BACHELOR

Using HAR On A augmented Reality Application To Identify The End of a Task in Manual Assembly Environment

Kaijim, P.H.M. (Ralph)

Award date:
2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Process Analytics

Using HAR On A augmented Reality Application To Identify The End of a Task in Manual Assembly Environment

Bachelor End Project Report

P.H.M. Kaijim

Supervisors:
Dr. Felix Mannhardt

21-01-2022

Abstract

The topic of this research is the use of Human Activity Recognition on an augmented reality application in a manual assembly environment. The specific question this research is focused on is: Can we reliably classify the end of a high-level task using a RNN-LSTM or CNN-LSTM model based on the little manual assembly data given by KIT-AR? Most related work does not have data from an actual manufacturing environment and focuses on identification of the whole task instead of the end of a task. To tackle the research question the focus of this paper lies on comparing the RNN-LSTM and CNN-LSTM behavior on the little amount of KIT-AR data. The main two steps in solving the research question are a hyperparameter tuning method involving different rounds of a grid search and a laptop-emulation of the model running online on the HoloLens 2. The impact of the research is to determine whether the proposed HAR models are able to be deployed to improve the efficiency in manufacturing environment with a small training dataset.

Contents

Chapter 1

Introduction

1.1 Context and Topic

The topic of this paper is Human activity recognition in a manual assembly process. Human activity recognition (HAR) is identifying human activity from sensor data. There are three different time horizons when HAR activity recognition can occur: predictive, online, and post-mortem. The focus during the project is on online HAR. Online HAR is the act of recognizing the activity while/ very shortly after it happens. HAR can be used on sensor data from a lot of different activities, I can be used to identify a person's movement like running, walking, climbing the stairs but also much more detailed like picking up a screwdriver or twisting this screwdriver. The reason for choosing a manual assembly environment is because in recent years there has been an evident rise in Industry 4.0. Industry 4.0 is the recent change in manufacturing technologies by increasing automation and data exchange [12]. Activity Recognition (HAR) can be vital in industry 4.0 due to the high volume of data. HAR get more accurate due to a high volume of data, especially when the tasks and the underlying patterns are less pronounced like for example in a manual assembly environment. HAR can be used to help to evaluate and increase human performance, and thus their overall efficiency in the production process [12]. The manual assembly environment for this project is the assembly of a cabinet using instructions supplied through augmented reality (AR). The augmented reality is used through a goggle that shows the real world and overlays instruction to help the person assemble the cabinet more efficiently than giving them a manual of instructions. The AR would show a clear diagram of the task in hand with a textual explanation and the placement of the different actions. Like screw the screw of type 1 into the board a the marked spot. HAR in this environment can be used to improve the user efficiency by classifying if he makes a mistake during the instruction or by identifying the end of a task. If the model can reliably identify these end of a task, the AR goggles can automatically show the next instructions and catch mistakes if the model and the user identify the end differently then the subject.

This study aims to leverage HAR in a manual assembly environment of the cabinet example as described before. We work during this project on the dataset provided by KIT-AR of the assembly process of a cabinet. The trial KIT-AR used to collect this dataset simulates the industry 4.0 concept where there is an innovative manufacturing technology with a large amount of data and increased automation. HAR can be used on the positional data of the head during the assembly in different ways, but during this project, the focus is on classifying the end of a task (a set of smaller instructions). Recently the focus of HAR has shifted towards neural networks due to the lower degree of domain knowledge needed for the classification using neural networks [3]. The two best performing neural networks from the literature are the RNN-LSTM, and CNN-LSTM [1, 3]. These models specialize in identifying patterns in a series of time or images, so they are fit for the classification task and data at hand. This research aims to determine if these two different neural networks from previous literature can reliably and accurately classify the end of a task based on the data collected and supplied by KIT-AR.

1.2 Research Question

The gap in the research is the different, more nuanced classification problem. Instead of classifying the task during the research, the classification task is to identify the end of a task. The end of a task is the last 10% of a high-level task as displayed by Figure 1.1. Figure 1.1 shows the structure of the high-level tasks and the low-level actions in the data. Most literature focuses [1, 3, 16, 20] on classifying the activity in our case those would be the low-level actions.

Also, another aspect missing from the current literature is a connection to Industry 4.0, especially in the type of data used in different papers. The dataset and its context used in the project are different by being much smaller and collected during manual assembly tasks. The differences bring their own challenges to the project. The research done during this project will focus on the techniques described in the literature but on a dataset with a manual assembly context. Most datasets used in the literature focus on more easily classifiable activities with a larger dataset, compare to the datasets supplied by KIT-AR. The goal of this research is to determine if HAR can be used by

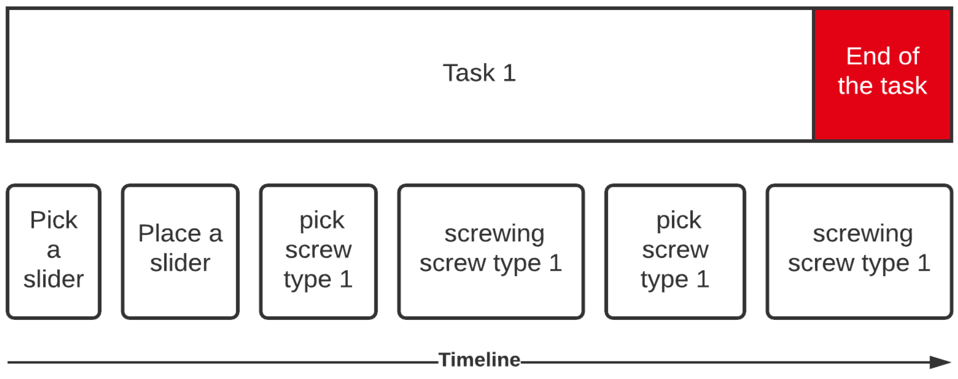


Figure 1.1: The first row of this figure shows the high level task (task 1) over the timeline with the end of the task marked red. The second row of the figure shows the accompanying low-level actions the high level task consist of.

KIT-AR in their AR application on the manual assembly to determine the end of a task. A use-case for KIT-AR for the classification system at the end of our project could be to improve efficiency by the user not having to use voice command. The elimination of voice control can especially beneficial in a environment with a lot of noise from large machinery. If the model is accurate enough the model could also tell the subject that they made a mistake and the task is not finished.

The research question is followed from the GAP in the literature and the goal of this project:
Can we reliably classify the end of a high-level task on the little data given by KIT-AR?

During the answering of the main research question we investigate two model architectures which leads to three sub-question. RQ1 and RQ2 focus on how the RNN-LSTM model and CNN-LSTM model compare in performance and resource cost-effectiveness to each other and the baseline model. RQ3 focuses on if the best model resulting from RQ1 and RQ2 is usable for KIT-AR and thereby answers the final part of the main RQ

Each sub-question brings its results and solves a part of the main research question. The different challenges and steps to solve these sub-questions will be discussed in Chapter 4 and Chapter 5.

1.3 Method

Our approach to investigating the research question is to make a baseline model, a RNN-LSTM model, and a CNN-LSTM model and evaluation and comparing these final models at the end of the project, as displayed in a visualisation of the project pipeline in Figure 1.2.

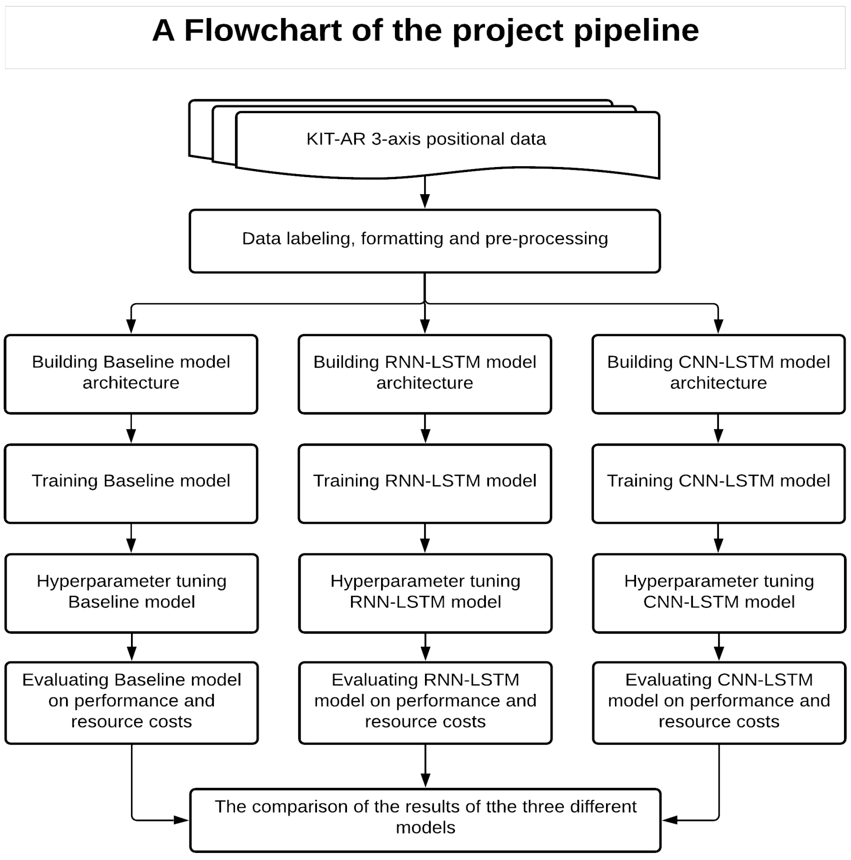

A Flowchart of the project pipeline
KIT-AR 3-axis positional data
Data labeling, formatting and pre-processing
Building Baseline model architecture
Building RNN-LSTM model architecture
Building CNN-LSTM model architecture
Training Baseline model
Training RNN-LSTM model
Training CNN-LSTM model
Hyperparameter tuning Baseline model
Hyperparameter tuning RNN-LSTM model
Hyperparameter tuning CNN-LSTM model
Evaluating Baseline model on performance and resource costs
Evaluating RNN-LSTM model on performance and resource costs
Evaluating CNN-LSTM model on performance and resource costs
The comparison of the results of tthe three different models

Figure 1.2: This Figure illustrates the pipeline used in this project for developing the three different models and answering the research question.

The first step in the pipeline to create any model is getting the data in the correct format after labeling and preprocessing. The data needs to be explored and cleaned from unusable cases. The preprocessed data needs to be labeled.

After the data is labeled, the data needs to be split into different windows and split into a training, validation, and test set. By using the 'Leave-one-user-out' (LOUO) method. LOUO means for this project's data to leave the data from a few selected cases out and divide this data in a validation and test set, which results in testing on unseen data which simulated the real-world implementation. The training, validation, and test split is 80:10:10 due to the smaller size of the KIT-AR dataset. More details on the data split are discussed in Section 4.1.1.

After the data is in the right shape a Baseline model, a RNN-LSTM and a CNN-LSTM models need to be developed. This start by using the architectures and parameter specified in the literature [1, 3]. These starting models are then improved on the KIT-AR data using an experimental method of hyperparameter tuning. Instead of doing hyperparameter tuning using a two-step method. More details and the results of the different rounds of hyperparameter tuning of each model can be found in Sections **??**, **??**, **??** and **??**.

The hyperparameter tuning results in a final Baseline model, RNN-LSTM model and a CNN-LSTM model that need to be evaluated and the results are compared between these models. The metrics used in the evaluation are the F1-score, precision, and recall over a 10-fold cross-validation. More details and the results of the evaluation of these models are discussed in Section 5.4 and Section 5.5.

Besides the performance, the amount of resources each model needs is also crucial for implementing the final model and is part of the model evaluation. The HoloLens has limited resources, especially when considering that the other processes also need to run and take a large amount of the working memory (RAM) and computational power (CPU). Due to us having no access to an actual HoloLens 2, an emulation of the circumstances on the HoloLens 2 is run on a laptop with throttled down CPU to 15% of one core. More details about the processors, conversion formulation and other aspect of the emulation are found in Section 5.3.2.

1.4 Findings

Results

The project results are the evaluation and comparison of the final RNN-LSTM and CNN-LSTM models. The clear observation made during the research is that both models show that the variance of F1-score, recall, and precision on a 10-fold cross-validation is higher than the variance of the training data and the mean is lower then the mean of the training data.

The main observation during the loss-curve evaluation of the CNN-LSTM models is that there are two trends of the loss-curves. The first trend is that the validation loss curves over the different runs are increasing and very volatile. The second trend of the loss curve shows that the validation and training loss curves are more similar. The 10-fold cross-validation displays fewer folds that follow the second trend. When comparing the loss curves over the 10-fold cross-validation of the RNN-LSTM and CNN-LSTM, the first main observation is that the RNN-LSTM model has more fold resulting from the 10-fold cross-validation follow the second trend.

The first observation when comparing the different F1-score, precision, and recall of the baseline model, RNN-LSTM and CNN-LSTM, is that the RNN-LSTM score higher in every metric. The CNN-LSTM scores higher than the baseline except in recall, where the score is approximately even. The analysis of the resources needed to run the models online revealed that: The RNN-LSTM model has a higher time it takes per prediction but needs a lower amount of working memory. The observation for the CNN-LSTM is vice versa (faster predictions, needs more working memory).

Interpretation

The main observation was that the RNN-LSTM model outperformed both the baseline and the CNN-LSTM models in every aspect. However the all the models were severely overfitting. The clear signs of overfitting of all models were:

- the very large difference in all metrics when comparing the results on the training and the test data
- A large variance in the performance of the model over a 10-fold cross-validation
- The large difference in shape between the training and validation loss-curves of the models show that the model was learning the signal but more the noise

The RNN-LSTM model's loss curve did show slightly more signs of the models starting to separate the underlying patterns (signal) from all the irrelevant details (noise). However, this slight increase in signal recognition is not enough, and the model is still overfitted significantly. All the models are lightweight enough (don't need too much working memory and computational power) to run online on the HoloLens 2 if the assumption defined in Section 1.3 hold.

