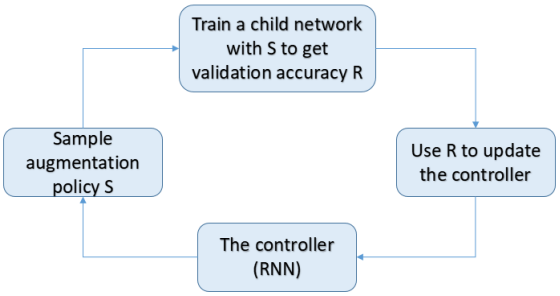


Figure 2.2: Framework of Evolved AutoAugment[5]

Cubuk et al.[5] proposed an evolved AutoAugment, in which they add a controller trained by reward signals. Figure 2.2 shows the framework of evolved AutoAugment. They still use the same definition of policy and same search algorithm. Compared with the results of ARS-Aug, they did not exceed the performance of ARS-Aug on the same dataset.

2.2 Deep Learning for Time Series Classification

We will use time-series data and classification results to evaluate the performance of data augmentation in our research. therefore, we first introduce time-series data classification. There are a lot of classification methods for time-series data. The requirement of data augmentation is for solving a lack of training data and over-fitting in deep learning. In this section, we only focus on deep learning for time-series classification.

2.2.1 Time Series Classification

Hassan Ismail Fawaz et al. gave the definition of time-series classification in their review work[8]. For a dataset D = $\{(x_1, y_1),(x_2, y_2),...(x_N, y_N)\}$, x_i denotes time-series, y_i presents its corresponding label vector. The label vector y_i is the vector to represent K classes. The goal of time series classification is training a classifier on a dataset D which can map the space of unknown inputs to a probability distribution over the labels.

2.2.2 Deep Neural Network

A deep neural network (DNN) is a more complicated artificial neural network (ANN)[30]. In general, there are not many levels of neurons in ANN, while in DNN, there are a lot of layers. Nowadays, deep neural networks can reach 5 to 1000 levels. Although there exist various types of DNNs, we mainly introduce two types of DNN architectures: Feedforward Neural Network (FNN), Convolutional Neural Network (CNN), which are related to our project.

Feedforward Neural Networks Figure 2.3 shows the architecture of an FNN model for time-series data. Generally, except for the input layer and output layer, there is at least one hidden layer in the FNN model. Their layers are fully connected (any neuron is connected to all neurons in the next layer). For a three layers FNN model (contains one hidden layer), it could be concluded as the following function:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))) \tag{2.1}$$

where W^1, b^1 and W^2, b^2 represents the weight and the bias of the hidden layer and output layer respectively, s means the sigmoid function and G denote the softmax function. In practical applications, the input data with T time length and S sensor channels will be first flattened into a (T × S)-dimensional vector, then pass to the hidden layers. Finally, the model could give the prediction probability among N classes through the final softmax.

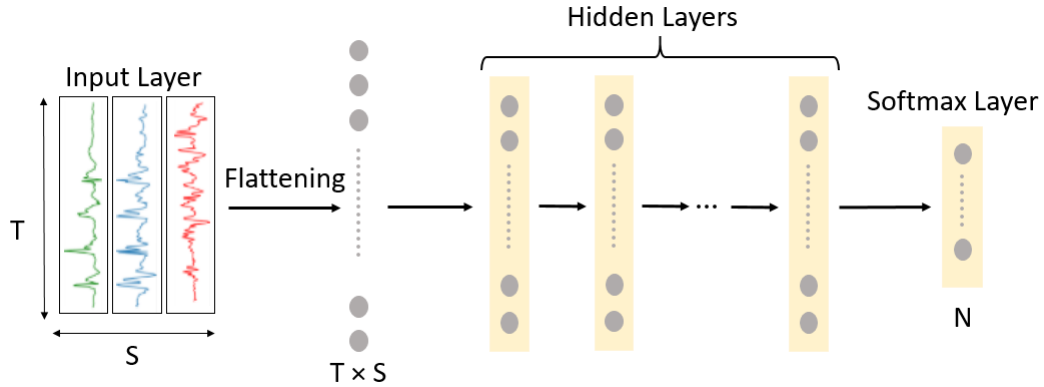


Figure 2.3: Architecture of a MLP model for time series, which is adapted from [13]

Convolutional Neural Networks CNN has given good results in image and speech recognition. Researchers are trying to apply CNN on the time-series data. Figure 2.4 shows the architecture of a CNN model. Convolution is sliding a filter $(f_T^{(i)}, f_S^{(i)})$ over the time-series. The convolution layer is usually followed by the pooling layer to speed up the training process on a large dataset. The most popular pooling operations are max or average pooling, which operates on a patch of size $(p_T^{(i)}, p_S^{(i)})$ of the input to downsample it. In addition to the combination of convolution and pooling layer, sometimes people also adopt normalization layer and dropout layer to avoid over-fitting in the process of training. After that, a fully-connected layer (also called dense layer) and softmax layer are connected to perform a fusion of the information extracted from the input and finally output a probability distribution over the class variables.

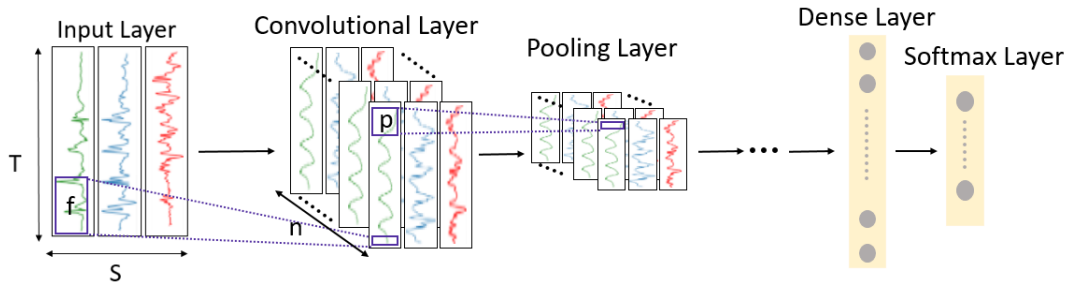


Figure 2.4: Architecture of a CNN model for time series. T, S, and n_k represent the length of the input data, the number of input channels and the number of convolutional kernels of the ith layer, respectively, which is adapted from [13]

2.3 Evolutionary Search Algorithm

The evolutionary search algorithm is a kind of search technique inspired by biological evolution. It simulates the collective learning progress of a group of individuals, where each individual represents a point in the search space for a specific problem. Figure 2.5 shows the scheme of evolutionary search algorithm. It starts with any initial population of individuals, and evaluates the fitness of each individual in that population. The algorithm selects the best-fit individual as parents, through a process of mutation and recombination to generate offspring. Then evaluates the fitness of the new individual, gradually, the population evolves to better and better regions of the search space.

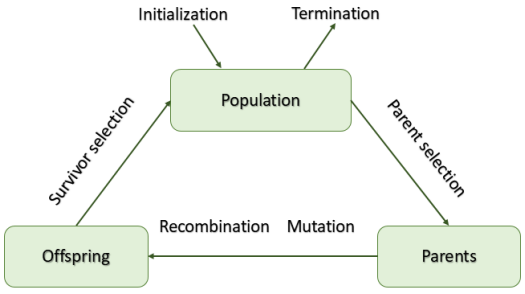


Figure 2.5: Scheme of Evolutionary algorithm

Due to their inherent parallelism, the evolutionary search algorithms are very suitable for large parallel machines. Furthermore, it uses the natural evolution mechanism to express complex phenomena and can solve very difficult problems quickly and reliably. Evolutionary search algorithms

have been used more and more widely in fields such as optimization, machine learning, and parallel processing.

2.3.1 Covariance Matrix Adaptation Evolution Strategy

Tim Salimans et al. proposed[23] highly parallel evolution strategy (ES) that completed the model training in a very short time. ES is a gradient-free stochastic optimization algorithm with good parallel scalability, invariance under some transformations, and sufficient theoretical analysis. Its main progress steps are similar to the evolutionary search algorithm. The covariance matrix adaptation evolution strategy (CMA) is a well-known evolution strategy. The main idea of CMA is to adjust the covariance matrix of normal distribution $N(m_t, \sigma_t^2 C_t)$ to deal with the dependence and scaling between variables. The normal distribution contains three parameters:

- m_t determines the central location of the distribution. In the algorithm, it determines the search area.
- σ_t^2 is the parameter of step size, which determines the global variance of the distribution. In the algorithms, it determines the size and intensity of the search area.
- C_t determines the shape of the distribution. In the algorithms, it determines the dependence between variables and the relative scale between search directions.

The core of the ES algorithm design is to adjust these parameters to achieve the best search effect. The adjustment of these parameters has a very important influence on the convergence rate of the ES algorithm. Generally, it adjusts parameters that make the probability of producing a good solution gradually increase (the probability of searching along with a good search direction increases). CMA-ES repeatedly iterates the following steps in search:

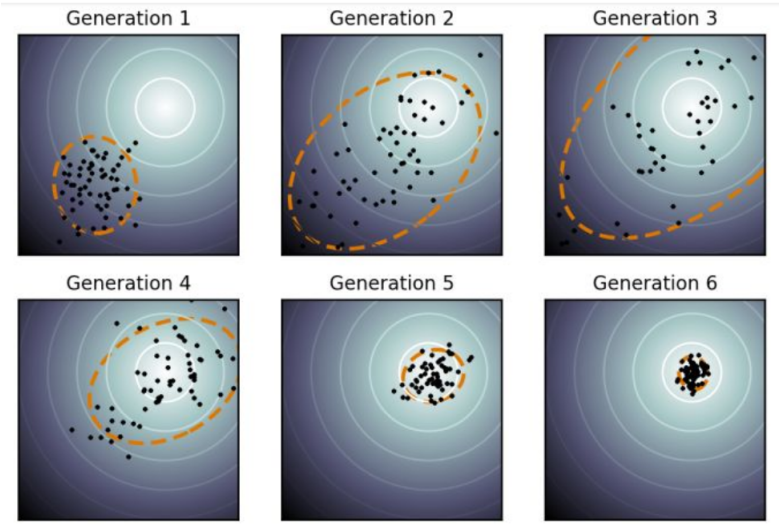


Figure 2.6: CMA-ES search process [21]

1. Sampling to produce new solutions. Each iteration produces λ solutions

$$x_i = m_t + \sigma_t y_i, y_i \sim N(0, C_t) \tag{2.2}$$

where y_i is a search direction.
2. Computing objective function $f(x_i)$ of the newly generated solutions, and sort them.

$$f(x_{1:\lambda}) \leq f(x_{2:\lambda}) \leq \cdots \leq f(x_{\lambda:\lambda}) \tag{2.3}$$

where the subscript means the ith place in these samples.
3. Updating distribution parameter
(1) Update of m_t comes from the weighted maximum likelihood estimation of λ solutions, the equation is shown in the following:

$$m_{t+1} = \sum_{i=1}^{\mu} w_i x_{i:\lambda} = m_t + \sum_{i=1}^{\mu} w_i (x_{i:\lambda} - m_t) \tag{2.4}$$

where w_i is the parameter of weight
(2) C_t's update principle is to increase the variance along with the successful search direction, that is, to increase the probability of sampling along these directions. The equation is:

$$C_{t+1} = (1 - c_1 - c_\mu)C_t + c_1 p_{t+1} p_{t+1}^T + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T \tag{2.5}$$

where p_t is evolution path, c_1, c_μ are learning rate, and $c_1 \approx \frac{2}{n^2}$, $c_\mu \approx \frac{\mu_w}{n^2}$. Factor $\mu_w = \frac{1}{\sum_{i=1}^{n} w_i^2}$
(3) By default, CMA-ES uses cumulative step size adaptation (CSA), which is currently the most successful, most used step size adjustment. The theory of CSA is the direction of successive searches should be conjugated. If there is a positive correlation between successive search directions (the included angle is less than pi/2), then the step size is too small and should be increased. If there is a negative correlation between successive search directions, then the step size is too large and should be reduced. CMA-ES repeats the above search steps until finding the optimal solution.

2.3.2 Particle Swarm Optimization (PSO)

PSO algorithm was proposed by Dr. Eberhart and Dr. Kennedy in 1995[7], which is designed by simulating the predation behaviour of birds. There are different food sources in the area, the task of the bird flock is to find the largest food source.
The goal of PSO is to enable all particles to find the optimal solution in multi-dimensional hypervolume. First, all particles in space are assigned initial random positions and velocities. Then, the position of each particle updates according to the velocity of each particle, the known global best position in the problem space and the known best position of each particle. As computation progresses, particles aggregate around one or more optima in the search space. When achieves the maximum number of iterations or the global bestl position satisfies the minimum boundary, algorithm break iteration. The benefit of the algorithm design lies in that it retains the global best position and the best position of the particles.