RQ1 and RQ2 resulted in unreliable and poorly performing models given the dataset and the methods used. Which meant that RQ3 results in the

RNN-LSTM model, which does not suffice the the requirements of the main RQ. Since the model resulting from RQ3 does not meet the requirements set in the main RQ is the answer to the main RQ that it is feasible given the dataset and the methods used. So there is a large GAP in the research for future work trying different techniques to reduce the overfitting and improve the overall performance of the models. The main two directions we would have explored given more time would be the data segmentation techniques discussed in the paper written by Zheng et al. [20] and the OHIT synthetic oversampling technique discussed in the paper written by Zhu et al. [21]. This gap in the research means that the degree to which the research question from Sect. 1.2 has been answered successfully is not as high as desired, but is answered with the given dataset and the methods used.

Chapter 2

Background

2.1 Human activity recognition

Human activity recognition (HAR) is the act of identifying human activity from sensor data. There are three different time when HAR activity recognition can occur: predictive, online and post-mortem. Online HAR is the act of recognising the activity while/ very shortly after it happens. Predictive HAR is trying to predict the next activity from the current sensory data. post-mortem HAR is classification of human activity using both data from the past and future data. HAR can be used in all different field. The first field to incorporate HAR recognition were the eldercare facilities. As explained by Md Zia Udd [16], who said "For instance, smart wearable sensor-based behavior recognition system can observe elderly people in a smart eldercare environment to improve their lifestyle and can also help them by warning about forthcoming unprecedented events such as falls or other health risk, to prolong their independent life". However HAR has also other fields where it can be used like in smart homes or during sport tracking.

The sensor data used in HAR has two main sources, wearable sensors and camera's. In earlier stage of HAR the models that were used are shallow machine learning models like Decision Trees, Support Vector Machines (SVM), and Hidden Markov Models (HMM) [2]. Recently neural network are outperforming the shallow model and require little to no feature engineering in contract to the shallower machine learning models used in the beginning [2]. The main reason why there is a lot of literature in HAR is the main differentiating factor: the type of data and the environment in which the data was collected. The actions have different levels of complexity and are very different. For example small patterns of recognising a certain head movement during a task or identifying walking up the stairs by looking at a obvious pattern in the 3-axis movement data. The focus of this project lies on a manufacturing environment since the recent rise of smart factories and Industry 4.0. Industry 4.0 is the recent change in manufacturing technologies by increasing automation and data exchange. HAR can be very important in industry 4.0 due to the high volume of data and can be used to help to evaluate human performance and thus their overall efficiency in the production process [12].

2.2 Related work

There is a lot of related work in the field of HAR on three-axis movement data but less literature on HAR in a manual assembly workflow due to the just recent rise of the industry 4.0 and accompanying use of HAR on a industry 4.0 environment. The main research that comes close to the research question of this project is the research on HAR using lightweight models on the WISDM dataset [1, 3]. The WISDM dataset [17] is a 3-axis smartphone-accelerometer collected dataset on activities like walking, jogging, walking up and down the stair, sitting and standing. The main difference is the type of tasks completed during the collection of the KIT-AR data and the WISDM data. Besides the differences in the tasks is there also a big difference in the classification task compared to the related work: instead of identifying the different tasks is the classification objectives of this project to identify the end of a task. The paper by Ankita et al. [3] and the paper by Agarwal et al. [1] are the closest in terms of process and how to get to the best lightweight solution for the HAR classification problem of this project. Both papers [1, 3] use a lightweight neural network approach (RNN-LSTM and CNN-LSTM) to HAR on a edge device with a 3-axis movement dataset, just like our project. The paper by Mohsen, Elkaseer and Scholz [12] and the paper by Knoch et al [10] are more focused on a similar environment and use case of the solution to the classification problem. The taxonomy by Mannhardt et al [11] is a adequate way to get overview of HAR in the manufacturing context due to the recent rise of industry 4.0. The taxonomy was also the starting place for the literature research of this paper and gives an insight in the current literature on the topic. In the table below the different related works mentioned before are discussed and the objective of each paper and the connection between this project and the literature.

Autor(s)	Title	Objective
2021 [1]	An Efficient and Lightweight Deep Learning Model for Human Activity Recognition Using Smartphones	Using A CNN-LSTM model for HAR on the WISDM dataset to get a lightweight and accurate model. This paper is the inspiration of the CNN-LSTM model tried in this project. The main difference is the tasks executed and the environment during the data collection.
2020 [3]	A Lightweight Deep Learning Model for Human Activity Recognition on Edge Devices	Using A RNN-LSTM model for HAR on the WISDM dataset to get a lightweight and accurate model. This paper is the inspiration of the RNN-LSTM model tried in this project. The main difference is the tasks executed and the environment during the data collection.
2021 [12]	Industry 4.0-Oriented Deep Learning Models for Human Activity Recognition	To compare a CNN, LSTM and CNN-LSTM model for HAR in more industry 4.0 context. They did use the WISDM dataset so rather less complex tasks. These are lightweight models, but there is no mention of edge devices.
2020 [10]	Enhancing Process Data in Manual Assembly Workflows	In manufacturing, information about human behavior in manual assembly tasks is rare when no interaction with machines is involved. This is very similar to the use case of this project. The data they are using in their paper is collected during a similar assembly as the experiment by KIT-AR. This data is collect using different wearable sensors: the data used is collected using camera's and is image data.

Chapter 3

Problem Exposition

3.1 Detailed Method

The overall methodology used in this project is the CRIPS-DM methodology. CRISP-DM stand for Cross-Industry Standard Process for Data Mining and is an industry-proven way to structure data mining projects. The CRISP-DM process is visualised in Figure 3.1.

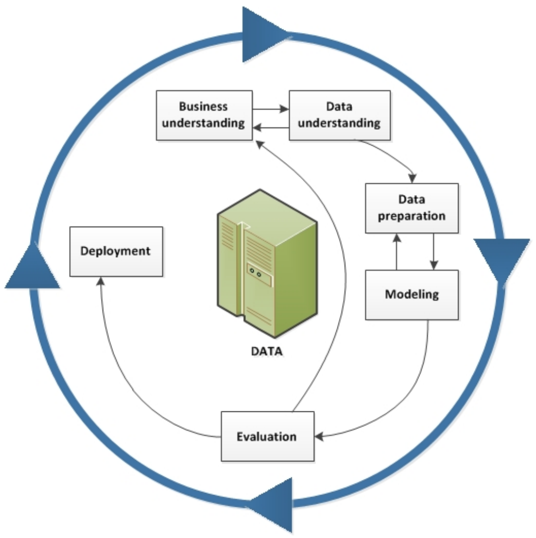


Figure 3.1: A process model of CRISP-DM

Business understanding is the starting point for most data science projects, including this project. The business understanding focuses on understanding the needs of the company that commissioned the project. During the business understanding the business objectives are established, the situation is assessed, the data science goals are determined and the project plan is created. The data understanding is the second phase of the CRISP-DM cycle. The data understanding is an expansion of the project understanding. During the data understanding normally the data is collected and the described. In the current project these step are already done by KIT-AR. The final two tasks during the data understanding phase is data exploration and the identification of potential problems in the data. The potential problems are related to data quality. The next phase in the CRISP-DM methodology is data preparation. The first step in the data preparation to handle the data problems identified in data understanding. The next steps in the data preparation are the data generation and data formatting steps for the modelling phase. After data preparation is the modeling phase. The modeling phase is the phase were your build and asses different models. To build these models, modeling techniques have to be selected based on literature. The second step is actually building the model and the final step is assessing these models. According to the CRISP-DM data frame these last two steps of build and evaluating the models have to be repeated until the best model is found. However, most of the time there is a time limit on a project and you find the best possible model and see if this model performs good enough. The time limit in this project leads to the same approach of finding the best model(s) and determine if the model(s) perform good enough for the goals of the project determined in the understanding phases. The second to last phase is the evaluation phase. During the evaluation phase is determined if the models meet the business requirement and are ready for the deployment phase. This determination is done by reviewing the work put into the model(s) and summarizing the results. The project goes back to the data understanding if the current model is not ready for deployment. The final phase is the deployment phase were the data science solution to the problem is deployed including a written plan for deployment. This plan will also include the monitoring and maintenance of the solution of the deployment. The final step of the deployment and the whole project is writing a report of the project including a review of the project. The most important characteristic of this methodology is the cyclic nature, especially the back-and-forwards between the data understanding and data preparation stage and between the data preparation and modelling stage. Each and every aspect of the CRISP-DM will appear in this report and will be completed during the project. CRISP-DM is a very flexible and easy to customize for different data science projects. The main difference for the CRISP-DM cycle in this project is that there are three parallel phases of both the modeling and evaluation, one for every model: Baseline model, RNN-LSTM model and CNN-LSTM model. In this project the focus of CRISP-DM are on these modelling an evaluation phases, due to the goal of this project.

3.2 Business Understanding

The first step in the CRISP-DM cycle is the business understanding. KIT-AR is a startup that uses a Augmented Reality (AR) on the manufacturing floor. They aim to lower the cost of poor quality (CoPQ). CoPQ consists mostly of prevention, inspection, auditing, rework, repairing, design changes, additional inventory, downgrading, sales returns. In the case study this projects focuses on KIT-ASSIST. KIT-ASSIST is a product which companies can use to improve their manufacturing workflow using AR. In the use case of this project the subject wearing a HoloLens 2 is given step-by-step instructions overlaid on the equipment using AR. The HoloLens 2 is a pair of augmented reality smart glasses developed by Microsoft. The HoloLens 2 has an integrated display which it uses to mix the reality and computer generated holograms (our use-case instruction of an assembly). KIT-ASSIST aims to increase production quality and traceabilty. KIT-AR achieves these goals with the HoloLens 2 which results in that the user never has to search for the current step in the instructions and get less confusing information of adjacent steps. The case study this project and the data is based on is a furniture assembly process from a furniture company using KIT-ASSIST to improve their assembly workflow. Reduction of time to competence (TTC) is another goal of the case study.

3.3 Data Understanding

A very crucial and second step in the CRISP-DM method is the data understanding. The data understanding step starts off with the description of the data source and collection. The next step in understanding the data is the data summary to explain what the different variables represent. The final crucial part in the data understanding is the data exploration, to see the trends and characteristics of the data and check if these match up with the business and previous data understanding. The data exploration also helps to identify the areas of data preprocessing.

Data source and collection

The data of this project is supplied by KIT-AR. The data was collected in a controlled environment in a laboratory. Every subject was asked to build the same cabinet between two and three time. The placement of all the parts and tools where identical in every assembly. An simple overview of the experimental setup is displayed in Figure 3.2.

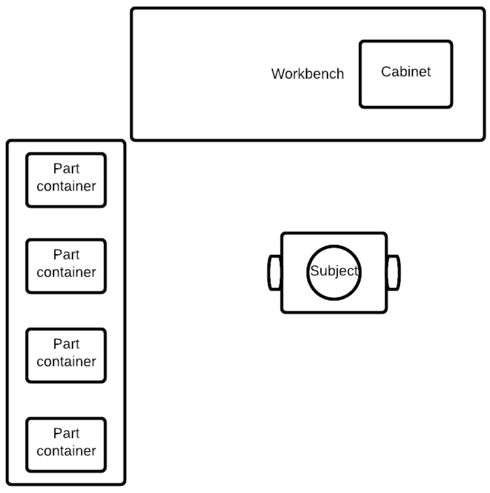


Figure 3.2: Top-down view of experimental setup

The data is collected by giving the participants 13 steps to assemble the furniture. These steps are called high-level tasks. The positional time series data of these high-level tasks are observed by the KIT-AR system. The second dataset is from the low-level actions. These low-level actions are the actions performed by the subject when performing the high-level tasks and are observed by a human annotator. The reason low-level tasks are defined by a human annotator is, because the HoloLens 2 just knows which task the subject should be on according to the instruction schedule. If the subject tries to understand something, take a brake or does the wrong thing, the HoloLens 2 will still label that in the data under the task you should be on. Besides possible mislabeling is there also a lack of detail in the high-level tasks, because there are different steps the subject needs to accomplish in each high-level task. There are 13 different high-level tasks and 15 different low-level actions. The focus of this project will mainly be on the high-level tasks dataset, due to the classification task of this project which focuses on the end of a high-level task which does not warrant the use of the low-level actions. The low-level data is also not available for the HoloLens 2 during the predictions of new data, because the human labeling of the low-level actions.

Data summary

A very important step in the data understanding is understanding the meaning behind each variable of the data. The data contains 28 unique assembles of the whole assembly sequence. There are 13 high-level tasks and 15 low-level actions. Each of the data frames contains 18 columns with the following semantics:

Name of variables	Explaination	Example
case	Case is the process instance identifier. Cases with the same prefix (two characters) are performed by the same worker. Every user was asked to do between two or three assemblies so the number behind the first two characters are assembly numbers.	An example is 'Ca_1-1', where Ca classifies the subject and the nummers the assembly and the run of this assembly.
label	The label of the low-level or high-level action/task	'task1' until 'task12' and 'GAP' task which signifies the time between tasks. The task are number chronological so the first task is 'task1'
plane	KIT-AR built an environment model during their experiment/ data collected and recorded whether the gaze of the worker (direction of the camera on their head) hit a certain plane in that environment.	Examples of planes defined during data collection are 'board_left', 'slider_left_bottom', 'table_side'. For the areas that are not defined the 'none' plane is used
timestamp	The relative timestamp in seconds in the case	The timestamp is generates with an average sampling frequency of 17Hz and significant until 0.1 nanoseconds. (Example: 0.0339192)
label instance	The label instance is a monotonically increasing instance number of the low or high-level task.	All the numbers between 1.0 and 751.0
PosX, PosY, PosZ	the non-normalised x, y and z coordinate of the camera. The camera on the HoloLens 2 can move along three planes, so the exact location of the head of the subject can be determined with the combination of these variables. Further explanation is visualized in Figure ...	These values are calibrated from a null point: If a subject moves straight to his/ her left, the PosX will decrease. If the subject bends down the reach a part, the PosY will decrease
HeadZx (and so on)	The rotation matrix of the camera. Als of these variables can determine the direction the subject is looking in.	The values of these variables are between -1 and 1 and each axis has its own x, y and z values.

Data exploration and preprocessing

Data exploration is the most crucial step in the CRISP-DM, because this is the first interaction with the actual data. The data exploration started with summary statistics for each variable grouped by each case. This different case names showed that some participants assembled the cabinet three times, some two times and some participants even only one time. For the classification problem of this project this is not very significant, because there is no analysis of the different assemblies by person but in general. The first insight the actual summary statistics showed when looking at the maximum timestamp of each case is about the case names that end with two, for example Ca_1-**2**. All the cases that end with two (indicated red in Figure 3.3) are considerably shorter then the cases that end with one (indicated green in Figure 3.3). Evidence of this considerable difference in length are displayed in Figure 3.3. These cases also start at the positional coordinates where the previous run has ended. Both the length difference and the fact that positional starting and endpoint match up results in the conclusion that these cases are not actually new assemblies but an automatic run started by the HoloLens 2 in directly at the end of the actual assemblies. This conclusion was confirmed by a contact person at KIT-AR. Due to this conclusion, these cases ending with a two are not used in this project.

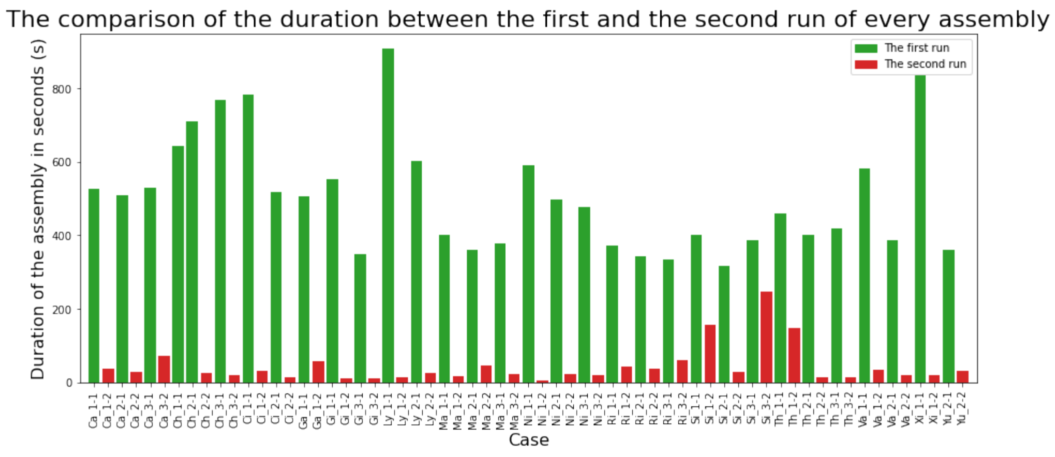


Figure 3.3: Visualizing the different duration of the cases to illustrate the clear difference in all the cases ending with a two compared to the cases ending with a one.

What also became apparent form the summary statistics per case is that the count (total amount of rows grouped by case) of case Ch_1-1 was much lower. This lower count was caused by for this case having a lower sampling frequency. All the average data has a labeling frequency 16.38 Hz. However for one case in the study this was significantly lower: Ch_1-1. In this case there are intervals between data points of up to 30.7 seconds. These gaps in the time series data lead to an average of 2.49 labels per second. The comparison between Ch_1-1 and all the other cases is demonstrated in Figure 3.4. This discrepancy will result in the removal of this case due to the large gaps in the time series data.

Besides understanding the characteristic of the different cases, the most important data is the positional and directional coordinate data, because

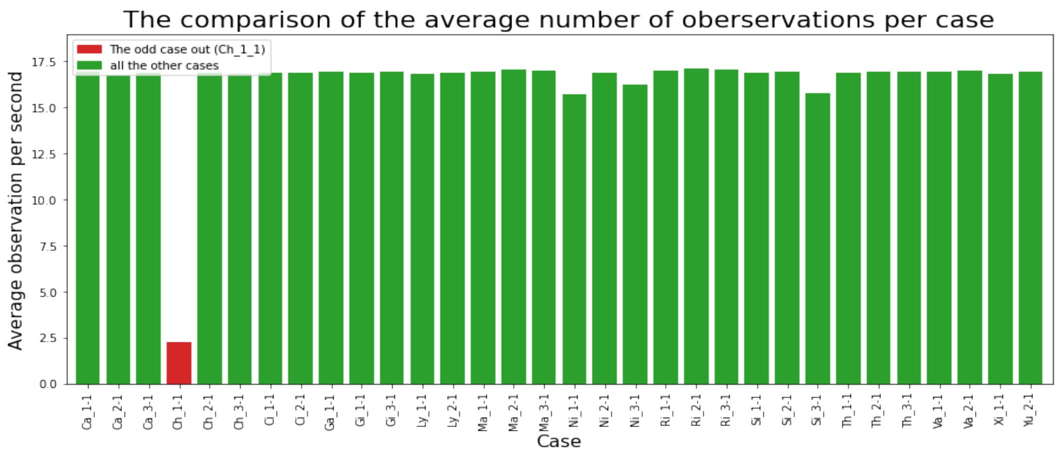


Figure 3.4: The comparison of the sampling frequency between the different cases. The clear standout is the Ch_1-1 which has a considerably lower average sampling frequency.

this will be the HAR input data for the models. The different position the subject can take are defined on the X, Y and Z axis. In The Figure 3.5, these axis are visualized relative to the subject.

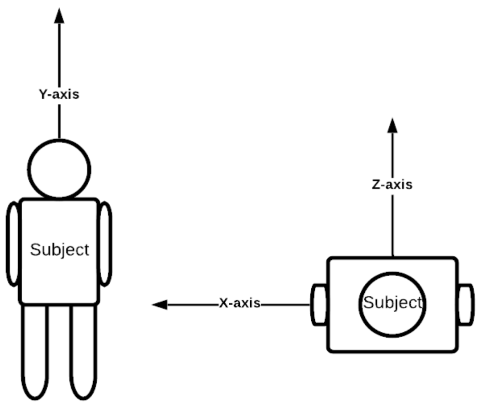


Figure 3.5: Visualizing the different axis relative to the subject

The explanation of the axes in Figure 3.5 also makes sense when looking at the middle plot in Figure 3.6, where the PosY is the variable with the lowest standard deviation out of the different positional axis. This observation is in accordance with the understanding that the height of the head will very the least. The mean of the Y coordinate also increases in the last five tasks.

This increase in the mean height at which the subjects head is positioned is due to that in the final stages of the assembly the cabinet get higher. The whiskers of the box plots of the X-axis are also closer to each other during task two, three, four, five, six and seven. These wiskers are calculated by $\pm 1.5 \cdot IQR$ ($interquartile\ range$) from either Q1 or Q3. So the smaller the wiskers, the smaller the IQR and the smaller the variability of the data. The reason for the decrease in variability of the data of these tasks is that these tasks don't involve parts that are on the table further left of the subject (as illustrated by Figure 3.2).

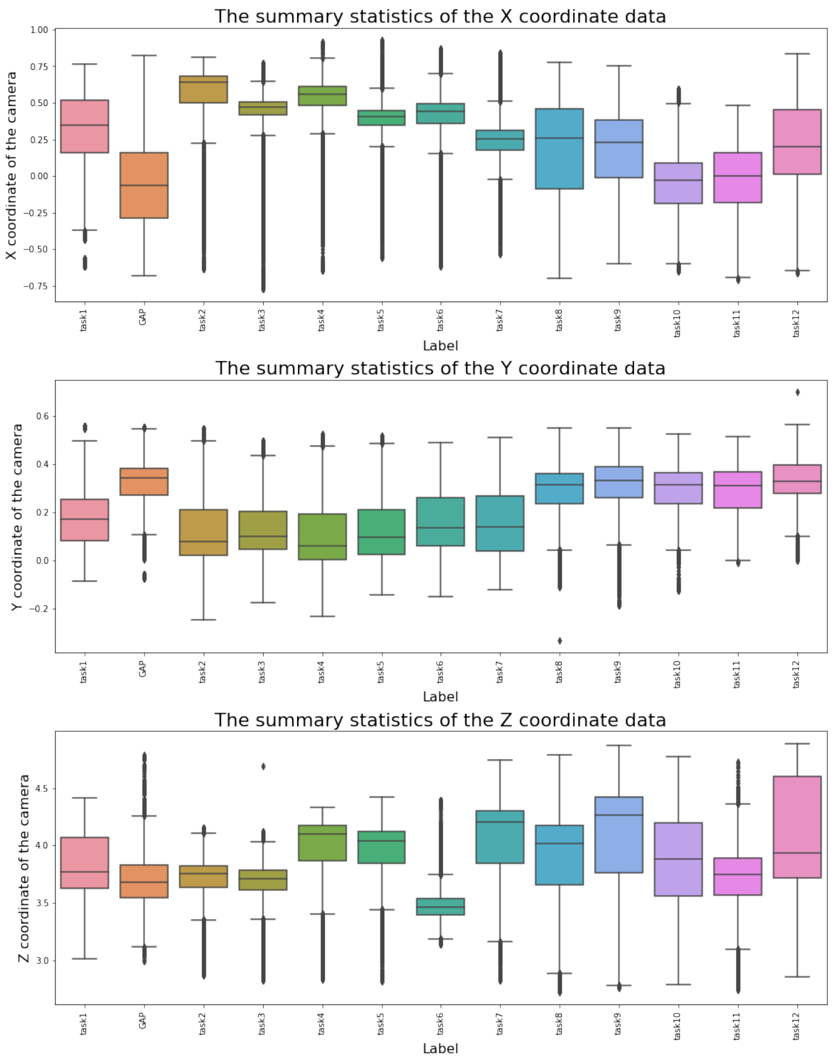


Figure 3.6: The box plots to summaries the different positional coordinate variables (Posx, PosY, PosZ)

The final stop of preprocessing is dealing with missing values. In the data the average of missing values of each case per task is 1.74%. This is a small enough percentage that these rows can be removed from the data, without affecting the data and performance of the model significantly. After this final step the data is completely clean, usable and understood. After the data was fully cleaned, the categorical variables were label encoded. This means that all the label, case and plane data were converted to numbers that correspond with a linked string variable. For example for the label data 0 is 'task1' and for the case data 1 is 'Ch_2-1'. The rows with label 'GAP' indicates the time between two tasks which not an actual task. So due to the classification problem of trying to identify the end of a task and due to the fact that during the 'GAP' periods no tasks are executed, all of these rows are removed from the data.

3.4 Detailed Research Questions

The research question (RQ) formed based on three angles: the literature on the subject of HAR on a edge device, the business understanding of the goals of KIT-AR and the available data in this project. The RQ prompted by the limited amount of data supplied and the interests of KIT-AR. KIT-AR interests are about the aim to lower the CoPQ. This interest leads to KIT-AR wanting a very robust and fast model in order to do online HAR. The literature on the subject of using HAR are most of the time not in the manufacturing environment, due to the fact that data collection about human behavior in manual assembly tasks rare is when there is no interaction with machines during the assembly process [10]. There is, however, more literature on the use of HAR on similar 3-axis movement data collected by wearable sensors [1, 13, 3]. This literature shows that CNN-LSTM and RNN-LSTM are very promising models in the field of HAR on 3-axis movement data from wearable sensors. Due to the online nature of the project and the fact it has to run on a HoloLens 2, the lightweight versions of the CNN-LSTM and RNN-LSTM show the best results on 3-axis movement data in the literature [1, 3]. That is how the two part research question below is formed using the data characteristics, literature and KIT-AR interests. The research question is: Can we reliably classify the end of a high-level task using a RNN-LSTM or CNN-LSTM model based on the little data given by KIT-AR?

The focus of the research question is to improve the KIT-ASSIST process by having the HoloLens 2 detect when a task is finished or about to finish. This will help increase the efficiency of the KIT-ASSIST process and could aid in auditing the tasks the subject thinks he has completed. For example when the model thinks the task isn't nearly finished, but the subject does say it is finished. These improvements could reduce their CoPQ and improve the implementation of KIT-ASSIST. However to achieve this goals the models needs to be reliable. The goal of using a HAR model to detect the end of a task in a manufacturing context together with the literature the leads to the comparison between the two most prominent lightweight HAR models on 3-axis movement time series data: RNN-LSTM vs. CNN-LSTM.

The main RQ leads to three sub-research questions:

RQ1: How does the RNN-LSTM (with the proposed architecture of the literature) compare on performance and efficiency to the other models?

RQ2: How does the CNN-LSTM (with the proposed architecture of the literature) compare on performance and efficiency to the other models?

RQ3: Is the best model resulting from RQ1 or RQ2 implementable for KIT-AR?

The literature showed a good performance in multiple different datasets for both RNN-LSTM and CNN-LSTM, but these datasets were all in a non-manufacturing setting collected and larger then the dataset for this project [1, 13, 3]. This project's comparison is on adjusted RNN-LSTM and CNN-LSTM models for my data set using Hyperparameter tuning and data augmentation.

The discussion around RQ1 and RQ2 start with the model architectures discussed in Sect. 4.2. The models needed to answer RQ1 and RQ2 are then finalized in Sect. 5.3. The answer to RQ1 and RQ2 are discussed in the compared evaluation of the proposed model in Sect. 5.4 and Sect. 5.5. The answers of RQ1 and RQ2 combined will answer RQ3 and are discussed in Sect. 6.1.

Chapter 4

The Three Different Models Used During This Project

The original research question is: Can we reliably classify the end of a high-level task using a RNN-LSTM or CNN-LSTM model based on the little data given by KIT-AR? So the first research problem of the original research question is to create a RNN-LSTM and CNN-LSTM model, which starts by understanding the three models and finding a starting point for their architectures in the literature.

A part of solving RQ1 and RQ2 consists of building a justifiable and straightforward baseline model and evaluating this model. However, the data needs to be formatted before any model can be applied. The following big step towards solving RQ1 and RQ3 is finding a starting point and understanding the RNN-LSTM and CNN-LSTM models. The next step towards finding the best RNN-LSTM and CNN-LSTM models is finding a suitable model architecture and a starting set op parameter to see the models' behavior.

4.1 Starting Requirements For The Model evaluation

The first problem is that there are no results of what a straightforward model would achieve with the same classification task, so we don't know, on a relative scale, if the model is doing well and if it is an improvement. A way to solve this problem is building a straightforward 'baseline' model and evaluating the baseline model to generate a starting point to compare the RNN-LSTM and CNN-LSTM models' performance. The first step in developing the models is to get training, validation, and test data in the right shape.