Chapter 3

Problem Description and Approach

3.1 Problem Statement

Although synthetic data generation is shown to significantly improve the performance of supervised deep learning, we have introduced in the previous chapter. Currently, there is little research on time-series data augmentation for classification and many of them rely primarily on simplex method, that is window warping. Um et al.[28] proposed various augmentation methods for Parkinson's disease monitoring data, and they concluded combining the different augmentation transformations could reach the best performance. However, facing more than a dozen methods, each method may with one or more parameters that need tuning, it is conceivable that this is cumbersome work to find out manually the optimal combination of transformations that obtain the best classification performance.
Ramponi et al. [18] used GAN to augment unbalanced data sets. However, the synthetic data generated by GAN cannot be transferred to other datasets. For the above reasons, the objectives of this project are

- Establish a system that could find the optimal combination of transformations with the required parameters for time series classification automatically.
- Evaluate the performance of the combination of transformations with chosen parameters (i.e. policies).
- Verify policies can be transferred to other related datasets.

3.2 Approach

Figure 3.1 shows an overview of our to implement the data augmentation. There are 8 steps in this framework. Policy in this framework is the method used on time series to generate synthetic data, which is combined with 5 sub-policies in this project. There are two operations in one sub-policy, each operation is associated with two parameters numerical parameters: probability of applying the operation, and the magnitude of the operation. These parameters constitute the policy search space. The individual steps contain the following actions.

1. Random choice of N policies as a start (given the predetermined policy structure).
2. Building the neural network model and using these chosen N policies on the training set to train a model.

3. Using the trained model with data augmentation to predict the labels of validation set samples and evaluating the performance.

4. Sorting the classification results and using them as the fitness to update the evolution direction.

5. Normalizing the outputs of the evolutionary algorithm by sigmoid function and generating the corresponding child augmentation policies. Then repeating the previous steps until the evolutionary search strategy achieves the ending condition. The ending condition is preset by the user.

6. Terminating search when it satisfies the ending conditions, and obtaining the top-10 best policies according to the F1-score.

7. Utilizing the top-10 policies to re-train the same model.

8. In the final step, evaluating the performance of data augmentation through different evaluation strategies.

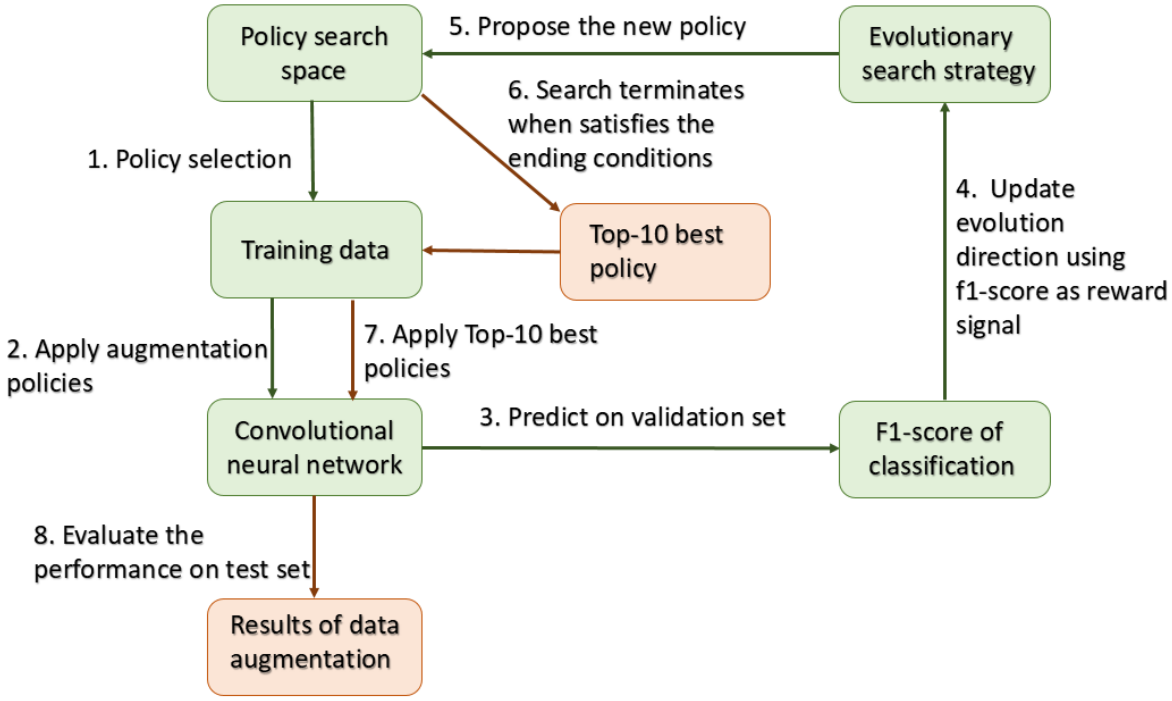


Figure 3.1: Framework of implementing data augmentation

In the following sections, we will explain how to implement some of the steps in the framework specifically.

3.2.1 Policy Parameter Search Space

Every policy consists of 5 sub-policies. We give a visualized structure of the one policy in Figure 3.2. We marked one sub-policy in detail. As we can see, there are two operations and each operation has a probability parameter and a magnitude parameter. We put them in a 30-dimensional vector and split it into ten 3-dimensional vectors. In each 3-dimensional vector, operation type takes the first position. Probability stands the second position which ranges randomly from 0 to 1 and represents the probability of applying this operation. Thus, note that an operation, according

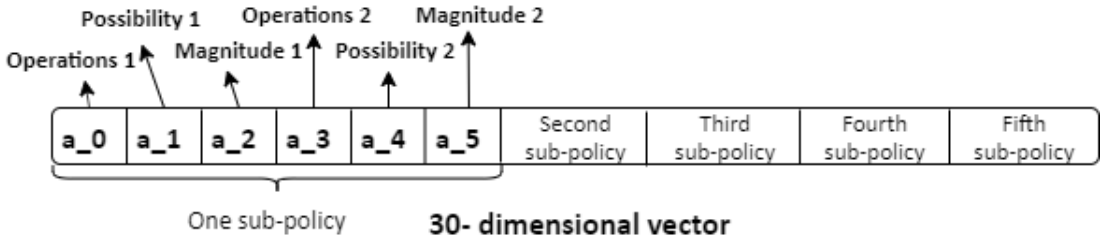


Figure 3.2: Structure of one policy

to a chosen policy, may or may not be applied to individual samples as a result of a random draw. Magnitude stays at the third position, which varies depending on the type of operation.
There are 12 operation types in this project, they are **Scaling**, **Jitter**, **Rotation**, **Time Warp**, **Permutation**, **Magnitude Warp**, **Shift**, **Channel Shuffle**, **Horizontal Flip**, **Negate**, **Cutout**, and **Blend**. While the first six of these come from [28], we proposed six more transformation methods. The operation is formulated by $o = \lfloor p_0 \times O_n \rfloor$, O is the set of operations; possibility equal to p_1; the magnitude of the operation is computed by $p_2 \times (Magnitude_{max} - Magnitude_{min}) + Magnitude_{min}$, where p_i denotes a random value ranging between 0 and 1. Therefore, there are uncountable possibilities in the search space.
The description of each operation and the range of magnitude are shown in Table 1, and we plot the examples of every operation in Figure 3.3. Magnitudes of **Rotation**, **Channel shuffle**, **Horizontal Flip**, and **Negate** vary in {0, 1}, which means this operation only has two magnitude parameters, this operation will be applied when magnitude is 1, will not be applied when magnitude is 0. Magnitudes of **Time Warp** and **Magnitude Warp** are the magnitude of the standard deviation of the Gaussian distribution. For example, for scaling operation, the random parameter p_i of magnitude and probability are 0.5 and 0.7, then through the calculation of $0.7((2-0)+0) = 1.4$, each sample in one batch has 0.5 probability of getting multiplied by 1.4.

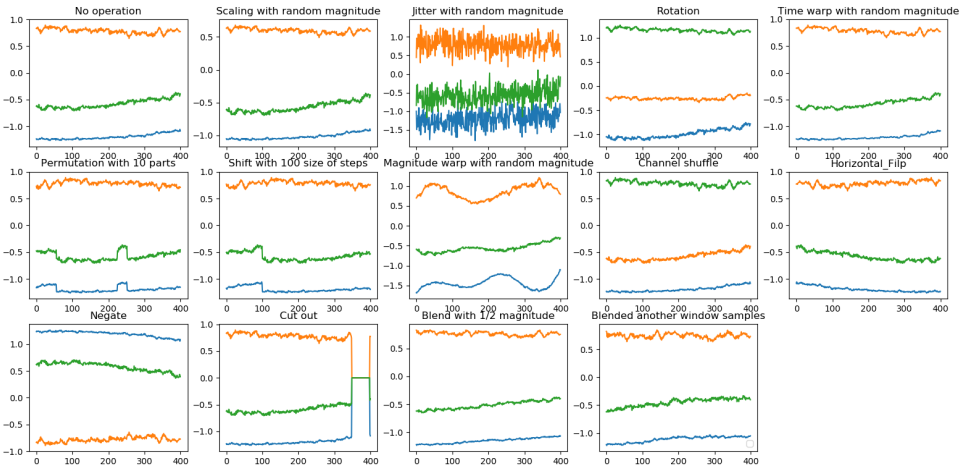


Figure 3.3: Visualization of each operation:the final subplot is the samples of another window to implement the operation **Blend**

Operation	Description	Magnitude range
Scaling	Multiply a random scalar N on entire samples	N ∈ [0, 2]
Jitter	Add the random noise R on each samples	R ∈ [0, 2]
Rotation	Change the sensor placement, that is the conversion of sample signs (without changing the associated class-label) for example, held upside down	{0, 1}
Time Warp	Stretches or warps a time-series of time intervals between samples through a smooth distortion.	[0, 2]
Permutation	Slice a window evenly into N segments, and permute the segments	N ∈ [1, 10]
Magnitude Warp	All samples multiply a random curve	[0, 2]
Shift	Roll all samples N steps along with x axis in one window	N ∈ [0, 100]
Channel shuffle	Entirely exchange the samples in different channel	{0, 1}
Horizontal Flip	Flip the samples along with vertical axis	{0, 1}
Negate	Change all samples to negative	{0, 1}
Cutout	Random remove a segment with N size in one window	N ∈ [0, 50]
Blend	Mix (1-N) times samples and N times samples from another window	N ∈ [0, 0.5]

Table 3.1: Descriptions of operations

3.2.2 Policy Evaluation Strategy

Generally, classification accuracy is applied to assess the performance, however, accuracy cannot reflect the classification performance of imbalanced data well. Therefore, in this project, we used the F1-score as fitness to update the evolutionary search strategy. Combining with kappa, precision and recall, we evaluate the performance of data augmentation.

Precision and Recall Precision and recall are widely used in the fields of information retrieval and binary classification, these two measures are used to evaluate the quality of results. Before we introduce the definitions of precision and recall, let us see a question of binary classification, given a training set:

$$D = \{(x_i, y_i)|x_i \in R^n, y_i \in \{0, 1\}\}_{i=1}^{N} \tag{3.1}$$

where $y_i = 1$ represents positive sample, $y_i = 0$ represents negative sample.
Assume there is classification model H, for every input x_i, compare $H(x_i)$ and y_i we can get four different situations, which are $H(x_i) = 1, y_i = 1$ named true positive (TP), $H(x_i) = 1, y_i = 0$ named false positive (FP), $H(x_i) = 0, y_i = 1$ named false negative (FN), and last situation $H(x_i) = 0, y_i = 0$ named true negative (TN). Further, we can define the precision and recall as the following equation:

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

From the above equation, we conclude recall indicates a model's ability of recognizing positive

samples. Higher recall indicates the model are more able to recognize positive samples, while precision indicates the model's ability of distinguishing negative samples. Higher precision indicates the model owns the ability to distinguish negative samples.

F1-score F1-score is a measure of the accuracy of binary classification in statistics, which both consider the precision and recall to compute the score. The higher the F1-score is, the more robust the classification model is. The F1 equation shows how to calculate F1-score.

$$F1 = \frac{2TP}{2TP + FN + FP} = \frac{2Precision \cdot Recall}{Precision + Recall} \tag{3.4}$$

Cohen's Kappa Score Cohen's kappa coefficient (k) is used for consistency checks and can also be used to measure the accuracy of a binary classifier, which is very useful in the case of imbalanced data. The definition of k is:

$$F1 = \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e} \tag{3.5}$$

where p_o is the sum of the number of correctly classified samples in each category divided by the total number of samples, that is, the overall classification accuracy. $p_e = \frac{a_1 \cdot b_1 + a_2 \cdot b_2 + ... + a_m \cdot b_m}{n \cdot n}$, where $a_1, a_2..., a_m$ represents the number of samples of each category respectively, $b_1, b_2..., b_m$ represents the number of samples that are predicted for each category, and n in the formula denotes the total number of samples. Kappa score is between -1 and 1. If Kappa is equal to 0 that means the classifier is no different from a random guess, but the guess here is based on the overall distribution. The higher the score, the better the classifier. Kappa close to -1 means the performance is awful.

3.2.3 Evolution Search Strategy

The implementation of the evolutionary search strategy relies on Nevergrad[19]. Nevergrad is a library in Python, which supports several different kinds of gradient/derivative-free optimization algorithms, including random search, CMA, PSO, and so on. Nevergrad's PSO algorithm is based on the latest Standard Particle Swarm Optimization algorithm (SPSO-2011) [31]. In [17] it is proposed that simple random search has higher efficiency than reinforcement learning. Then ARS-Aug[9] evolves AutoAugment by using random search as search algorithm and achieves the better performance. Except for these empirical results, evolutionary search strategy also has theoretical advantages, including high scalability, high robustness, and convenient use.
In this project, we adopt random search as a basic search algorithm. In addition, we adopt two more evolution search strategies, CMA and PSO. CMA is one of the most widely used derivative-free optimization algorithms. An obvious feature of this method is the self-adaptation of sampling distribution parameters based on the sorting of existing solutions. PSO is suitable for finding the optimal solution region in high dimensional space. However, for a function with multiple local extreme points, it is easy to get stuck into the local extreme points. PSO algorithm is a kind of heuristic bionic optimization algorithm. At present, there is no strict theoretical basis to explain why the algorithm is effective.

Chapter 4

Experiments

In this chapter we will introduce the time-series data we used in this project firstly and pre-processing on these data, then will describe the neural network with specific hyper-parameters, finally, we will represent all results we obtained and analyse them.

4.1 Data Description

In this project we used five time-series datasets, they are all collected for human action recognition (HAR). All of these datasets are collected by two embedded sensors: Accelerometer and Gyroscope, which are the most common sensors to determine the angular position and movement of an object. In most of the experiments in this project, we use both sensory data. The gyroscope is a device using Earth's gravity to determine an angular position, the accelerometer is a device to measure non-gravitational acceleration. Both of them capture by three channels data that are x, y, and z-axis.

Heterogeneity Dataset for Human Activity Recognition This dataset is collected by Allan Stisen et al. in [27], which is used to systematically evaluate the specific heterogeneity of sensors, devices, and workloads. There are six different user activities: "Cycling", "Sitting", "Standing", "Walking", "Up-stair" and Downstairs". The dataset is collected from 9 participants using 8 smartphones and 4 smartwatches. The sampling rate of signals varied across different devices therefore we conclude them in Table 4.1. This dataset was divided into two datasets in our project, one is all collected from smartphones and denoted by HHAR in the following, the other one is collected from smartwatches and denoted by SW in the following part.

Models	Number of instance	Max sampling rate (Hz)
Smartwatch		
LG G	2	200
Samsung Galaxy Gear	2	100
Smartphone		
LG Nexus 4	2	200
Samsung Galaxy S+	2	50
Samsung Galaxy S3	2	150
Samsung Galaxy S3 mini	2	100

Table 4.1: Device models used with THE number of instances used, and accelerometer sampling rate

MobiAct Dataset This dataset was introduced by Charikleia Chatzakweet et al. and published in [1]. The MobiAct dataset is collected by a smartphone including 4 different types of falls and 12 different activities of daily living (ADLs) from a total of 66 subjects with more than 3200 trials. In practical use, there are 11 activities in total because the twelfth activity "Lying" is used to determine the fall. The remaining eleven activities used in our experiment include: "Standing", "Walking", "Jogging", "Jumping", "Stairs up", "Stairs down", "Stand to sit", "Sitting on chair", "Sit to stand", "Car step in" and "Car step out". According to [29] authors use 87 Hz to sample accelerometer data and 200 Hz to sample gyroscope data.

MotionSense Dataset This dataset is published by Mohammad Malekzadeh et al. in [16]. This dataset is collected from accelerometer, gyroscope and attitude (roll, yaw, pitch) installed in an iPhone 6s, which was kept in the 24 participants' front pocket. We used the accelerometer and gyroscope data in this project. Activities in this dataset include "Downstairs", "Upstairs", "Walking", "Jogging", "Sitting", and "Standing".

Human Activities and Postural Transitions Dataset (HAPT) This dataset is published in [20] and used to present a Transition-Aware Human Activity Recognition (TAHAR) system architecture proposed by the authors. The activities they recorded by a smartphone (Samsung Galaxy S II) wearing on the 30 participants' waist include: three static postures (standing, sitting, lying), three dynamic activities (walking, walking downstairs and walking upstairs), and six postures transitions between posture pairs (stand-to-sit, sit-to-stand, sit-to-lie, lie-to-sit, stand-to-lie, and lie-to-stand). Data is captured by a 50 Hz sampling rate.

4.2 Data Pre-processing and Segmentation

All datasets are recorded as the format of Comma-Separated Values (.csv). The specific pre-processing is to sample the data into a fixed size through sliding a window. For all datasets, we used the same window size 400, which means the sequence size fed into the neural network is 400 $\times$3, 400 data points in one sequence and each data point is represented by 3-axis x, y, z.

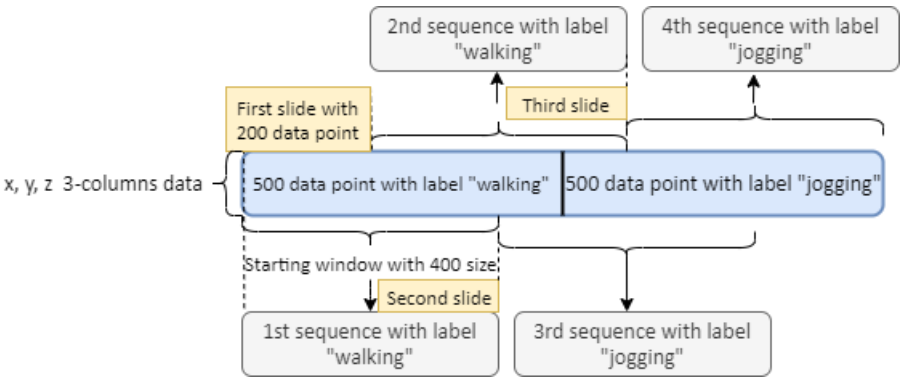


Figure 4.1: Visualization of segmentation of data and window sliding

When using a window to slide the data, there is a problem related to the label of a sequence, because there are two activities in one window. However, every activity needs to be recorded for a period of time, it is impossible to have more than two activities in one window size. Facing this situation, the activity (label) will be marked as the one with more data points in a window. There is a simple example shown in Figure 4.1, the bar represents the original 3-columns 1000 data points. The window starts from beginning to sample data, every time slides 200 data points and generates a new sequence.
We use one-hot encoding to process labels. For example, there are six activities in the dataset, we

put them in a 6-dimensional vector. The first dimension of the vector represents the first activity "Biking", the second activity is put in the second dimension, and so on. In the label vector, 1 means the data point belongs to this activity, 0 means the opposite.
We used classification results of the validation set to update the evolutionary search strategy, the validation data split from the training set, a quarter of training data is used as the validation set. All datasets are processed by the same method, and we conclude the information of all datasets in Table 4.2 after processing.

Dataset	Training set size	Test set size	Activities number
HHAR	30991	15922	6
SW	18616	8702	6
MobiAct	59316	21262	11
Motionsense	4408	2612	6
HAPT	2282	1166	12

Table 4.2: Information of every dataset after pre-process

4.3 Neural Network Architecture Design

The neural network is built using Keras in this project, which is a high-level neural networks API [2]. The model we used mainly is based on convolutional neural network. We have introduced the convolutional layer, pooling layer, normalization layer and dropout layers in Chapter 2. Considering our time-series data, the model in this project combined two inputs to predict labels jointly. The structure of the model with two modalities is shown in Figure 4.2. In the following sections, we also represent the results of using a single modality, which means we use the accelerometer or gyroscope data individually to train the neural network and evaluate the performance of data augmentation.

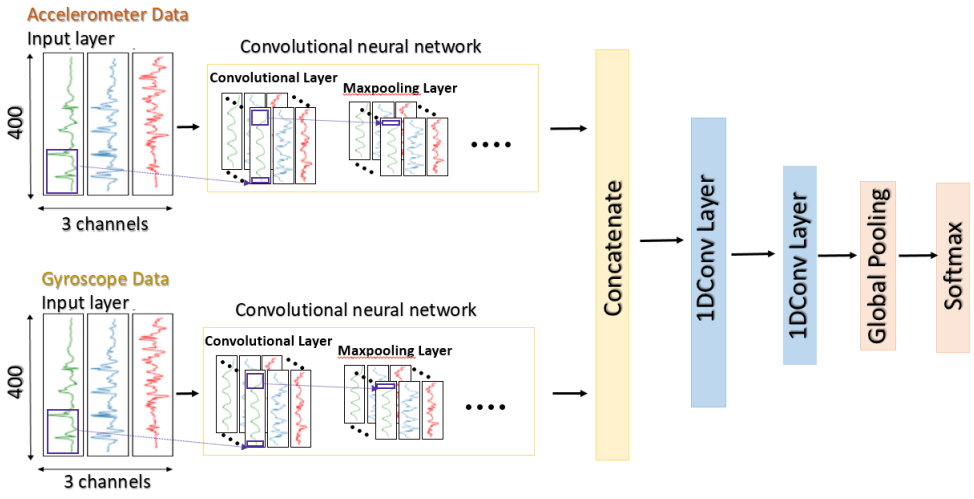


Figure 4.2: The structure of two modalities model used in this project

The convolutional neural network in this model is made up of three 1D convolution layers with 32, 64, 96 filter size and 24, 16, 8 kernel size respectively. Filter controls the dimension of output, and kernel size specifies the length of the convolution window. The activation function of convolution layer uses ReLu, defined as $f(x) = max(0, x)$. L2 regularization function applies penalties to the kernel weights matrix to avoid the overfitting. L2 regularization is adding a regularizer in the cost

function. The function is defined as the following:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{4.1}$$

where the $\lambda \sum_{j=1}^{p} \beta_j^2$ represents L2 regularization. We set the rate of the L2 regularizer to 0.0001. Between each two convolution layers we add a Maxpooling layer with 4 pooling windows and 2 strides to downscale, which could halve the input. Then it connects to a dropout layer with 0.1 rates. These layers form the convolutional neural network that appears in Figures 3.4 and 4.2.
For the single modality model, after the convolutional neural network, we connect to one convolutional layer, the first one matched the 64 filter size and 4 kernel size and the other parameters are the same with the convolution layers introduced before. The output part consists of one convolutional layer that the filter size equal to the number of classes for the different dataset, one GlobalAveragePooling layer, which could replace the traditional dense layer at the end of the classification model introduced by [14], and a softmax activation layer to implement classification. The detailed structure of this model is given in Appendix A. For the model with two modalities, there are two inputs and connect to a convolutional neural network respectively, then using a concatenate layer to joint two networks, the rest of the model is all the same single modality model. The parameters and structure of the model are determined based on good classification performance of the model without data augmentation.

Generator Because there is a mass of combinations of transformation and one time-series can be transformed into countless forms, it is impossible to transform the original data first and save it for training, so we used the original data to generate new data while training, that requires a generator. In Keras, the "fit_generator" is used for this type of training, which trains the model on data generated batch-by-batch by a Python generator. There are two parameters of the "fit_generator" we set, one is "steps_per_epoch" which is the total number of batches to yield from the generator before the next epoch starting [2]. The other one is the number of times to train the model. In the part of generating the top-10 best policies, we set the "steps_per_epoch" equal to 1000 and the epoch is 10, in the part of using the best policies to augment data, the epoch is 20 and "steps_per_epoch" varies depending on the amount of data.

4.4 Experimental Results

We will represent all the results obtained from this project in this section. All experiments will be presented with the average weighted precision, average weighted recall, average weighted F1-score, and Kappa. Average weighted means that the number of true instances for each label will be considered. The following results of each experiment in each table are averaged over ten separate experiments to avoid accidental high or low result from one experiment.