4.1.1 Data Labeling And Formatting

The formatting started by labeling the target value in the data: for each row, the progress in their task is calculated by using the following formula.

$$Progress = \frac{CTS - STS}{ETS_i - STS_i}$$

CTS = the current timestamp of the row
STS = the timestamp of the start of the current task of the row
STS_i = the timestamp of the start of the current task based on historical data. The i stands for the label of the current task
ETS_i = the timestamp of the end of the current task based on historic data

This calculation would result in the progress based on the completed percentage of the task based on the timestamp of that moment. If the progress is higher than 0.9, the target value is 1, and the task is considered ending. An example of the data after labeling is displayed in Table 4.1. A few rows in the middle of the first task of the first case are displayed. To show the process of labeling, take the row with index 300. The CTS of row 300 is 17.885919 *seconds*, and the STS is 0.0 *seconds* because this is the first task. The ETS_i is ETS_0 and STS_i is STS_0, because the label is 0. The value of ETS_0 is the average time tasks with label 0 end; this is 26.864457 *seconds*. The historical data is used to calculate the ETS_0. The STS_0 is the time tasks with label 0 and case 0 start; this is 0.0 *seconds*.

$$Progress_{300} = \frac{17.885919 - 0.0}{26.864457 - 0.0} = 0.665784$$

After creating the target value, the data needed to be reformed into a sliding window. The size of the sliding window in the literature [1] was 180 observations. The literature was on very similar kinds of data but collected doing a very different task in a non-manufacturing environment and with a different sampling frequency. The data in the literature was collected by using the sensors of a phone while the subject was doing everyday activities like walking, walking downstairs, running, etc. The different window sizes and step sizes will be evaluated during the hyperparameter tuning.

The data is also transformed from data frame to matrices, after which the data is split into a training, validation, and test set, where each data window has its own matrix inside a larger 3D matrix. So the training data will be a three-dimensional matrix where each data window has its own page or sheets as shown in Figure 4.1. Unfortunately, we cannot use standard randomized techniques to split the data into a training set, test set, and validation set

	Timestamp (CTS)	Case (recoded)	Label (recoded)	STS_0 and STS	ETS_0	Progress	Target
296	17.636919	0	0	0.0	26.864457	0.656515	0
297	17.702919	0	0	0.0	26.864457	0.658972	0
298	17.752919	0	0	0.0	26.864457	0.660833	0
299	17.818919	0	0	0.0	26.864457	0.663290	0
300	17.885919	0	0	0.0	26.864457	0.665784	0
301	17.935919	0	0	0.0	26.864457	0.667645	0
302	18.002919	0	0	0.0	26.864457	0.670139	0
303	18.068919	0	0	0.0	26.864457	0.672596	0
304	18.135919	0	0	0.0	26.864457	0.675090	0
305	18.202919	0	0	0.0	26.864457	0.677584	0

Table 4.1: Showing examples of rows after the data has been labeled with the columns used for the labeling. The purpose of this Table is to illustrate how the target value is labeled.

due to the issue of training on future data. Training on future data is caused by data in the training set that occurs later in the time series than data in the test or validation set.

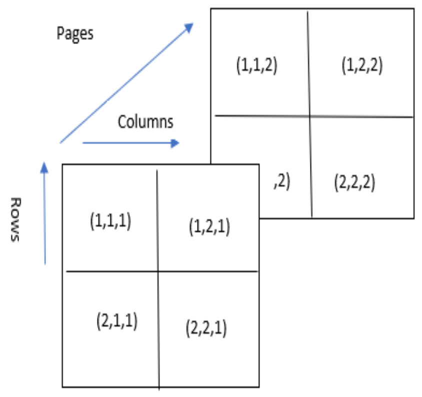


Figure 4.1: A visual representation of a 3-D matrix to illustrate the data structure after the formatting for the modeling phase

There are two different cross-validation schemes to prevent training on future data: 'Leave-one-supertrial-out' (LOSO) or 'Leave-one-user-out' (LOUO) [6]. LOSO would mean to leave all the data from a few randomly selected tasks (labels) out of the training set en divide over the validation and test set. LOSO does not make sense with this project's data because the classification task is not focused on one specific task (label) but on identifying the end of any given task. LOUO would mean for this project's data to leave the data from a few selected cases out and divide this data in a validation and test set. This method is perfect for accurate evaluation because the model will also predict unseen data that it did not use during training. The training-validation-test split is 80:10:10 instead of the normal 70:15:15 based on the literature to counteract the limited training data. The LOUO means that from the 30 different cases, the test data consists of randomly selected 24 cases, and both the test and validation set will have the data from three out of the six other cases.

The data has two inherent problems limiting modeling performance, especially on neural networks:

1. The first and main problem is the lack of training data even after the 80:10:10 split

2. The second problem is the significant imbalance in the data: 89% of the training data has target label 0, and 0.11% of the training data has target label 1.

A possible way to moderate both problems is using random over- and undersampling. Data random over- and undersampling are data augmentation techniques. Random oversampling generates new data by randomly copying already existing data points of the minority class. Random undersampling reduces the imbalance in the data by randomly removing data points from the majority class. Random over- and undersampling can be effective for the machine learning algorithms where the imbalanced data negatively influences the model's fit, which duplicate examples of the minority class can solve. These models that benefit the most are algorithms that iteratively learn coefficients, like artificial neural networks that use stochastic gradient descent. The RNN-LSTM and the CNN-LSTM models compared during this project also use iteratively learn coefficients. Both LSTM models use a stochastic gradient descent optimizer. The paper of Ghazikhani et al. [7] shows that wrapper-based random over- and undersampling outperforms regular random over- and undersampling on almost all datasets they tried. A wrapper-based approach signifies using the model's output to tune the different ways of over-and undersampling. The wrapper-based approach is used in the literature to select a subset of the training data on which the oversampling is applied. Due to the high-dimensionality of the data and the fact that the data is time-series data, standard random sampling is applied instead of the wrapper-based approach. Using random or "naive" over- and undersampling on the complex data speeds up the data augmentation compared to oversampling techniques where synthetic samples are generated. However, in this project, a different wrapper-based approach is used to determine the level of over-and undersampling. Even though random oversampling seems to fix all the problems in the data, there is a big downside to random oversampling: all the duplicate data might cause the model to overfit heavily on the training data. To combat the overfitting, an even data augmentation strategy of over and undersampling is used to generate more balanced data than the 89:11 split of the original data. However, To determine which more balanced split to generate, a wrapped-based approach is used to decide between the following balances in the training data: 50:50, 60:40, 70:30, 80:20. These are the percentages of majority:minority split. An example is if there are 9000 data points with target value 0 and 1000 data points with target value 1, the split would be 90:10, and after oversampling 3000 data points and undersampling 3000 data points, the split will be 60:40. This same strategy is used during the data augmentation of the KIT-AR data to reduce the imbalance of the dataset.

4.1.2 The Baseline Model Concept

The baseline model is a very simple and naive model. The baseline model is a rule based model that solely focuses on the task label and the last timestamp. The rule is based on the same process as the data labeling. The rule is:

$$if \ \frac{current\ timestamp - the\ start\ of\ the\ current\ task}{average\ length\ of\ this\ task} > 0.90$$

$$then\ classification = [0., 1.],$$

$$else\ classification = [1., 0.]$$

Classification [0., 1.] means that the model classifies the data near the end of a task. classification [1., 0.] implies that the model classifies the data as not near the end of a task. The average duration of each task is based on the training data. This baseline model is justified because this simple rule-based model is based on a similar process as the data labeling. The labeling process is solely based on the timestamp task variable and uses the same formula but has all the cases and not just the training cases. An example of how the baseline model would work is that the baseline model would calculate the average duration of each task (label) based on the training data. All the other data is used in the formula can be extracted from the test variables the model is trying to classify. The threshold used for the rule-based model was found by trying different thresholds (an elementary form of grid search hyperparameter tuning on one hyperparameter).

4.2 Understanding of the Proposed Neural Networks

Besides a baseline model for reference and getting data in the right shape, is the next requirement to answer RQ1 and RQ2 an understanding of the RNN-LSTM and CNN-LSTM and their proposed architecture from the literature. Therefore, the following section will focus on understanding the different models and result in a model architecture for the RNN-LSTM model and the CNN-LSTM model.

4.2.1 RNN-LSTM concept

The second sub-problem of the RQ1 is finding the suitable RNN-LSTM model for the project's dataset, classification task, and environment. The approach for this is to use the proposed architecture from the paper [1] as a starting point. Different steps are needed to find the best lightweight RNN-LSTM model. The first step is to understand why LSTM models are chosen in the first case. Then, this understanding is paired with explaining why the model proposed by P. Agarwal [1] is a good starting point and where there are gaps in the paper. Finally, the model needs to be further improved to fit a manufacturing environment and the KIT-AR data.

The research question focuses on using HAR to identify the end of a task. The traditional approach to solve this classification power involves a lot of feature engineering, after which a machine learning model like a decision tree was trained. The main problem with this approach is that the feature engineering needed for the non-neural network models required a high degree of domain knowledge. Recently, neural networks (NN) like LSTMs and variations that use one-dimensional convolutional neural networks or normal CNNs have provided state-of-the-art results on challenging activity recognition tasks with little or no data feature engineering. These NN can use raw data for their feature learning. The reason for choosing a RNN-LSTM model is that training regular RNNs is limited by possible vanishing or exploding gradient problems that hinder the network's ability to backpropagate gradients through long-range temporal intervals [9]. LSTM, on the other hand, uses memory cells, which helps prevent the problem normal RNN have by maintaining information in memory for an extended period of time. The Different gates on the memory cell help control the interaction with the memory cell and the data. An LSTM cell consists of four main parts: the input gate, a neuron with a self recurrent connection, the forget gate, and the output gate. The input gate controls if the data go into the memory cell or not. The output gate can determine if the state of the memory cell influences other neurons. The forget gate can decide if the memory call remembers or forgets the previous state using the self-recurrent connection [5].

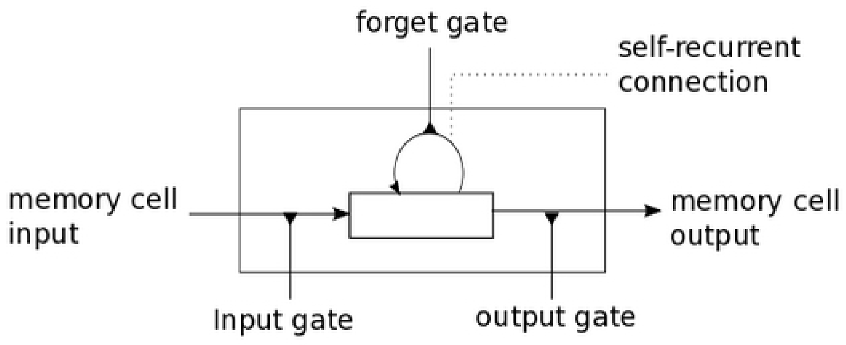


Figure 4.2: A visual representation of a memory cell to clearly explain the different parts and their function and why it is better then a normal RNN cell. This Figure was featured in the paper by Chen et al[5].

The main inspiration and reason for choosing a RNN-LSTM model is the RNN-LSTM model proposed by P. Agarwal [1]. A very similar problem occurs in the paper compared to the research question of this project. The main two differences between this project and the paper are related to the data. In the paper, the WISDM dataset is used. The WISDM data is 3-axis movement data collected by time-series data of very regular actions (Jogging, Walking, Standing, Sitting, Upstairs, and Downstairs) collected using a wearable sensor. The first main difference between the KIT-AR data and the WISDM data is the tasks during the data collection. The KIT-AR data is collected in a manufacturing setting. This difference in tasks can cause the results gathered on the KIT-AR data to vary from the results achieved by P. Agarwal because their movements are less nuanced than the head movement during the assembly of a cabinet. Besides the difference in the settings, the data properties are also very different. The KIT-AR main problems are discussed in Sec.4.1.1, and these problems could inherently limit the performance of the RNN-LSTM model. The final problem not related to the data is that the classification task is different. Instead of classifying the task, this project focuses on classifying the end of any task. However, the type of data, the techniques of HAR, and the use case of the solution needing to run an edge device are similar.

The model architecture of the RNN-LSTM used in the paper by P. Agarwal is a very shallow model with a lightweight approach to accommodate the limited computational power on the edge device (HoloLens 2 in the project's use case). The working of the RNN-LSTM model is visualization in Figure 4.3.

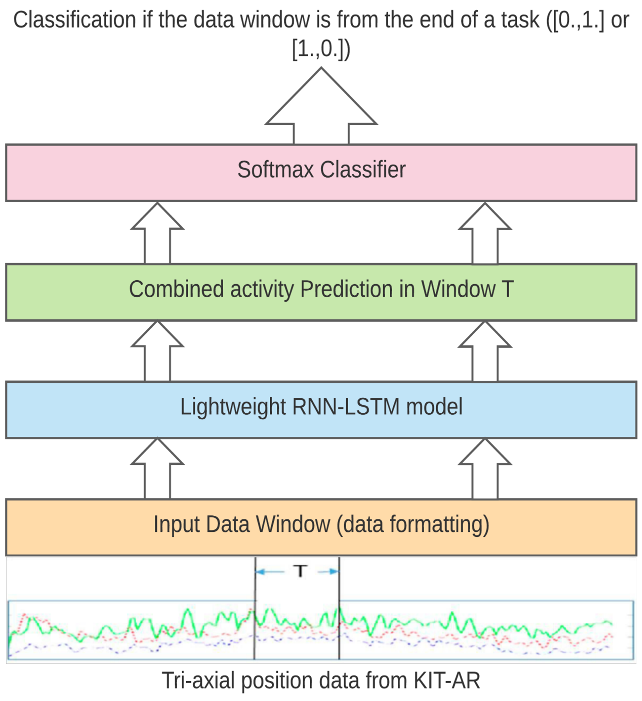

Classification if the data window is from the end of a task ([0.,1.] or [1.,0.])
Softmax Classifier
Combined activity Prediction in Window T
Lightweight RNN-LSTM model
Input Data Window (data formatting)
T
Tri-axial position data from KIT-AR

Figure 4.3: A visual representation of the workings of the RNN-LSTM model used in this paper inspired by the literature of P. Agarwal [1]

The main component of the prediction is the lightweight RNN-LSTM, of which the architecture proposed by P. Agarwal is visualized in Figure 4.4. This architecture will be the starting point of finding the suitable RNN-LSTM model for this project's research question. The architect consists of two hidden LSTM layers with 30 neurons each. The input is time series data of a window with size T. The output of the RNN-LSTM model is a vector of numbers that is converted to probabilities by the softmax function.