4.4.1 Varying Budget for Policy Search

The budget is a hyperparameter related to ES algorithms, the budget represents the number of trainings used to find the optimum. Therefore, before obtaining the augmentation policy, we need to determine this parameter. We change the number of budgets and use the random search algorithm to generate policies on the HHAR dataset. Classification results are summarized in Table 4.3.
In the beginning, all the indicators get better with the increase of budget amount, but when the budget is 75, all the indicators start to drop, even worse than when the budget is 30, therefore, in the following the experiments we fix the budget number at 50.

Budget Number	Precision	Recall	F1-score	Kappa
15	0.81±0.0126	0.802±0.0116	0.792±0.0097	0.7626±0.0114
30	0.842±0.0132	0.821±0.0122	0.812±0.0124	0.7861±0.0141
50	0.852±0.0074	0.83±0.01	0.819±0.0137	0.7958±0.012
75	0.8309±0.0221	0.815±0.0162	0.8059±0.0128	0.7776±0.017

Table 4.3: Classification results of different number of budget using random search algorithm to generate augmentation policies on HHAR dataset

4.4.2 Varying Number of Operations In A Policy

We have described the structure of the one policy in Section 3.2.1. One policy consists of 5 sub-policies and each sub-policy has two operations. We want to check whether the number of operations has influence on the classification result. Thus we try some experiments changing the number of operations and using PSO and CMA algorithms respectively on HHAR dataset, and classification results is shown in Table 4.4.

Algorithm and number of operations	Precision	Recall	F1-score	Kappa
PSO algorithm with two operations	0.8379±0.0218	0.817±0.0155	0.807±0.0155	0.7802±0.0181
PSO algorithm with three operations	0.818±0.0124	0.799±0.0113	0.788±0.01017	0.7572±0.0138
CMA algorithm with two operations	0.843±0.0126	0.825±0.0067	0.813±0.0064	0.7878±0.0067
CMA algorithm with three operations	0.852±0.0074	0.83±0.0089	0.818±0.0116	0.7929±0.0099

Table 4.4: Classification results of using different number of operations in one policy

Through Table 4.4 we could see, using three operations in one sub-policy is inferior to using two operations in one sub-policy when using the PSO algorithm. By comparison, using the CMA algorithm with three operations has no obvious improvement with using two operations. In the meanwhile, the training time for picking up the top-10 best policies increase dramatically. More operations do not demonstrate more advantages, so in all the other experiments, we will adopt two operations in one sub-policy.

4.4.3 Experiments of Applying Data Augmentation

We would first like to show that our data augmentation method is working, so we train the models without data augmentation and with data augmentation. The model without augmentation has the same structure and hyper-parameters with the model with augmentation which we have introduced before. We conclude classification performance into Table 4.5. In this table, the results of using data augmentation all use the top-10 best policies generated by the CMA-ES algorithm with 50 budgets.
From Table 4.5 we could see the classification results of the model trained with data augmentation improve clearly, especially on HHAR, HAPT, and SW dataset. As for the performance of MobiAct and Motionsense, maybe because of the characteristics of the data, the classification results are high without data augmentation. However, there still exists a slight improvement when uses data augmentation. Therefore, our data augmentation method is proved effective.

Dataset and Model	Precision	Recall	F1-score	Kappa
HHAR without Aug	0.764±0.0143	0.7123±0.0166	0.7123±0.0158	0.6546±0.0199
HHAR with Aug	**0.86±0.0126**	**0.836±0.091**	**0.827±0.0089**	**0.8045±0.0114**
MobiAct without Aug	0.928±0.0024	0.9245±0.0038	0.9242±0.0032	0.903±0.0041
MobiAct with Aug	**0.96±0.0**	**0.96±0.0**	**0.96±0.0**	**0.9487±0.0024**
Motionsense without Aug	0.9332±0.012	0.9218±0.0186	0.9233±0.0179	0.9038±0.0224
Motionsense with Aug	**0.96±0.0063**	**0.954±0.0066**	**0.955±0.0067**	**0.9458±0.0104**
HAPT without Aug	0.8025±0.0185	0.8079±0.0195	0.7995±0.0187	0.7734±0.0232
HAPT with Aug	**0.895±0.0048**	**0.894±0.0048**	**0.893±0.0045**	**0.876±0.0063**
SW without Aug	0.7±0.0232	0.6863±0.0231	0.6805±0.0259	0.622±0.0199
SW with Aug	**0.792±0.0222**	**0.778±0.0294**	**0.771±0.0334**	**0.733±0.0406**

Table 4.5: Conclusion of performance of non-using and using data augmentation: "dataset name without Aug" denotes the model is trained without any augmentation, "dataset name with Aug" denotes the model is trained with data augmentation

4.4.4 Comparing Different Evolutionary Algorithms

In the previous section we presented the results of using augmentation policies generated by the CMA-ES algorithm. Nevergrad provides many ES algorithms, therefore we also compare the classification performance using different algorithms to generate policies to augment data. In this part, we use PSO, CMA and random search (RS) as search algorithms and the budget is also 50. Basic random search is a simple algorithm, which can be described as : initialize x with a random position in the search space, then randomly sample a new position y from the given radius surrounding x, if fitness or cost function of the new position is smaller than old position, position will be updated as the position of y, repeat above steps until a termination condition is met. The classification results are also concluded in Table 4.6.
According to the results from Table 4.6, on the whole, the performance of the three algorithms is close to each other. Random search shows the best performance on the last two datasets. PSO and CMA perform almost the same on the MobiAct and HAPT dataset, CMA performances a slightly worse than PSO on the Motionsense dataset. CMA algorithm is better than PSO on the HHAR dataset and the advantage is obvious. Nevergrad also proposes the PSO algorithm is not very efficient when the number of the budget is smaller than 60. CMA is a representative evolutionary search strategy, besides, it has some advantages of performance among the three algorithms. Therefore the CMA algorithm will be the main algorithm in the following experiments.

Dataset and Policy	Precision	Recall	F1-score	Kappa
HHAR using RS policies	0.852±0.0074	0.83±0.01	0.819±0.0137	0.7958±0.012
HHAR using PSO policies	0.836±0.0142	0.817±0.0141	0.807±0.0141	0.7815±0.0161
HAAR using CMA policies	**0.86±0.0126**	**0.836±0.091**	**0.827±0.0089**	**0.8045±0.0114**
MobiAct using RS policies	0.9579±0.004	0.9579±0.004	0.9569±0.0045	0.945±0.0034
MobiAct using PSO policies	0.96±0.0	0.96±0.0	0.96±0.0	0.9487±0.0021
MobiAct using CMA policies	**0.96±0.0**	**0.96±0.0**	**0.96±0.0**	**0.9487±0.0024**
Motionsense using RS policies	0.96±0.0063	0.9559±0.0091	0.9559±0.0091	0.9458±0.0104
Motionsense using PSO policies	**0.964±0.0137**	**0.961±0.0137**	**0.961±0.0137**	**0.9531±0.0159**
Motionsense using CMA policies	0.96±0.0063	0.954±0.0066	0.955±0.0067	0.9458±0.0104
HAPT using RS policies	**0.902±0.004**	**0.901±0.003**	**0.899±0.005**	**0.883±0.0043**
HAPT using PSO policies	0.895±0.005	0.893±0.0045	0.893±0.0045	0.874±0.0058
HAPT using CMA policies	0.8959±0.0048	0.894±0.0048	0.893±0.0045	0.876±0.0063
SW using RS policies	**0.816±0.0142**	**0.809±0.0157**	**0.803±0.0167**	**0.768±0.0189**
SW using PSO policies	0.796±0.0161	0.79±0.0141	0.784±0.0149	0.747±0.01664
SW using CMA policies	0.792±0.0222	0.778±0.0294	0.771±0.0334	0.7335±0.0406

Table 4.6: Conclusion of performance of data augmentation policies generated by different algorithms

4.4.5 Experiments of Single Modality Model with Data Augmentation

In the previous experiments, all results were obtained from two modalities model, which are combined with accelerometer and gyroscope data together to train the model. Now we want to test the effect of single data set on classification results. Also, using the CMA algorithm with 50 budgets to produce policies, and the approach flow and parameters are the same except for the structure of the model. The final results we summarize them in Table 4.7.
Combined with the results of Table 4.7 and Table 4.8, we could find that the accelerometer data could exhibit better performance, however, the results are relatively lower than combining the two sensories data. It is worth to notice the results of using gyroscope data of HAPT are overly poor, which almost does not have any classification value. However, referring to Table 4.5, concatenating two time-series data still improves the score of every indicator. At the same time, gyroscope data of MotionSense and MobiAct also cannot perform a favorable classification. If we just assess the scores of using gyroscope data of MotionSense and MobiAct dataset, which seem to be good. However, compared with the results without augmentation in Table 4.5, they are still not competitive. In conclusion, using gyroscope data alone cannot classify well, but combining this data with accelerometer data can indeed improve the classification effect to a certain extent.

Dataset Source	Precision	Recall	F1-score	Kappa
Accelerometer of HHAR	0.8459±0.0185	0.822±0.0166	0.8119±0.0166	0.7846±0.0207
Accelerometer of MotionSense	0.9469±0.0064	0.943±0.0064	0.943±0.0064	0.9295±0.0068
Accelerometer of MobiAct	0.946±0.0004	0.9489±0.003	0.946±0.0048	0.9318±0.0031
Accelerometer of SW	0.773±0.0179	0.766±0.022	0.761±0.0192	0.7167±0.0248
Accelerometerof HAPT	0.8649±0.0102	0.861±0.0094	0.86±0.0109	0.8369±0.0113

Table 4.7: Classification results of using accelerometer data alone.

Dataset Source	Precision	Recall	F1-score	Kappa
Gyroscope of HHAR	0.5999±0.0126	0.596±0.0142	0.59±0.0161	0.5113±0.0161
Gyroscope of MotionSense	0.883±0.0141	0.88±0.0184	0.881±0.0164	0.8505±0.0216
Gyroscope of MobiAct	0.899±0.004	0.898±0.004	0.898±0.004	0.8699±0.0052
Gyroscope of SW	0.655±0.0143	0.655±0.0128	0.641±0.0175	0.584±0.0161
Gyroscope of HAPT	0.382±0.0097	0.403±0.01	0.38±0.01	0.2891±0.0129

Table 4.8: Classification results of using gyroscope data alone.

4.4.6 Assessing Transferability of Policy

One of the main purposes of our project is to prove that policy generated by one dataset could be transferred to another similar dataset. The polices we used in this and the following experiments are all generated using the CMA algorithm with 50 budgets from every dataset. The specific implementation is we choose a target dataset, then train it with the augmentation policy learned from the other datasets. For a specific example, We choose HHAR as a target dataset and respectively use policies learned from the other four datasets to implement data augmentation. Table 4.9-4.13 are the summaries of classification results of HHAR, MobiAct, MotionSense, HAPT, and SW dataset as target dataset respectively. The last row of each table is the result of using the policy learned from the target dataset.

Policy Source	Precision	Recall	F1-score	Kappa
MobiAct	0.837±0.0179	0.823±0.01	0.813±0.0109	0.7872±0.0137
MotionSense	0.798±0.0124	0.792±0.0107	0.782±0.0107	0.7484±0.0122
SW	0.825±0.0162	0.804±0.0135	0.794±0.0128	0.7633±0.0157
HAPT	0.835±0.0287	0.817±0.0189	0.807±0.0179	0.7789±0.0232
HHAR	**0.86±0.0126**	**0.836±0.091**	**0.827±0.0089**	**0.8045±0.0114**

Table 4.9: Classification results of HHAR dataset using augmentation policies learned from the other datesets: the last row is the results of using augmentation policy produced by the dataset itself

The results in bold in each table show the best performance among the different policies adop-

Policy Source	Precision	Recall	F1-score	Kappa
MotionSense	0.954±0.0048	0.954±0.0048	0.954±0.0048	0.9406±0.039
HHAR	0.959±0.003	0.959±0.003	0.959±0.003	0.9477±0.0039
HAPT	0.949±0.0	0.949±0.0	0.949±0.0	0.9376±0.0029
SW	0.9479±0.004	0.9469±0.0045	0.946±0.0048	0.9311±0.0025
MobiAct	**0.96±0.0**	**0.96±0.0**	**0.96±0.0**	**0.9487±0.0024**

Table 4.10: Classification results of MobiAct dataset using augmentation policies generated by other dateset

Policy Source	Precision	Recall	F1-score	Kappa
MobiAct	**0.968±0.004**	**0.965±0.0067**	**0.965±0.0067**	**0.9559±0.0079**
HHAR	0.952±0.006	0.9439±0.079	0.946±0.0079	0.9324±0.0092
HAPT	0.962±0.006	0.9579±0.0097	0.959±0.0094	0.9459±0.0103
SW	0.954±0.0101	0.949±0.0129	0.952±0.0124	0.938±0.0144
MotionSense	0.96±0.0063	0.954±0.0066	0.955±0.0067	0.9458±0.0104

Table 4.11: Classification results of MotionSense dataset using augmentation policies generated by other dateset

Policy Source	Precision	Recall	F1-score	Kappa
MobiAct	0.894±0.0066	0.894±0.0048	0.891±0.0053	0.8746±0.0045
HHAR	**0.898±0.004**	**0.8959±0.0044**	**0.8949±0.0048**	**0.8774±0.0035**
MotionSense	0.895±0.005	0.892±0.004	0.8899±0.0044	0.8744±0.0043
SW	0.885±0.005	0.8869±0.0045	0.885±0.005	0.8671±0.006
HAPT	0.8959±0.0048	0.894±0.0048	0.893±0.0045	0.876±0.0063

Table 4.12: Classification results of HAPT dataset using augmentation policies generated by other dataset

Policy Source	Precision	Recall	F1-score	Kappa
MobiAct	0.796±0.0091	0.79±0.01	0.782±0.0116	0.7458±0.0113
HHAR	**0.8139±0.0091**	**0.806±0.0111**	**0.8019±0.0107**	**0.7669±0.0114**
MotionSense	0.782±0.0132	0.776±0.0111	0.7699±0.0126	0.7306±0.0129
HAPT	0.8109±0.0129	0.805±0.012	0.799±0.0157	0.765±0.0147
SW	0.792±.0222	0.778±0.0294	0.771±0.0334	0.7335±0.0406

Table 4.13: Classification results of SW dataset using augmentation policies generated by other dateset

ted. Observing each table, we can discover the score of every indicator is close to each other, which means the augmentation policy learned from similar datasets has the transferability. In the summaries of the SW dataset, it appears the situation that using policies learned from the other datasets (HHAR, MobiAct, and HAPT) obtains the better classification result than using policies generated by itself. In Table 4.9 and Table 4.13, using policies generated from the MotionSense dataset performs the worst and the gap is relatively obvious. Correspondingly, using the policy learned from HHAR is also the worst one in the results of MotionSense as target dataset. According to the description of HHAR in [27], they also tried to collect data using the iPhone. However, because of the different operating system, it is hard to compare fairly the iPhones' HAR performance with other device models. Therefore, they did not use data collected from iPhone. As it happens, MotionSense dataset was collected using the iPhone. This reflects from the side that our data augmentation method can make up for the problem caused by different data collection equipments to some extent.

4.4.7 Effectiveness of Data Augmentation on Few Data

The ultimate goal of data augmentation is to produce more artificial data. Therefore, we simulate the situation that we have a very small size dataset. In this part of the experiments, we pick randomly 5, 10, 20, 50 instances per class in one dataset. Then we use these data to train a model without augmentation, and train models with augmentation using policies learned from the other normal size datasets. Finally, we evaluate the classification performance. We plot the results in the following pictures (Figure 4.3-4.7). In each following picture, the left picture is the representation of Kappa and the right one corresponds to F1-score. The x-axis of each sub-picture represents the number of instances, and each line in every sub-picture represents the results of different policy source. We also summarize all specific results of five datasets with different instances in tabular form and provide them in Appendix B.

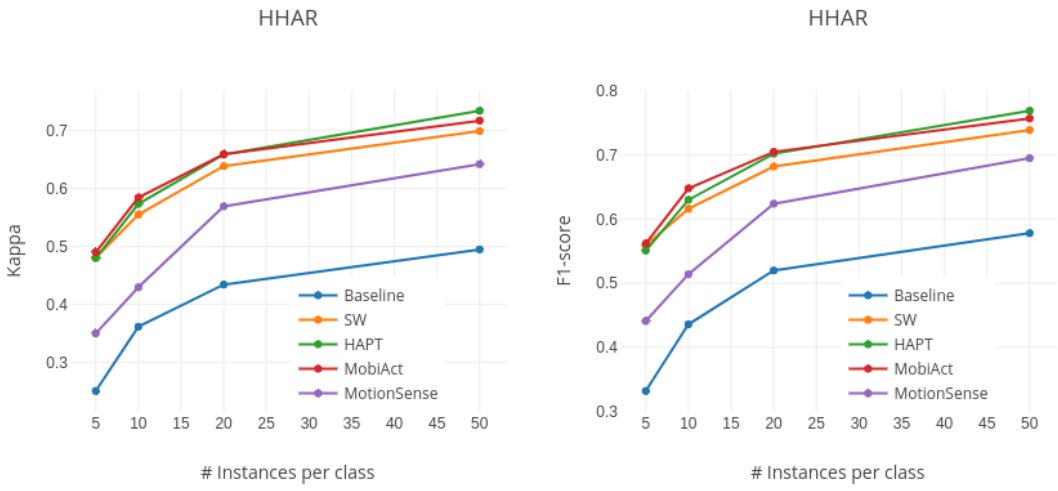


Figure 4.3: Classification results of using few instances of HHAR

The blue line in every picture denotes the model trained without any data augmentation. Obviously, for normal neural network model can not handle the situation with a very small size. Using data augmentation improves the results significantly. When the number of instances is 20, the model without augmentation could achieve comparable performance on indicators with the model trained with augmentation using 5 instances. The results of every dataset reveal a distinct gap between models trained with and without data augmentation when using 50 instances per class except the MotionSense. Although, the more instances, the better, when using 20 instances per

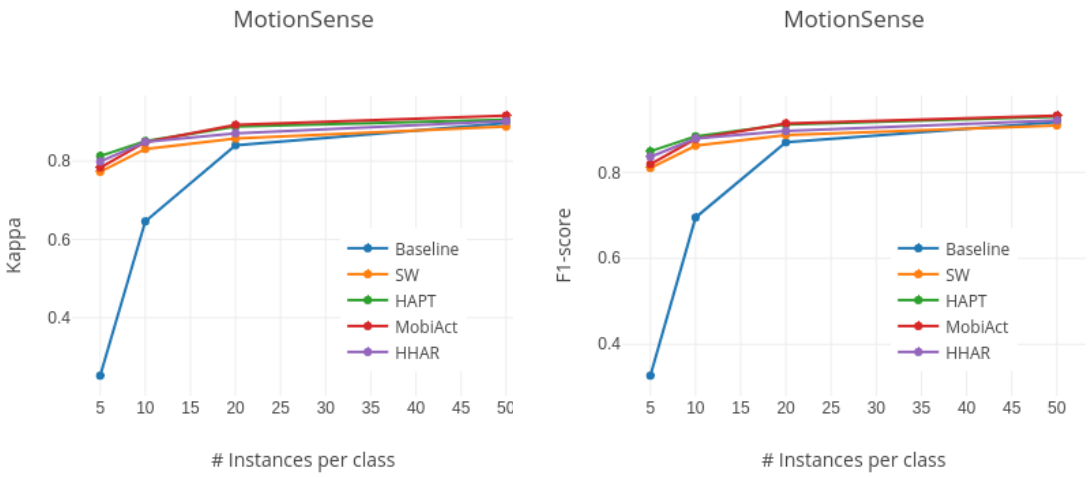


Figure 4.4: Classification results of using few instances of MotionSense

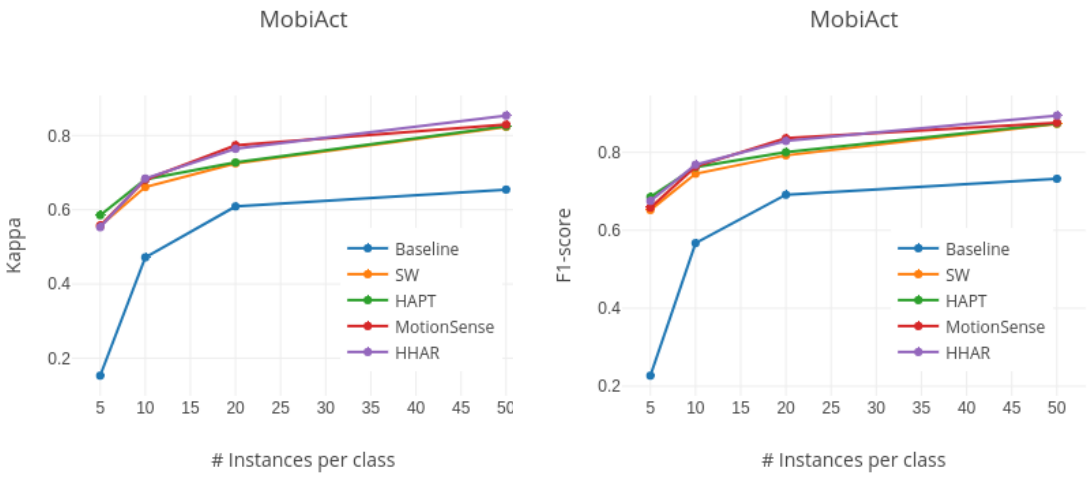


Figure 4.5: Classification results of using few instances of MobiAct

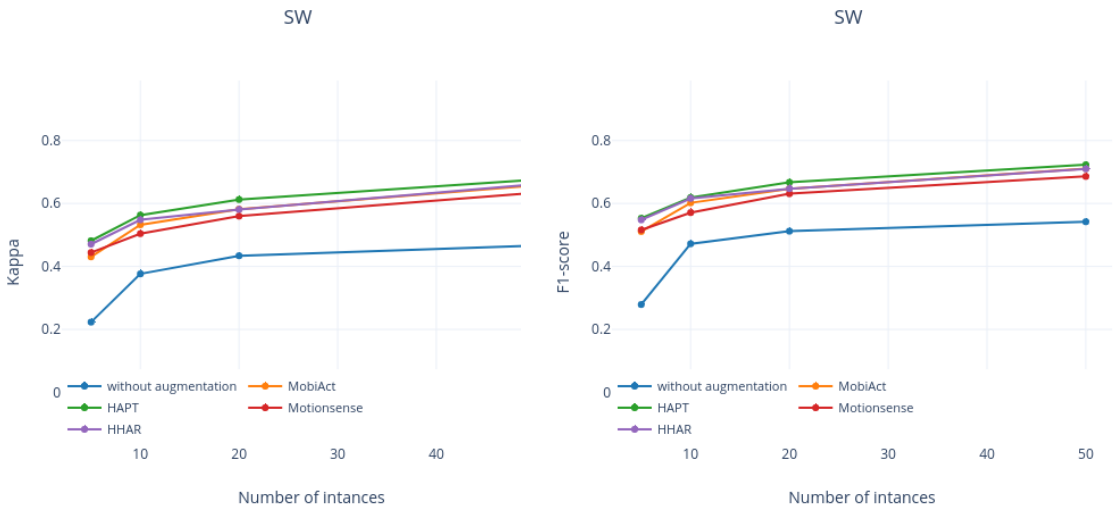


Figure 4.6: Classification results of using few instances of SW

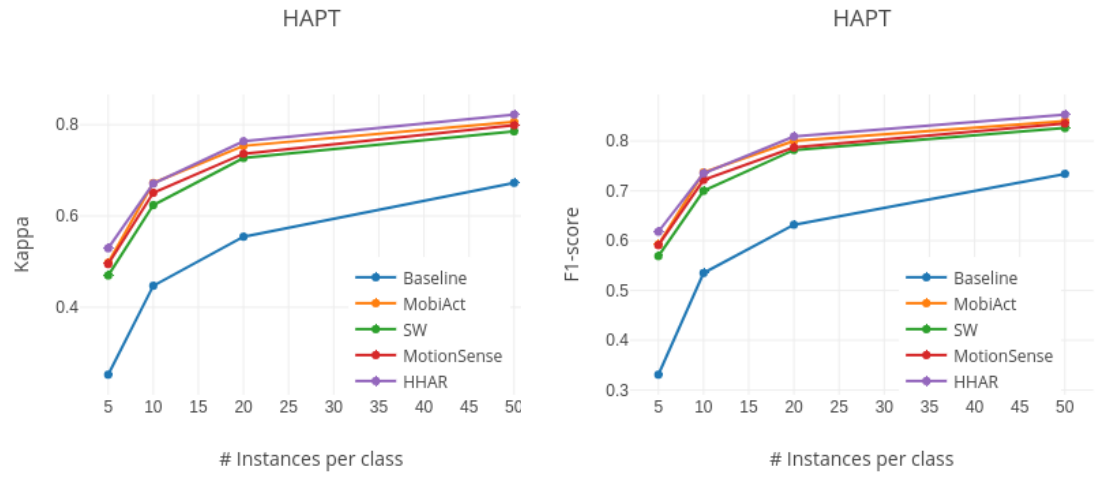


Figure 4.7: Classification results of using few instances of HAPT

class, the model trained with augmentation owns the ability of effective classification referring to the results of Figure 4.5 and Figure 4.7.