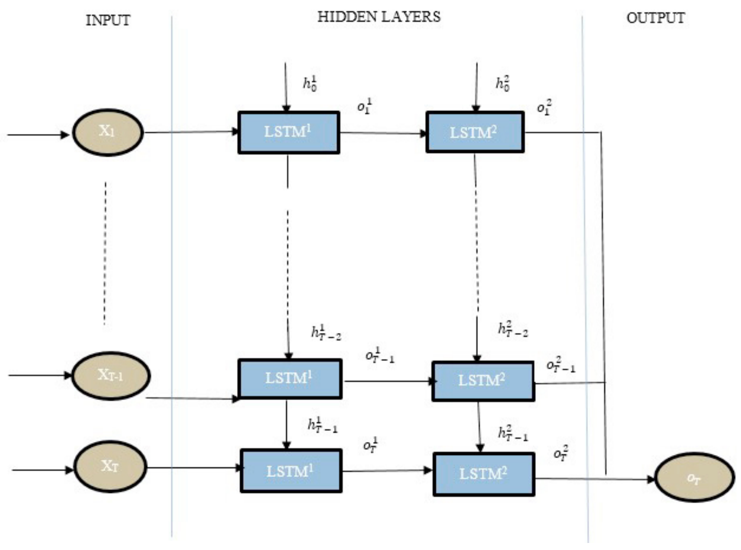


Figure 4.4: A visual representation of the workings of the RNN-LSTM model used in this paper inspired by the literature. The Figure is features in the paper of P. Agarwal [1]

The full model architecture of the RNN-LSTM model for this project is visualized in Figure 4.5. Figure 4.5 shows the different parts of the model described in Figure 4.3. The see through box in Figure 4.5 shows the part of the model described in Figure 4.4.

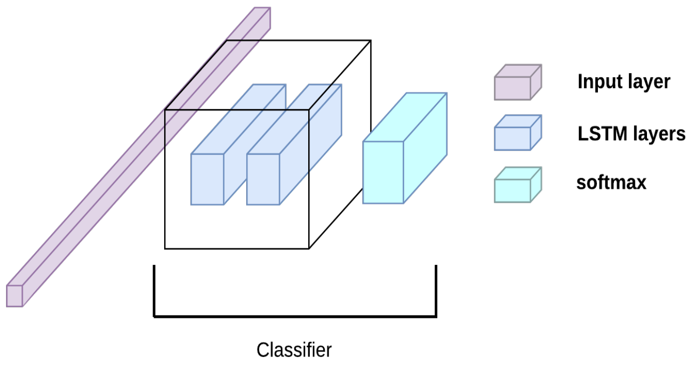


Figure 4.5: The figure shows the RNN-LSTM model architecture by visualising the different layers

4.2.2 CNN-LSTM concept

The second sub-problem of the RQ2 is finding the suitable CNN-LSTM model for the project's dataset, classification task, and environment. A crucial part of making an efficient and lightweight CNN-LSTM model is understanding the CNN-LSTM model. The CNN-LSTM starting model of this project is based on the paper Ankita et al. [3]. The papers [3] and [1] have the same classification problem and the CNN-LSTM model resulted in a slightly higher F1-score of 0.9789 compared to 0.9578. So these models are very comparable, and the CNN-LSTM is just an extension on the RNN-LSTM model architecture proposed by P. Agarwal [1]. The CNN-LSTM architecture is displayed in Figure 4.6. The only difference between the RNN-LSTM architecture is the CNN layers. The architecture displayed in Figure 4.6 is called a Long-term Recurrent Convolutional Network (LRCN) and is referred to as a CNN-LSTM model in the literature. The combination of CNN-LSTM gives an efficient and lightweight deep learning model by reducing the number of features used in the LSTM part of the model.

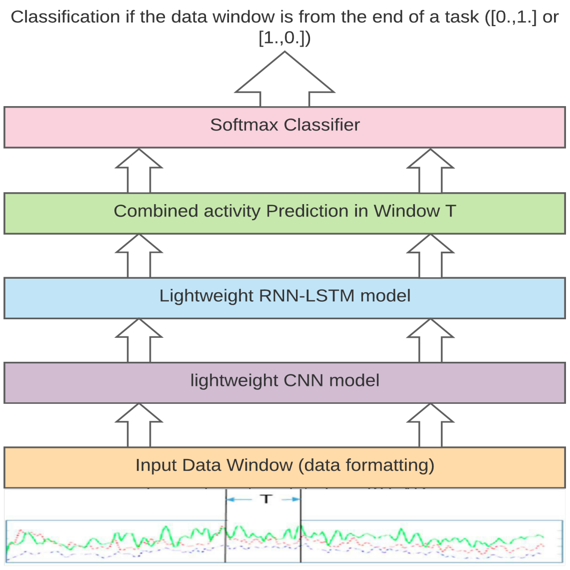

Classification if the data window is from the end of a task ([0.,1.] or [1.,0.])
Softmax Classifier
Combined activity Prediction in Window T
Lightweight RNN-LSTM model
lightweight CNN model
Input Data Window (data formatting)
T

Figure 4.6: A visual representation of the workings of the RNN-LSTM model used in this paper inspired by the literature of Ankita [3]

The CNN part of the model reduces the features using convolutional and pooling layers. These layers extract helpful information and reduce noise using convolution operations to process the sensor data and generate new feature values. The convolution kernel is used as a filter in the convolution operations. The kernel size of the window is used in the convolution and results in a feature map. The literature [19] states that: "These feature maps are typically more useful than the original input data and considerably improve the performance of the overall model". The max-pooling layer following the convolution layers uses a sub-sampling technique that extracts the maximum of a sliding window over the feature map and creates a low-dimensional matrix. The new low-dimensional matrix created by the pooling layer can be considered a concise version of the convolution feature map. Pooling reduces the impact of small changes on the output of the max-pooling layers and makes the model more robust.

The CNN part of the proposed model functions as the feature extractor and consists of two one-dimensional convolutional layers followed by a dropout layer and by a max-pooling layer, as shown in Figure 4.7. The convolution results are flattened, after which the LSTM model (classifier in Figure 4.7) is applied. CNN part of the CNN-LSTM performs feature extraction. Each window of size T is split into four sub-sequences of equal length. The entire CNN model is wrapped in a time Distributed layer to apply the CNN model to each of the four sub-sequences of the window T. The CNN part of the model will extract flattened features from these four sub-sequences, and the LSTM layers are applied to these features. The full model architecture of the CNN-LSTM model for this project is visualized in Figure 4.7. Figure 4.7 shows the different parts of the model described in Figure 4.6.

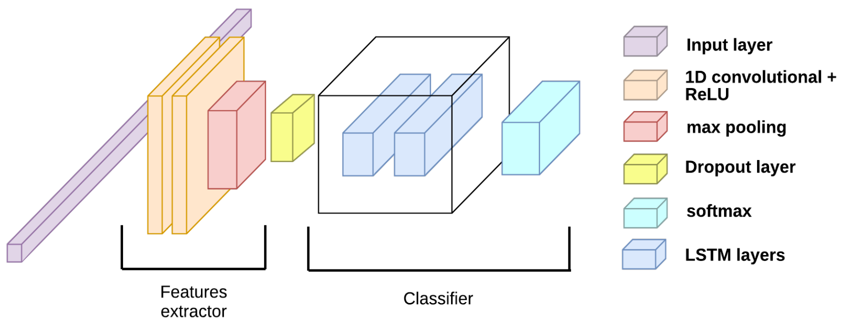


Figure 4.7: The figure shows the CNN-LSTM model architecture by visualising the different layers

The RNN-LSTM and CNN-LSTM model architectures resulting form this section (Section 4.2) are the starting point of the hyperparameter tuning in Section 5.3.1 and evaluated in Section 5.4 and Section 5.5 to answer the main research question.

Chapter 5

Evaluation Of The Different Models

5.1 Objective

The goal of the research question is to find a suitable model out of RNN-LSTM and CNN-LSTM that can accurately predict when a task is ending. This model also has to be suited to run on an edge device. The models are the proposed RNN-LSTM and CNN-LSTM models from the literature [1, 3]. The performance metrics used to compare both improved LSTM models to the baseline are the precision, recall, and F1-score [15]. The reason that these metrics suit our objective is that the data is unbalanced in the target variable.

These metrics use the different types of classification errors in their calculations. The different classification errors start with the true positives (TP), which are the number of correctly predicted task endings. False positives (FP) are the amount of wrongly predicted endings of a task. False positives (FN) is the number of times a task ends, but the model predicts it is not. True negatives (TN) are the number of correctly predicted instances where the task does not end. These different classification errors are visualized in Figure 5.1

- Precision: The amount of time the model correctly predicts a task is ending out of the total amount of time the model predicts a task is ending. Precision is the accuracy solely over the predictions that a task ends. The formulate for precision:

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

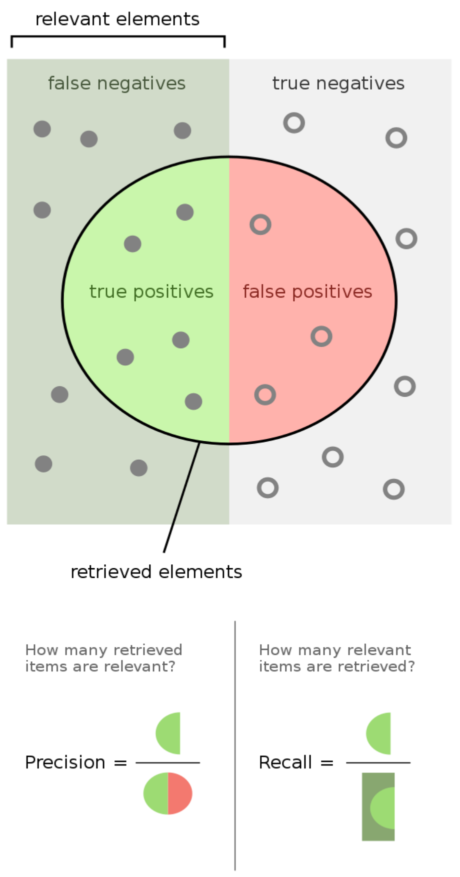


Figure 5.1: A visualization of what recall and precision entail to illustrate why the these matrix are suitable for our evaluation [18]

- Recall: The amount of time the model correctly predicts a task is ending out of the total amount of time the task is actually ending. The recall is the accuracy solely over the instances that a task is actually ending. The formulate for precision:

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

- F1-score: The harmonic mean of the precision and recall:

$$F_1 - score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

  The F1-score will be the primary metric because this combines both precision and recall.

5.2 Setup

In the setup section, a clear overview of all the different aspects will be discussed in detail and every tool and type of data used for the various experiments of the project. The project overview is visualized in the flowchart found in Figure 5.2. The start of the project is the research question and the raw data. The research question requires finding the best possible lightweight RNN-LSTM model and CNN-LSTM model. First the raw data needs to be preprocessed, formatted, and labeled which is explained in Section 3.3 and 4.1.1. The output of this process is a train, validation, and test 3D matrix with a target value of [0., 1.] or [1., 0.]. The train and validation data and a model architecture are used as the input of Experiment 1, as shown in Figure 5.2.

Experiment 1 aims to find the best hyperparameters for the RNN-LSTM and CNN-LSTM models. So Experiment 1 is split into two parallel runs of the same experiment on the different models. These experiments are outlined with a hashed outline in Figure 5.2 and labeled experiments 1.1 and 1.2. This first experiment has the input of: 1) the goal of finding the best RNN-LSTM and CNN-LSTM model, 2) the final training and validation set, and 3) a model architecture from the literature. Both Experiments 1.1 and 1.2 consist of two rounds of hyperparameter tuning, where the first round results dictate the grid of the second round's grid search. The output of both iterations of experiment 1 is a model with a final set of hyperparameters based on the best performance. This analysis is based on the F1-score on the validation data compared to the performance on the training data. This comparison is used to determine the degree of over-or underfitting. Both final models are also the input for the second experiment.

Experiment 2 is an emulation of each model to determine the resources each model needs to run online on the HoloLens 2 (the edge device). The input of the second experiment is the final RNN-LSTM and the final CNN-LSTM together with 1000 data points of the validation data. The laptop used during the emulation uses an Intel i7 core at 15% capacity of one core. The emulation and comparison of the CPUs between the laptop and the HoloLens 2 are discussed in more detail in Section 5.3.2. The emulation records the meantime the model needs to predict on 1000 random data points and uses a formula defined in Section 5.3.2 to output an estimation of the time either model needs per prediction on the HoloLens 2. The second output of experiment 2 is the amount of working memory (RAM) the model needs to run online. The final models resulting from experiment 1 are evaluated by 10-fold cross-validation to determining the performance of the model using the metrics discussed in Section 5.1.