4.4.8 t-SNE Visualization

T-distributed stochastic neighbour embedding is a machine learning algorithm for dimensionality reduction, which was proposed by Laurens van der Maaten and Geoffrey Hinton [15]. It is a non-linear dimensionality reduction algorithm, which is very suitable for high-dimensional data reduction to 2 or 3 dimensions for visualization.
The implementation of t-SNE in this project depends on the scikit-learn. We get the activations of the penultimate convolution layer from the model trained with and without data augmentation respectively. Then we pass them to a global max-pooling layer with 64 hidden units and map them on to the 2D space. We pick randomly about 4000 samples to train a model from each dataset, specifically, 650 instances per class from HHAR and SW; 350 instances per class from MobiAct and all data of HAPT and MotionSense because their size is not over 4000. For all datasets, we use 100 perplexities to generate t-NSE. The perplexity can be interpreted as a smooth measure of the effective number of neighbours, and it has a complex effect on the final output. The following pictures are the distribution of data points. The left sub-picture of each figure is the distribution of model trained without augmentation and the right sub-picture corresponds to the distribution of model trained with augmentation.

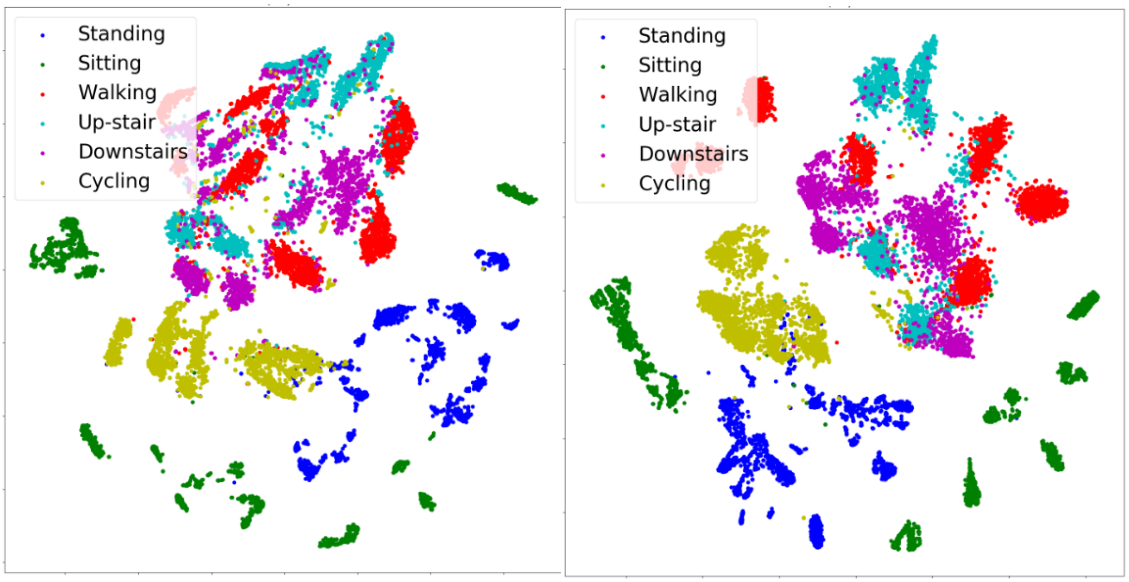


Figure 4.8: t-SNE visualization of HHAR dataset

Through comparing sub-pictures in each figure, we could find the model trained with data augmentation owns a clearer boundary of the cluster. Firstly, let us observe Figure 4.8 and 4.9 because they are collected by the same activities. Either in Figure 4.8 or Figure 4.9, activities "Walking", "Up-stair" and "Downstairs" have some overlaps. However, results trained with augmentation obviously have less overlap. Comparing Figure 4.8 and 4.9, these three activities and "cycling" data cluster more closely in Figure 4.8 than the distributions in Figure 4.9. Observing Figure 4.10, the model trained with augmentation seems to have a better cluster. For the results of 4.11, the interesting thing that catches our attentions is the boundary of each cluster is clear in both sub-pictures. Maybe this situation can explain this dataset can achieve a good classification performance without data augmentation. In the left sub-picture of Figure 4.12, activity "laying", "lie-to-stand" and "sit-to-lie" are grouped separately from the other activities, other activities have some overlap. By comparison, the model trained with augmentation can distinguish activ-

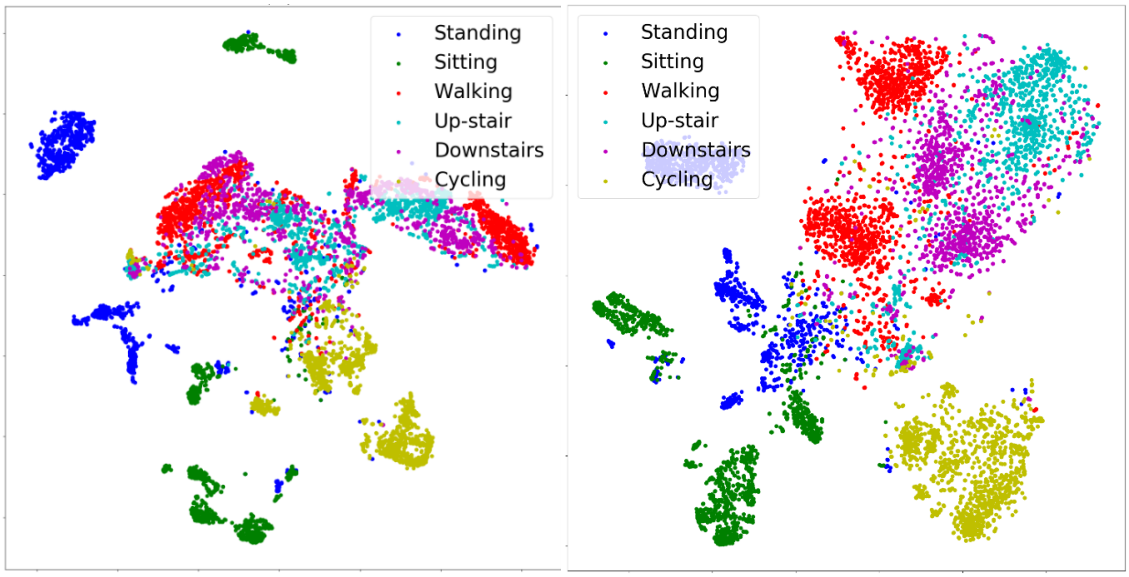


Figure 4.9: t-SNE visualization of SW dataset

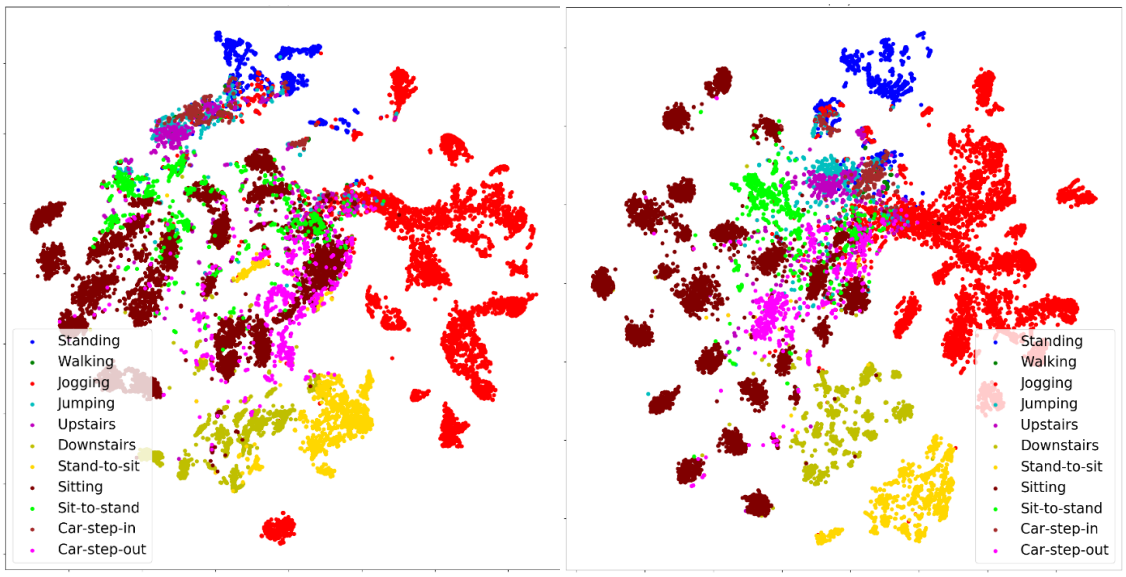


Figure 4.10: t-SNE visualization of MobiAct dataset

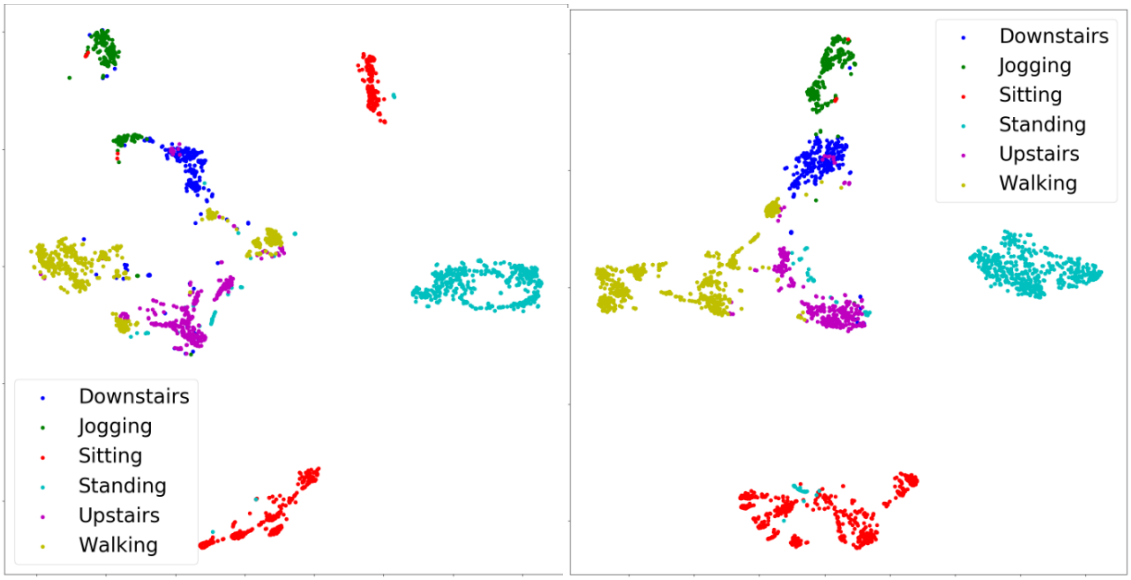


Figure 4.11: t-SNE visualization of MotionSense dataset

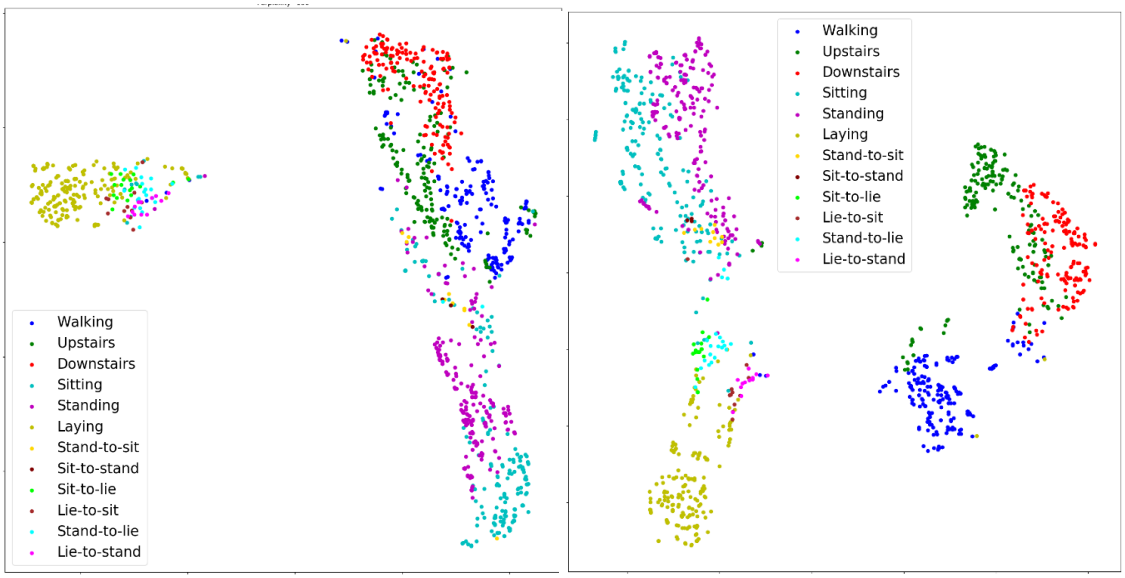


Figure 4.12: t-SNE visualization of HAPT dataset

ities "downstairs" and "upstairs" in a group, "sitting" and "standing" within a group. Activity "laying", "sit-to-lie", "lie-to-stand" and "stand-to-sit" cluster closely.

4.4.9 Experiments on the Imbalanced Dataset

In some classification problems, the bad results are due to imbalanced data. Therefore, we want to verify whether our data augmentation method is effective for this kind of problem. We choose the HHAR dataset to simulate an imbalanced situation. There are six classes in HHAR dataset, which are "Standing", "Sitting", "Walking", "Upstairs", "Downstairs" and "Cycling". The training set size of each class is 5192, 5852, 5852, 4394, 4358 and 4843, and the test set size of each class is 2682, 2755, 3330, 2117, 2482 and 2556. Each time we decrease the amount of one class, the number of training data decreases to 500 and the test data decrease to 250. Thus we trained the model six times, each time there is an imbalanced class. The augmentation policy we used is generated from MobiAct dataset, because it performs the best among all datasets except HHAR itself. We compare the precision, recall, and F1-score to evaluate the performance, we plot the histogram to present results intuitively. From left to right are the classification of precision, recall and F1-score

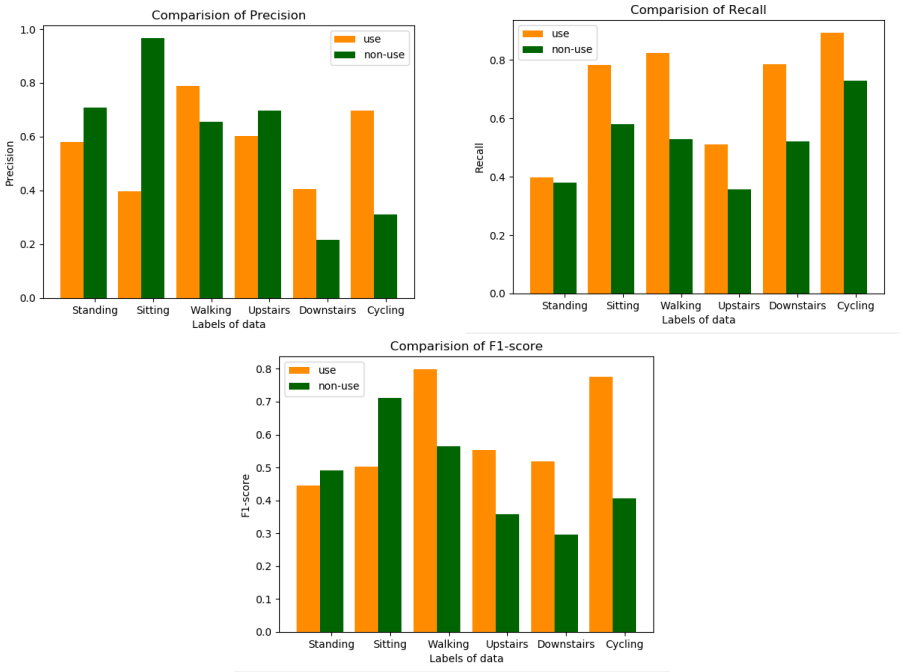


Figure 4.13: Classification results on imbalanced HHAR data

respectively. The dark green bar represents the model trained without augmentation, the orange bar denotes the results of the model trained with augmentation.
We can observe from the Figure 4.13 that using augmentation are all better than non-using augmentation only in recall result. In the precision results, the non-using augmentation model performs better on classes "Sitting", "Standing" and "Upstairs", especially class "Sitting". And because of this significant advantage, in the results of the F1-score, the model without augmentation still performs better than the model with augmentation on "Sitting" class. Using augmentation is also slightly worse than non-using augmentation on "Stand" in the results of the F1-score. Such an unsatisfactory result gives us a direction for the future work. If we want to take advantage of our approach to deal with the problem of imbalanced data, we need to carry out more experiments to verify or adapt our approach.

Chapter 5

Conclusions

In this thesis, an automatic search method was established for time-series data augmentation policies. A series of experiments are carried out to verify the effectiveness of our proposed method. In the following sections, the contributions are concluded, and the future work is discussed.

5.1 Contribution

The research objectives proposed at the beginning are achieved in the end. Through a lot of research, we located the research direction on establishing an automatic system to search the optimal data augmentation policy, which can improve the performance of time-series data classification. Motivated by existing state-of-the-art image data augmentation methods, we proposed the state-of-the-art approach for time-series data augmentation. In addition to realizing search automation, our method also has advantages in the use of search optimal algorithm. At the same time, we also implemented more transformations than currently known the transformations for time-series data to generate synthetic data.
We conducted a series of experiments from multiple angles to verify our approach. We tried to make adjustments to find the best augmentation policies, including changing the evolutionary search algorithms, testing different policy search budgets and adjusting the structure of the policy. After determining the augmentation policies, we used empirical evidence to present the validity and transferability of our augmentation policies. Furthermore, we tested the performance of our augmentation policies in the situation that available instances are fairly few. The final results indicate our augmentation policies can deal with this kind of situation well and prove again the transferability of augmentation policies across domain-specific datasets.
To summarize, we proposed a method that can search time-series data augmentation policy automatically, and our method supports 12 different time-series data transformations, which can guarantee the diversity of synthetic data. Based on our experimental results, the augmentation policy generated by our approach has validity and transferability.

5.2 Future Work

For future work, we can focus on expanding the capabilities of our methods, which means we can carry out more experiments simulating more realistic situations to evaluate the performance. Based on this direction, we put forward two aspects:

- In the last experiment in our previous chapter, our augmentation policies did not show absolute advantages. However, imbalanced data are still the main factor to affect the consequence of the classification in machine learning. Therefore, we advise to optimize our approach so that it can achieve more stable and excellent performance in handling imbalanced data.

- Another aspect we advise is utilizing our approach in the multi-label classification task. Compared to multi-classes classification, multi-label classification is more complicated in the field of deep learning. One reason why it is difficult to achieve very good results is the imbalance of data labels. If there is an optimized approach that can achieve better performance on the imbalanced data, it could perform better in dealing with multi-label classification task.

Bibliography

[1] Charikleia Chatzaki, Matthew Pediaditis, George Vavoulas, and Manolis Tsiknakis. Human daily activity and fall recognition using a smartphones acceleration sensor. In *International Conference on Information and Communication Technologies for Ageing Well and e-Health*, pages 100–118. Springer, 2016. 16

[2] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015. 17, 18

[3] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*, 2012. 5

[4] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018. 5

[5] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019. ixix, 1, 5

[6] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-scale convolutional neural networks for time series classification. *arXiv preprint arXiv:1603.06995*, 2016. 4

[7] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43. Ieee, 1995. 9

[8] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019. 6

[9] Mingyang Geng, Kele Xu, Bo Ding, Huaimin Wang, and Lei Zhang. Learning data augmentation policies using augmented random search. *arXiv preprint arXiv:1811.04768*, 2018. 5, 14

[10] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012. 1

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[12] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data augmentation for time series classification using convolutional neural networks. 2016. 4

[13] Frédéric Li, Kimiaki Shirahama, Muhammad Nisar, Lukas Köping, and Marcin Grzegorzek. Comparison of feature learning methods for human activity recognition using wearable sensors. *Sensors*, 18(2):679, 2018. ixix, ixix, 6, 7

[14] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 18

[15] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 27

[16] Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. Protecting sensory data against sensitive inferences. In *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems*, W-P2DS'18, pages 2:1–2:6, New York, NY, USA, 2018. ACM. 16

[17] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018. 5, 14

[18] Giorgia Ramponi, Pavlos Protopapas, Marco Brambilla, and Ryan Janssen. T-cgan: Conditional generative adversarial network for data augmentation in noisy time series with irregular sampling. *arXiv preprint arXiv:1811.08295*, 2018. 5, 10

[19] J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`, 2018. 14

[20] Jorge-L Reyes-Ortiz, Luca Oneto, Albert Samà, Xavier Parra, and Davide Anguita. Transition-aware human activity recognition using smartphones. *Neurocomputing*, 171:754–767, 2016. 16

[21] Juan Rojo. Machine learning tools for global pdf fits. *arXiv preprint arXiv:1809.04392*, 2018. ixix, 8

[22] Charissa Ann Ronao and Sung-Bae Cho. Deep convolutional neural networks for human activity recognition with smartphone sensors. In *International Conference on Neural Information Processing*, pages 46–53. Springer, 2015. 1

[23] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017. 8

[24] Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. Apac: Augmented pattern classification with neural networks. *arXiv preprint arXiv:1505.03229*, 2015. 5

[25] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003. 5

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 4

[27] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 127–140. ACM, 2015. 15, 24

[28] Terry Taewoong Um, Franz Michael Josef Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. *arXiv preprint arXiv:1706.00527*, 2017. 4, 10, 12

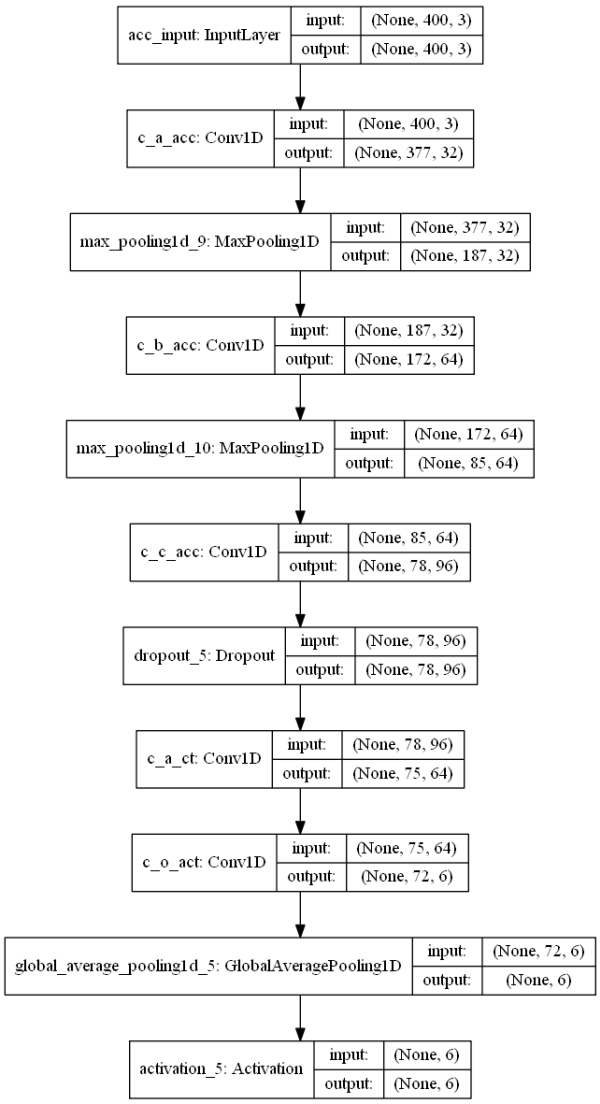
[29] George Vavoulas, Matthew Pediaditis, Emmanouil G Spanakis, and Manolis Tsiknakis. The mobifall dataset: An initial evaluation of fall detection algorithms using smartphones. In *13th IEEE International Conference on BioInformatics and BioEngineering*, pages 1–4. IEEE, 2013. 16

[30] Wikipedia contributors. Deep learning — Wikipedia, the free encyclopedia, 2019. [Online; accessed 31-July-2019]. 6

[31] Mauricio Zambrano-Bigiarini, Maurice Clerc, and Rodrigo Rojas. Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements. In *2013 IEEE Congress on Evolutionary Computation*, pages 2337–2344. IEEE, 2013. 14

Appendix A

The detailed structure of single modality model



Appendix B

Detailed experimental results of data augmentation on a few data

We attached all classification results on different dataset with different number of instances per class in this appendix, section 1, 2, 3, 4 and 5 are results of HHAR, MobiAct, MotionSense, Smartwatch and HAPT dataset respectively.