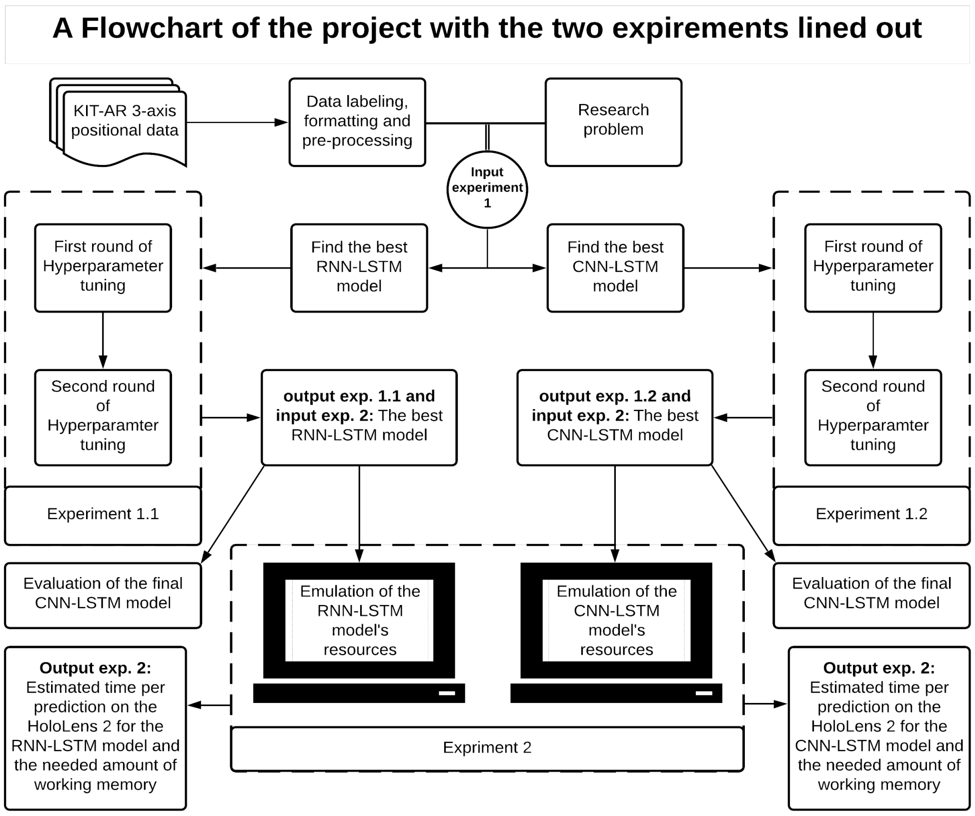

A Flowchart of the project with the two expirements lined out
KIT-AR 3-axis positional data
Data labeling, formatting and pre-processing
Research problem
Input experiment 1
First round of Hyperparameter tuning
Find the best RNN-LSTM model
Find the best CNN-LSTM model
First round of Hyperparameter tuning
Second round of Hyperparamter tuning
output exp. 1.1 and input exp. 2: The best RNN-LSTM model
output exp. 1.2 and input exp. 2: The best CNN-LSTM model
Second round of Hyperparamter tuning
Experiment 1.1
Experiment 1.2
Evaluation of the final CNN-LSTM model
Emulation of the RNN-LSTM model's resources
Emulation of the CNN-LSTM model's resources
Evaluation of the final CNN-LSTM model
Output exp. 2: Estimated time per prediction on the HoloLens 2 for the RNN-LSTM model and the needed amount of working memory
Expriment 2
Output exp. 2: Estimated time per prediction on the HoloLens 2 for the CNN-LSTM model and the needed amount of working memory

Figure 5.2: A flowchart of the projects overview to visualize the different steps of the project and the accompanying experiments and their input and output.

5.3 Execution

The execution section is a clear description of the two different experiments introduced in Section 5.2. The two experiments are the improvement of the proposed model architectures by hyperparameter tuning and the estimation of the resources the models need to run online. The hyperparameter tuning will result in two final models, which will be evaluated on their performance and their resources in Section 5.4 and Section 5.5.

5.3.1 The Hyperparameter Tuning of the Proposed Models

RNN-LSTM improvement methodology and their results

Besides the base architecture, many more hyperparameters influence the model's performance. The paper by P. Agarwal [1] clearly stated these parameters for their classification problem. Still, to transform the model to perform better on the assembly data, these hyperparameters need to be tuned on the KIT-AR data of a manufacturing environment. The process to tune these parameters is called hyperparameter tuning. The hyperparameter tuning method used is a experimental grid search approach. The experimental grid search is an "educated" brute force approach due to the different rounds of parameter tuning. The search grid for the hyperparameter function can be narrowed by experimenting on different parameters based on prior knowledge of the kind of model and patterns displayed during the rounds of hyperparameter tuning. The concept behind the different rounds of hyperparameter tuning is that this reduces the number of initial parameters that need to be checked by reacting to the model's behavior to the initial parameter grid.

Results of the first round of hyperparameter tuning the RNN-LSTM model

The first grid search is over the step size and batch size with the proposed amount of neurons (30) by P. Agarwal in the two LSTM layers. The number of epochs is limited by an early stopper, which prevents overtraining by monitoring the trend of the valuation loss. A grid search is used instead of a random search to reduce how much the different hyperparameters affect each other by checking all possible combinations of parameters and taking the mean and standard deviation. During the hyperparameter tuning, we used three-fold cross-validation to show how robust the models were and decrease the effect of the split between the train, test, and validation data on the model performance.

The first results from the hyperparameter tuning are shown in Figure 5.3. Smaller step sizes seem to better affect the mean F1-score of the RNN-LSTM models and a decreasing standard deviation. A slight standard deviation signifies a more robust model with less volatile performance. The effect of a smaller step size also makes sense because this generates more windows in the data, so more data points. This increase in the data points is a significant advantage when training on a small training dataset. So in the next round of the hyperparameter tuning, the focus is on step sizes smaller than 5.

The second parameter tested besides the step size is the batch size in the

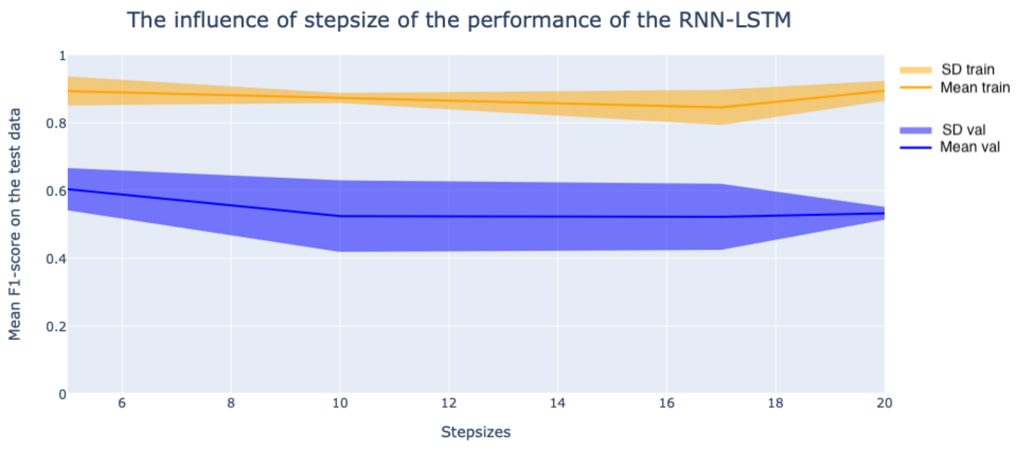


Figure 5.3: The effect of different step sizes on the performance of the model, to show the behavior of the model and see which way to go without having too many different parameter combinations in the grid search. The orange line signifies the mean training F1-Score, and the orange area is the standard deviation. The blue line signifies the mean validation F1-Score, and the blue area is the standard deviation.

first round of hyperparameter tuning. The batch size determines the number of samples the model is shown before adjusting the weights. The performance under the different batch sizes is visualized in Figure 5.4. The larger batch sizes seem to have a better mean F1-score and decrease standard deviation in the model performance on the validation data. However, the effect on the mean performance is minimal on the training data. Larger batch sizes result in faster progress in training but don't always converge as fast. Smaller batch sizes train slower but can converge faster. However, the models all stay well below the maximum epochs of 300 (due to the early stopper) before converging, so they don't have problems converging due to the large batch size.

The window size is the third hyperparameter of the first round of hyperparameter tuning. The effects of different window sizes are visualized in Figure 5.5. The optimal window size is 60, with the best combination of the highest performance and lowest standard deviation. The main factor is the signal length of the ending of a task.

The general insight from the first round of hyperparameter tuning is that the models are significantly overfitting by an average of 0.3226 in F1-score. A potential cause for the overfitting could be that the models were all using

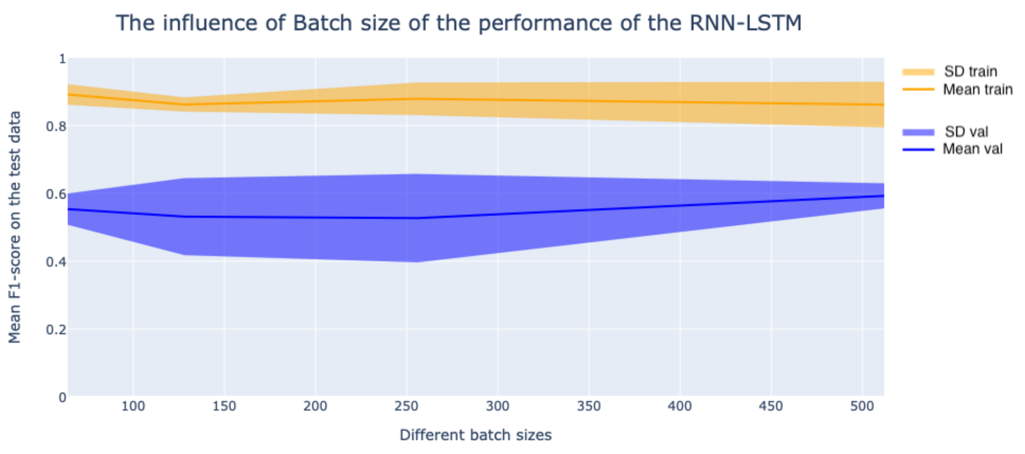


Figure 5.4: The effect of different batch sizes on the performance of the model, to show the behavior of the model and see which way to go without having too many different parameter combinations in the grid search.

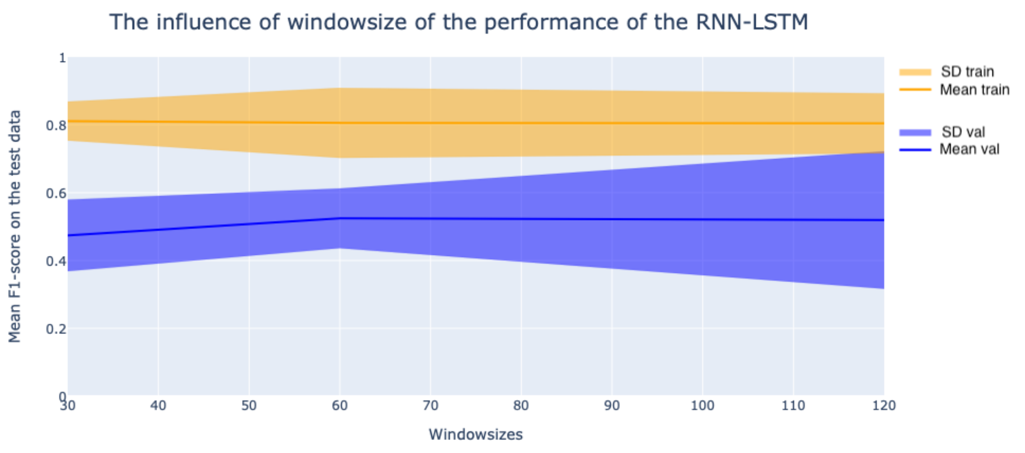


Figure 5.5: The effect of different window sizes on the performance of the model, to show the behavior of the model and see which way to go without having too many different parameter combinations in the grid search.

an over-and undersampling strategy, resulting in a 60:40 imbalance in the data, with an equal amount of over-and undersampling. This large amount of oversampling can cause overfitting.

Results of the second round of hyperparameter tuning the RNN-LSTM model

The first round of hyperparameter tuning showed that a smaller step size and a bigger batch size enhanced the performance, but the models suffered significant overfitting. The second round of hyperparameter tuning focuses on the different over-and undersampling strategies to reduce overfitting. The grid that will be searched during the second grid search focuses on smaller step sizes, bigger batch sizes, and less over- and undersampling. The imbalance in the data before over- and undersampling is: 89% of the windows in the data is of periods where the tasks are not ending, and 11% of the windows in the data is of periods where the tasks are ending. For the first round, this 89:11 split was turned into a 60:40 split with an equal amount of over-and underfitting. The splits tested during the second round are the 70:30 split and the 80:20 split. The results of the different sampling rates are displayed in Figure 5.6. The reduction in data augmentation reduces the overfitting by lowering the F1-score on the training data without affecting the F1-score on the validation data.

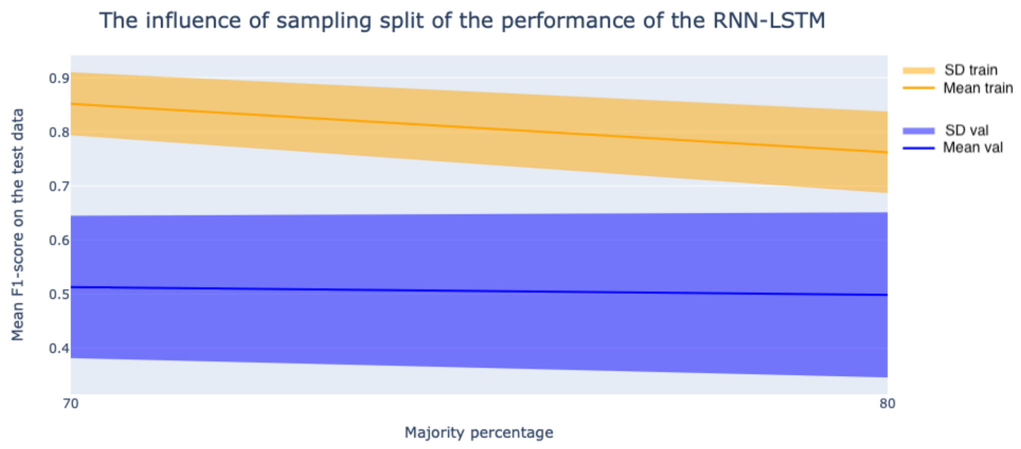


Figure 5.6: The effect of different sampling rates on the performance of the model, to show the behavior of the model and see which way to go without having too many different parameter combinations in the grid search.

Besides sampling, the batch size and step size are further tuned on a grid educated by the first round of hyperparameter tuning. The results of the second round hyperparameter tuning of the different batch sizes are visualized in the line graph in Figure 5.7. The trend is that a larger batch size reduces overfitting by improving the model's generalization.