B.1 Results of HHAR dataset

Policy Source	Precision	Recall	F1-score	Kappa
SW	0.601±0.0472	0.567±0.0538	0.5589±0.0518	0.4805±0.0633
HAPT	0.601±0.052	0.566±0.0438	0.5509±0.0367	0.4796±0.049
MobiAct	0.621±0.0552	0.577±0.0555	0.5619±0.0584	0.4898±0.068
MotionSense	0.5109±0.0924	0.454±0.0851	0.441±0.0922	0.3495±0.1014
Without augmentation	0.38±0.0773	0.368±0.0711	0.332±0.0726	0.2502±0.0826

Table B.1: Classification results of HHAR dataset using 5 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.662±0.0577	0.6319±0.0426	0.616±0.0424	0.5547±0.051
HAPT	0.684±0.0512	0.646±0.0463	0.63±0.0545	0.573±0.0548
MobiAct	0.681±0.053	0.655±0.044	0.6479±0.05	0.584±0.0549
MotionSense	0.562±0.0649	0.5319±0.1002	0.514±0.0945	0.4291±0.1149
Without augmentation	0.5319±0.0799	0.4649±0.0304	0.4359±0.0307	0.3611±0.0349

Table B.2: Classification results of HHAR dataset using 10 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.717±0.0209	0.702±0.0213	0.682±0.0239	0.6383±0.0254
HAPT	0.7489±0.027	0.717±0.0279	0.702±0.0271	0.6583±0.0326
MobiAct	0.744±0.0293	0.717±0.0306	0.705±0.0349	0.6588±0.0372
MotionSense	0.658±0.0655	0.643±0.066	0.624±0.0671	0.5689±0.0797
Without augmentation	0.6±0.0433	0.526±0.041	0.5199±0.0409	0.4339±0.048

Table B.3: Classification results of HHAR dataset using 20 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.775±0.0283	0.751±0.0234	0.7389±0.0221	0.6988±0.0273
HAPT	0.807±0.0236	0.7809±0.0202	0.769±0.0211	0.7339±0.0241
MobiAct	0.788±0.0401	0.766±0.0329	0.757±0.0328	0.7169±0.0386
MotionSense	0.752±0.0227	0.7009±0.0323	0.695±0.0253	0.6416±0.0376
Without augmentation	0.649±0.0372	0.579±0.0385	0.578±0.0409	0.4945±0.0485

Table B.4: Classification results of HHAR dataset using 50 instances per class

B.2 Results of MotionSensen dataset

Policy Source	Precision	Recall	F1-score	kappa
SW	0.835±0.0249	0.816±0.366	0.81±0.0363	0.7727±0.0449
HAPT	0.8779±0.0278	0.8469±0.0464	0.8489±0.0463	0.813±0.054
MobiAct	0.8539±0.02	0.825±0.0382	0.819±0.0415	0.7836±0.0459
HHAR	0.855±0.0329	0.838±0.0359	0.836±0.039	0.7988±0.0441
Without augmentation	0.435±0.1177	0.367±0.0572	0.327±0.0626	0.252±0.0582

Table B.5: Classification results of MotionSense dataset using 5 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.8859±0.0174	0.863±0.0241	0.8619±0.0244	0.8309±0.0292
HAPT	0.9019±0.0213	0.877±0.026	0.884±0.0205	0.8512±0.0316
MobiAct	0.889±0.0225	0.877±0.0303	0.8779±0.0302	0.8485±0.0347
HHAR	0.898±0.0107	0.8779±0.016	0.8789±0.0137	0.8492±0.0188
Without augmentation	0.747±0.094	0.711±0.071	0.695±0.0913	0.646±0.0848

Table B.6: Classification results of MotionSense dataset using 10 instances per class

Policy Source	Precision	Recall	F1-score	kappa
Policy from SW	0.905±0.0102	0.884±0.0205	0.887±0.02	0.8577±0.0251
HAPT	0.9189±0.0144	0.908±0.0193	0.911±0.0186	0.888±0.0238
MobiAct	0.9199±0.0126	0.915±0.0185	0.914±0.0174	0.8924±0.0223
HHAR	0.914±0.0135	0.8939±0.021	0.8959±0.0195	0.871±0.0235
Without augmentation	0.875±0.0196	0.871±0.0254	0.87±0.0268	0.8404±0.0302

Table B.7: Classification results of MotionSense dataset using 20 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.9179±0.0198	0.907±0.0249	0.909±0.0254	0.8881±0.0293
HAPT	0.932±0.0107	0.924±0.0156	0.9299±0.0141	0.9055±0.0182
MobiAct	0.9359±0.0119	0.932±0.0116	0.932±0.0116	0.9162±0.0144
HHAR	0.933±0.0126	0.9199±0.0173	0.9209±0.0175	0.9019±0.0205
Without augmentation	0.923±0.0155	0.9179±0.0193	0.917±0.0195	0.8972±0.0239

Table B.8: Classification results of Motionsense dataset using 50 instances per class

B.3 Results of MobiAct dataset

Policy Source	Precision	Recall	F1-score	kappa
SW	0.7539±0.0215	0.635±0.0628	0.6519±0.0685	0.5569±0.065
HAPT	0.776±0.0245	0.656±0.0724	0.685±0.0713	0.5852±0.078
MotionSense	0.774±0.0335	0.63±0.0725	0.658±0.0755	0.5571±0.766
HHAR	0.771±0.0277	0.6279±0.0774	0.675±0.0805	0.5529±0.0803
Without augmentation	0.4489±0.1609	0.25±0.0824	0.227±0.0971	0.1529±0.0746

Table B.9: Classification results of MobiAct dataset using 5 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.813±0.0205	0.723±0.0407	0.745±0.0387	0.661±0.0451
HAPT	0.83±0.0199	0.74±0.0614	0.762±0.0565	0.6816±0.0668
Motionsense	0.833±0.0109	0.7422±0.0405	0.768±0.0351	0.6844±0.0453
HHAR	0.833±0.0173	0.74±0.0453	0.763±0.0422	0.6808±0.0503
Without augmentation	0.683±0.0494	0.553±0.0588	0.567±0.0694	0.4716±0.0578

Table B.10: Classification results of MobiAct dataset using 10 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.859±0.0122	0.778±0.0559	0.792±0.0552	0.7256±0.0631
HAPT	0.866±0.0101	0.7779±0.0487	0.8±0.0481	0.7272±0.055
MotionSense	0.871±0.0053	0.8109±0.0301	0.8293±0.0241	0.7647±0.0352
HHAR	0.873±0.01	0.818±0.0312	0.836±0.0237	0.7737±0.0348
Without augmentation	0.783±0.0228	0.6769±0.0698	0.691±0.0687	0.6089±0.073

Table B.11: Classification results of MobiAct dataset using 20 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.899±0.0094	0.858±0.0198	0.873±0.0167	0.8234±0.025
HAPT	0.905±0.012	0.858±0.0295	0.874±0.0261	0.8247±0.0353
MotionSense	0.9019±0.0107	0.8639±0.0228	0.876±0.0185	0.8292±0.0267
HHAR	0.913±0.0064	0.883±0.0141	0.894±0.0111	0.8535±0.0158
Without augmentation	0.855±0.008	0.712±0.0708	0.732±0.0735	0.6542±0.076

Table B.12: Classification results of MobiAct dataset using 50 instances per class

B.4 Results of Smartwatch(SW) dataset

Policy Source	Precision	Recall	F1-score	kappa
MobiAct	0.542±0.0561	0.525±0.0358	0.5109±0.0434	0.4305±0.0411
HAPT	0.581±0.0696	0.569±0.0543	0.553±0.0681	0.4816±0.0638
MotionSense	0.537±0.07975	0.5389±0.0731	0.516±0.0806	0.4442±0.0865
HHAR	0.577±0.0531	0.5609±0.043	0.5479±0.513	0.4708±0.0525
Without augmentation	0.4489±0.1609	0.25±0.0824	0.227±0.0971	0.1529±0.0746

Table B.13: Classification results of SW dataset using 5 instances per class

Policy Source	Precision	Recall	F1-score	kappa
MobiAct	0.626±0.0377	0.61±0.036	0.602±0.036	0.5324±0.0415
HAPT	0.638±0.0354	0.639±0.025	0.619±0.027	0.5631±0.0293
MotionSense	0.605±0.0382	0.5889±0.0506	0.571±0.0543	0.5042±0.0585
HHAR	0.6409±0.033	0.625±0.0385	0.616±0.0338	0.5481±0.0442
Without augmentation	0.5059±0.0915	0.481±0.0781	0.472±0.0852	0.3764±0.0941

Table B.14: Classification results of SW dataset using 10 instances per class

Policy Source	Precision	Recall	F1-score	kappa
MobiAct	0.674±0.0363	0.6519±0.0321	0.647±0.0334	0.582±0.0397
HAPT	0.686±0.0374	0.678±0.0359	0.667±0.0357	0.6125±0.0422
MotionSense	0.6649±0.0441	0.635±0.471	0.631±0.0454	0.5599±0.0555
HHAR	0.669±0.0415	0.6539±0.044	0.647±0.0405	0.5811±0.0524
Without augmentation	0.526±0.0548	0.5289±0.05	0.512±0.0526	0.433±0.0583

Table B.15: Classification results of SW dataset using 20 instances per class

Policy Source	Precision	Recall	F1-score	kappa
MobiAct	0.725±0.0332	0.716±0.0303	0.711±0.0333	0.6579±0.0366
HAPT	0.737±0.0296	0.731±0.0273	0.723±0.0286	0.6759±0.0331
MotionSense	0.707±0.0286	0.696±0.0269	0.686±0.0253	0.686±0.0253
HHAR	0.726±0.018	0.719±0.0164	0.71±0.0184	0.6611±0.0199
Without augmentation	0.565±0.039	0.557±0.0325	0.542±0.0378	0.466±0.0386

Table B.16: Classification results of SW dataset using 50 instances per class

B.5 Results of HAPT dataset

Policy Source	Precision	Recall	F1-score	kappa
SW	0.6869±0.0296	0.53±0.0419	0.569±0.0388	0.4696±0.045
MobiAct	0.692±0.0348	0.557±0.0525	0.592±0.0489	0.4974±0.0562
MotionSense	0.695±0.0241	0.5519±0.0373	0.591±0.0311	0.4945±0.0393
HHAR	0.715±0.0329	0.587±0.0521	0.618±0.0503	0.5292±0.0544
Without augmentation	0.44±0.0692	0.345±0.0294	0.331±0.0403	0.2523±0.031

Table B.17: Classification results of HAPT dataset using 5 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.779±0.032	0.671±0.0385	0.7±0.0306	0.6238±0.0398
MobiAct	0.7979±0.0239	0.714±0.0355	0.7369±0.0346	0.6722±0.0394
Motionsense	0.792±0.0305	0.695±0.039	0.722±0.037	0.6507±0.0314
HHAR	0.8039±0.0283	0.714±0.04115	0.735±0.0398	0.6709±0.0467
Without augmentation	0.647±0.0387	0.516±0.0445	0.535±0.0504	0.447±0.0475

Table B.18: Classification results of HAPT dataset using 10 instances per class

Policy Source	Precision	Recall	F1-score	kappa
SW	0.831±0.0181	0.764±0.0261	0.782±0.0235	0.727±0.0282
MobiAct	0.837±0.0134	0.787±0.0249	0.8±0.0228	0.7536±0.0272
MotionSense	0.829±0.0157	0.772±0.0222	0.787±0.0195	0.7363±0.0246
HHAR	0.847±0.0126	0.796±0.0135	0.8089±0.0122	0.7636±0.0156
Without augmentation	0.712±0.0295	0.6119±0.0337	0.632±0.0312	0.5545±0.0365

Table B.19: Classification results of HAPT dataset using 20 instances per class

Policy Source	Precision	Recall	F1-score	Kappa
SW	0.857±0.0078	0.815±0.0185	0.826±0.0156	0.7854±0.0021
MobiAct	0.866±0.0162	0.833±0.0195	0.84±0.0189	0.8062±0.0228
MotionSense	0.862±0.0124	0.826±0.019	0.835±0.0168	0.7985±0.0216
HHAR	0.862±0.0124	0.826±0.019	0.835±0.0168	0.7985±0.0216
Without augmentation	0.7799±0.0154	0.717±0.0219	0.734±0.0205	0.6725±0.0253

Table B.20: Classification results of HAPT dataset using 50 instances per class