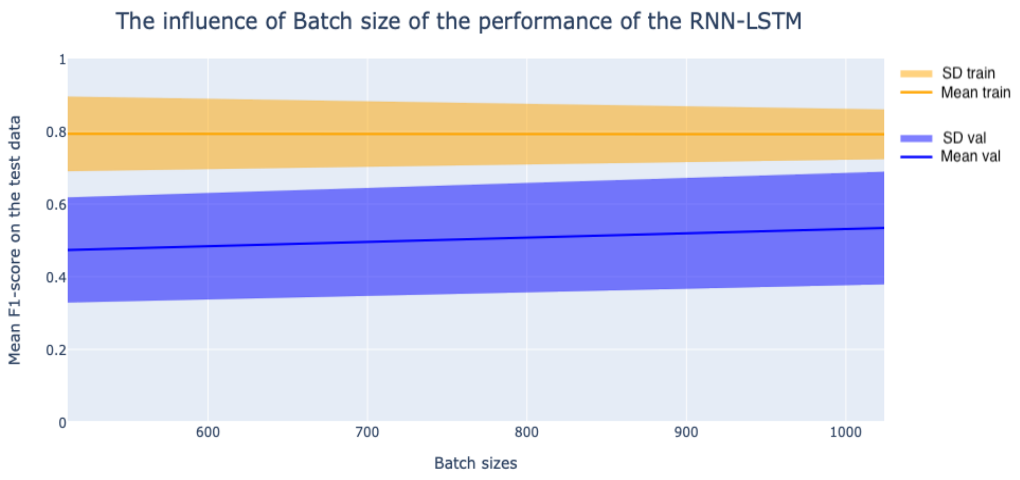


Figure 5.7: The effect of different larger batch sizes on the performance of the model, to show the behavior of the model and see which way to go without having too many different parameter combinations in the grid search.

The first round of hyperparameter tuning also showed that a smaller step size increased the performance of the models. So in the second round, even smaller step sizes are added to the grid search. The results are shown in Figure 5.8. There is no change in performance between a step size of 5 or smaller.

The overfitting is still a problem, so the number of neurons in the two hidden layers (LSTM) is a parameter in the final hyperparameter tuning. All the models before had 30 neurons in each hidden layer, just like the model proposed in the paper written by P. Agarwal. Reducing the neurons and thereby reducing the complexity of the model is a very effective way of reducing overfitting. The results of the effect of the different number of neurons confirm this theory and are visualized in Figure 5.9. This Figure clearly showed an increase in the mean F1-score on the validation data from 30 to 5 neurons, resulting in less overfitting. Lower than five neurons in each hidden layer (3 neurons) result in a model lacking complexity and a lower mean F1-score on the validation data.

To summarize the finding and the different parameters tried during the two rounds of hyperparameter tuning and their results, a summary table of Section 5.3.1 is shown in Table 5.3.1.

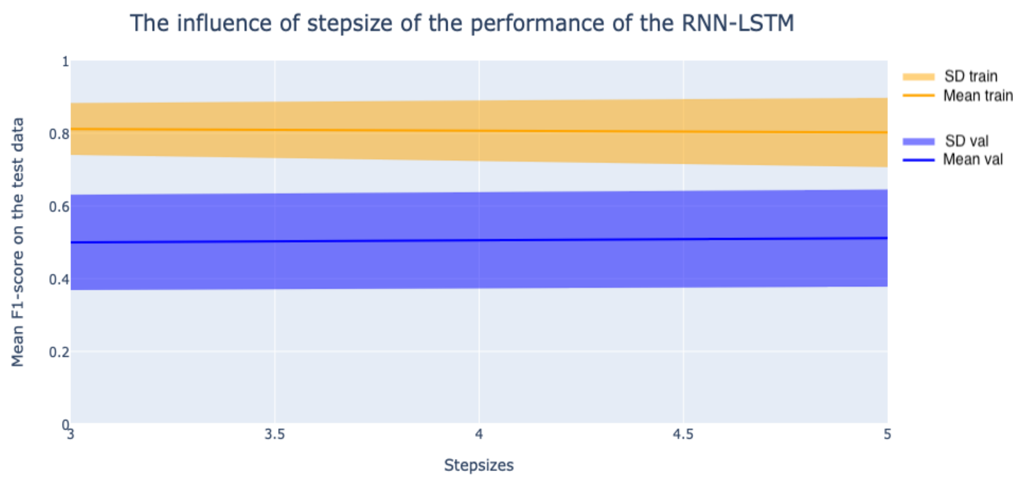


Figure 5.8: The effect of different smaller step sizes on the performance of the model, to show the behavior of the model and see which way to go without having too many different parameter combinations in the grid search.

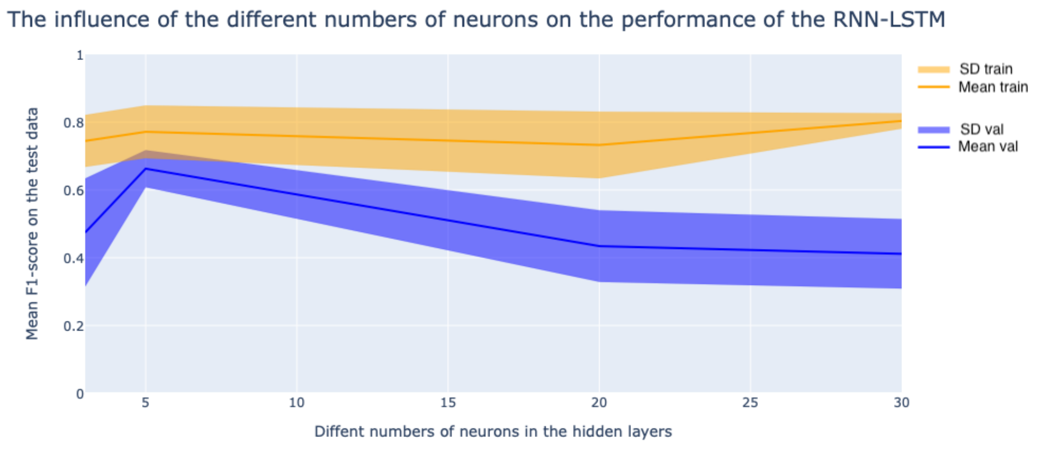


Figure 5.9: The effect of different numbers of neurons in the hidden layers on the performance of the model, to show the behavior of the model and see which way to go without having too many different parameter combinations in the grid search.

Hyperparameter of the RNN-LSTM	Result first round The optimal value:	Result second round The optimal value:	Final results of the Hyperparameter tuning RNN-LSTM
Window size	Tried: 30, 60, 120 Result: 60	Not needed	60
Step size	Tried: 5, 10, 17, 20 Result: 5	Tried: 1, 3, 5 Result: 5	5
Batch size	Tried: 64, 128, 256, 512 Result: 512	Tried: 512, 1024 Result: 1024	1024
Number of Neurons	Tried: 30 Result: 30 (overfitting)	Tried: 3, 5, 20, 30 Result: 5	5
Oversampling	Tried: 60:40 Result: Less oversampling than 60:40 (overfitting)	Tried: 70:30, 80:20, 87:13 Result: 80:20	80:20

Table 5.1: This table gives a clear summary of the different hyperparameters tuned and the resulting hyperparameter for the final RNN-LSTM model.

CNN-LSTM improvement results

Results of the first round of hyperparameter tuning the CNN-LSTM model

The methodology of the hyperparameter tuning is the same as during the improvement of the RNN-LSTM model. The first round of hyperparameter tuning focused on batch size, window size, step size, and the number of neurons in the LSTM part of the CNN-LSTM model. The effect of the window size on the performance of the CNN-LSTM model is visualized in Figure 5.10. The clear optimum of the window size for the highest performance is at a window size of 60. A larger window size seems to decrease the performance of the model.

The next hyperparameter related to the training data format is the step size. The effect of the different step sizes on the model performance is visualized in Figure 5.11. The step size has a lower impact on the performance of the CNN-LSTM. However, a larger step size does cause a higher standard deviation in the results, indicating less robust models. The ideal step size is five or smaller. The smaller step sizes are included in the second round of hyperparameter testing.

The first round of hyperparameter tuning also focuses on the different batch sizes used in training. The effect of the other batch sizes on the grid of the second round of hyperparameter tuning is visualized in Figure 5.12. The optimal batch size is 128, which has the highest F1-score on the test data and less overfitting than larger batch sizes.

The number of neurons in the LSTM model is the final hyperparameter tuned in the first round of hyperparameter tuning. The number of neurons

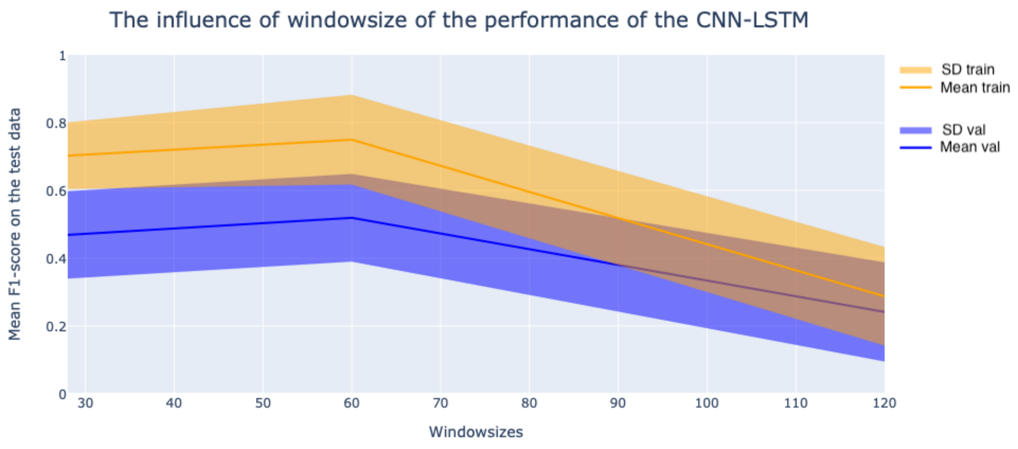


Figure 5.10: The effect of different window sizes on the performance of the model, to show the behavior of the model and see which way to go without having too many different parameter combinations in the grid search. The blue line signifies the mean training F1-Score, and the blue area is the standard deviation. The orange line signifies the mean validation F1-Score, and the blue area is the standard deviation.

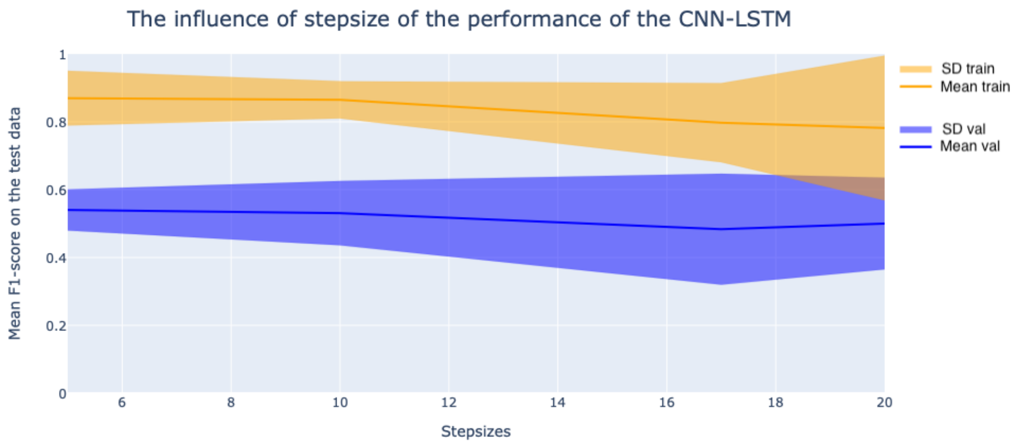


Figure 5.11: The effect of different step sizes on the performance of the model, to show the behavior of the model on the different step sizes.

in the RNN-LSTM model from Sect. 4.2 is 5. However, there are fewer

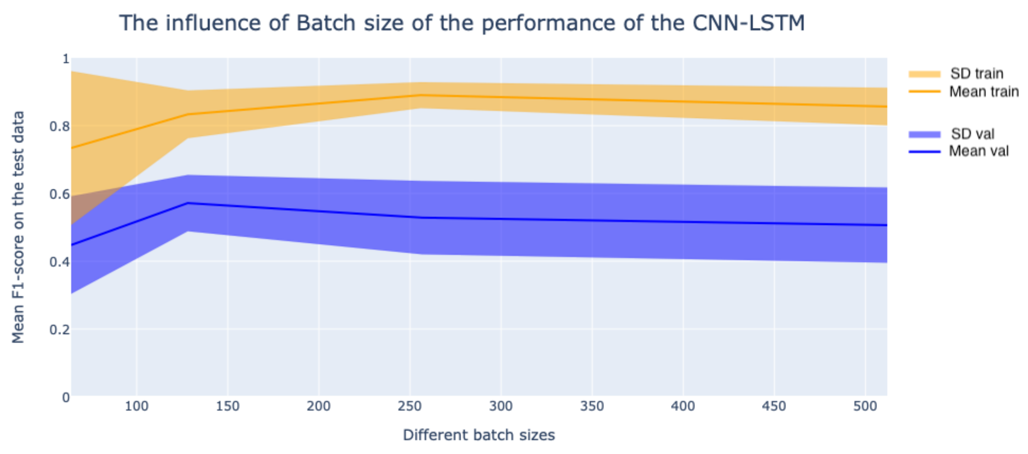


Figure 5.12: The effect of different batch sizes on the performance of the model, to show the behavior of the model on the different Batch sizes.

features used by the LSTM layers in the CNN-LSTM model. So for the first round of hyperparameter tuning, many neurons are tried to gauge the effect of neurons on the performance of the convoluted features. This effect of the different number of neurons is displayed in Figure 5.13. This Figure shows that the best performance is either 50 or 100 neurons in the LSTM layers because the standard deviation of the results with a lower amount of neurons is much higher and suggests a less robust model.

The overall results of the first grid search are that the optimal batch size is 128, the optimal step size is five or smaller, and the optimal window size is 60. However, the model is still overfitting in all different hyperparameter configurations.

Results of the second round of hyperparameter tuning the CNN-LSTM model

Due to the model overfitting after the first round, the second round of hyperparameter tuning focuses on the different pooling sizes and over-and undersampling rates. The tuning on smaller step sizes is also continued in the second round of hyperparameter tuning. The first change that could help solve overfitting is reducing the amount of over-and undersampling. All the models tested in the first round of hyperparameter tuning had an over- and undersampling strategy of a 60:40 split. A 60:40 split is the final split between the amount of training data with no ending as the target variable and the amount of training data with ending as the target variable. can a

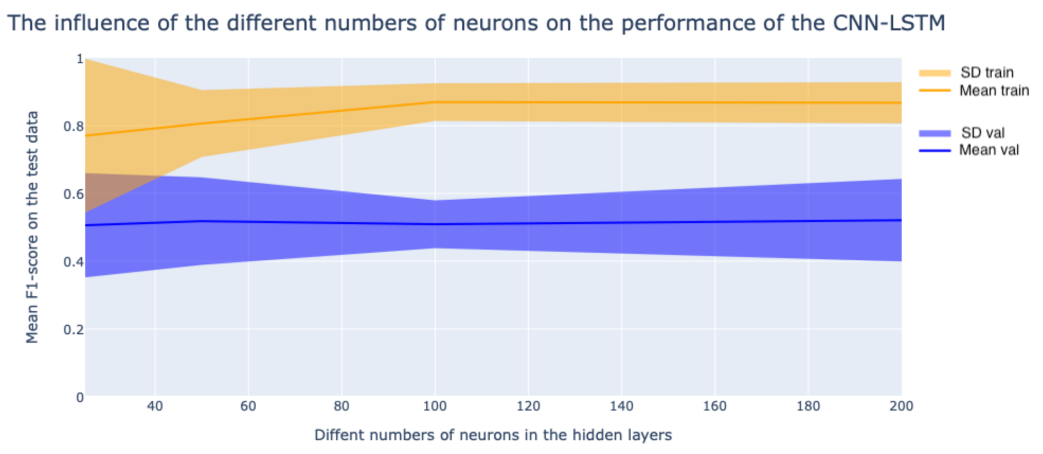


Figure 5.13: This Figure shows the effect of different numbers of neurons in the LSTM layers on the performance of the model.

high amount of oversampling cause overfitting. So in the second round of hyperparameter tuning, 70:30, 80:20, 87:13 are tried to determine the best over- and undersampling strategy. The results of the effect of these different sampling splits are visualized in Figure 5.14. This Figure clearly shows a decrease in the difference between the model's performance on the training and test data between 70:30 and 80:20. After less data augmentation (87:13), the sampling shows a parallel decrease in performance on both the test and training data. So the optimal sampling strategy is over-and undersampling the training data until the data imbalance is 80:20.

The first round of hyperparameter tuning revealed that smaller step sizes are beneficial for the model's performance, so a step size of 5 or smaller is tested in the second round of hyperparameter tuning. The results of this tuning are shown in Figure 5.15. This Figure showed better performance on both the training and test data at a step size of 5.

The final hyperparameter tuned for the CNN-LSTM model is the pooling size of the max-pooling layer. The max-pooling tried is from 2 up to 6. The pooling size determines the size of the window used in the downsampling of the input representation by taking the maximum value in the window. The results of the different pooling sizes for the max-pooling layer are visualized in Figure 5.16. This Figure clearly shows an increase in the performance using pool sizes between 2 and 5, and the model performs worse on a pooling size of 6. The decline also makes sense due to data loss caused by a larger pooling size.

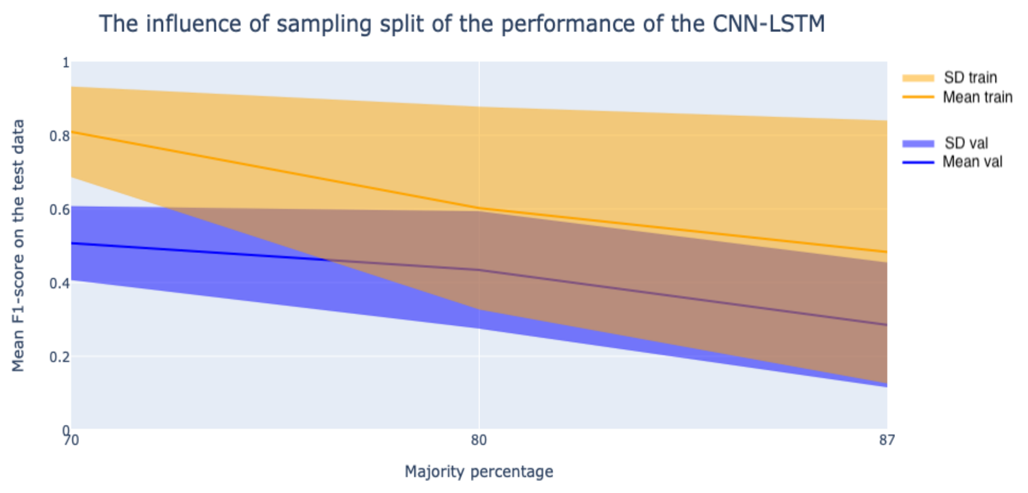


Figure 5.14: The effect of different sampling strategies of the training data on the performance of the model, to show the behavior of the model on the different over- and undersampling strategies.

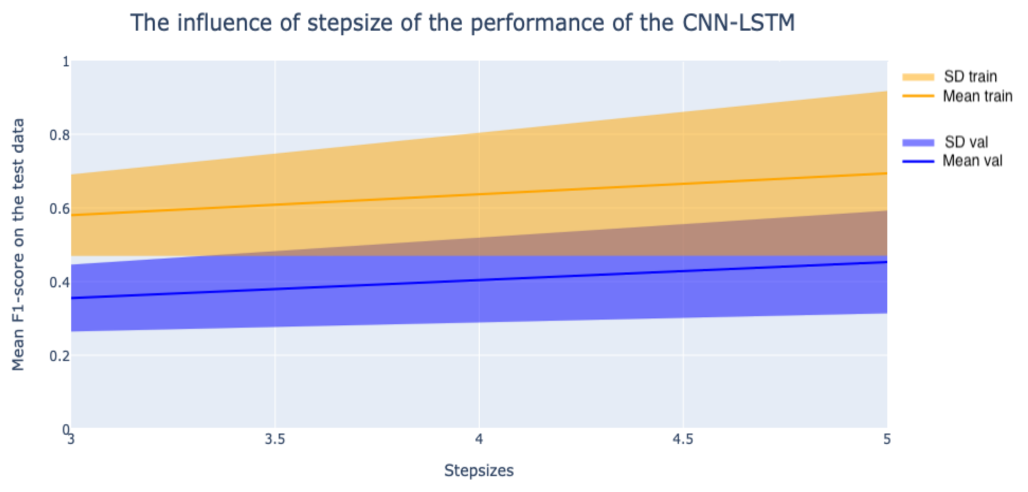


Figure 5.15: The effect of different smaller step sizes of the training data on the performance of the model, to show the behavior of the model on the different smaller step sizes.

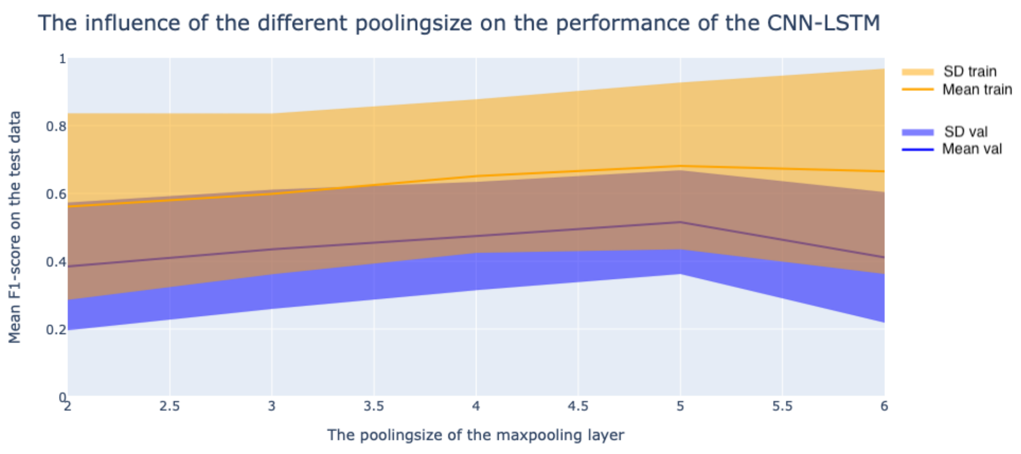


Figure 5.16: The effect of different pooling sizes of the max-pooling layer on the performance of the model, to show the behavior of the model on the different pooling sizes to see the optimal value with the best performance and limited overfitting.

To summarize the finding and the different parameters tried during the two rounds of hyperparameter tuning and their results, a summary table of Section 5.3.1 is shown in Table 5.3.1.

Hyperparameter of the CNN-LSTM	Result first round The optimal value:	Result second round The optimal value:	Final results of the Hyperparameter tuning CNN-LSTM
Window size	Tried: 30, 60, 120 Result: 60	Not needed	60
Step size	Tried: 5, 10, 17, 20 Result: 5	Tried: 1, 3, 5 Result: 5	5
Batch size	Tried: 64, 128, 256, 512 Result: 128	Not needed	120
Number of Neurons	Tried: 30, 50, 100, 200 Result: 50	Not needed	50
Oversampling	Tried: 60:40 Result: Less oversampling than 60:40 (overfitting)	Tried: 70:30, 80:20, 87:13 Result: 80:20	80:20
Max-pooling size	Tried: 4 Result: 4 (overfitting)	Tried: 2, 3, 4, 5, 6 Result: 5	5

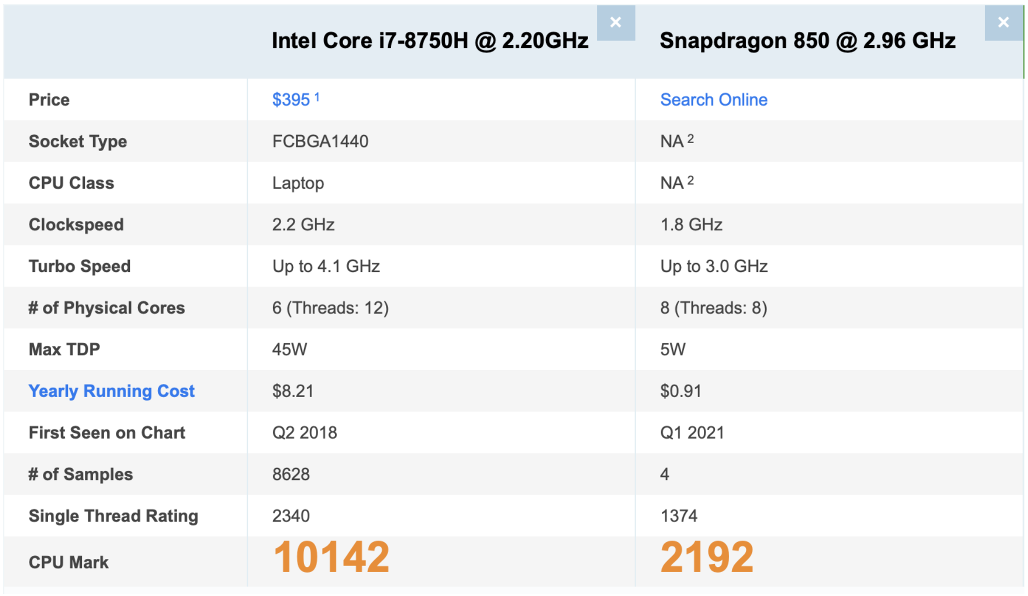
Table 5.2: This table gives a clear summary of the different hyperparameters tuned and the resulting hyperparameter for the final CNN-LSTM model.

5.3.2 The resource analysis of the proposed models

The second aspect that determines how useful a model is the amount of resources the model requires to run online on the HoloLens 2. The HoloLens 2 has a limited amount of resources, and a large portion of these resources are used in different processes. The RNN-LSTM architecture used and the CNN-LSTM architecture used are according to the paper of P. Agarwal [1] and the paper of Ankita et al. [3] lightweight models. However, the hyperparameters have changed, and the papers did not have an analysis of the RAM and CPU needed to run the model fast enough for an online application. The final model needs to run on the HoloLens 2, with limited resources left besides all the other processes required for the instructions and the augmented reality. The main resources that the model need are working memory (RAM) and processing power (CPU). However, since we did not have access to a HoloLens 2 during this project, an emulation on a laptop has to serve as a close estimation of the circumstances on the HoloLens 2. Due to not knowing the resources left on the HoloLens 2, there need to be some assumptions made about the space left. The first assumption is that at least 15% of one out of 8 cores is accessible for running the model; this is 1.875% of the total CPU on the HoloLens 2. The amount of CPU power left for the model will affect the classification speed of the model. However, the classification speeds is also affected by the amount of RAM available to the model. The HoloLens 2 has 4 GB of total RAM, and the assumption is that at least 1.5% of this working memory is left for the model.

The processor used in the HoloLens 2 is a Snapdragon 850, and the processor used during the emulation is an Intel i7 processor. During the emulation of the laptop, the resources of the model will be limited to 15% of a single core to simulate the computational power of the HoloLens 2 and the limited amount of resources. Even by limiting the amount of a single core the CPU used during the emulation, there are still differences in speed and efficiency of these cores. A comparison between the different specifications of the CPU used in the emulation and the CPU in the HoloLens 2 is visualized in Figure 5.17. The clock speed does not solely dictate the speed of the CPU; that is why the single thread rating is essential from Figure 5.17.

The comparison found in Figure 5.17 is completed using the CPU benchmark tool of PassMark software. First, they calculate the single thread rating as follows: "The single thread CPU benchmark, like all processor benchmarks, attempts to estimate how quickly a processor is able to perform a wide variety of calculations. The test issues a series of complex instructions to the processor and times how long the processor takes to complete the tasks. The faster the processor is able to complete the tasks, the higher



	Intel Core i7-8750H @ 2.20GHz	Snapdragon 850 @ 2.96 GHz
Price	$395 1	Search Online
Socket Type	FCBGA1440	NA 2
CPU Class	Laptop	NA 2
Clockspeed	2.2 GHz	1.8 GHz
Turbo Speed	Up to 4.1 GHz	Up to 3.0 GHz
# of Physical Cores	6 (Threads: 12)	8 (Threads: 8)
Max TDP	45W	5W
Yearly Running Cost	$8.21	$0.91
First Seen on Chart	Q2 2018	Q1 2021
# of Samples	8628	4
Single Thread Rating	2340	1374
CPU Mark	10142	2192

Figure 5.17: The comparison of the processor (CPU) used in the HoloLens 2 and during the emulation on the laptop. This comparison in used to argue the differences between the processor and to add strength to this argument by taking the difference into account [14].

the benchmark score" [14]. By comparing the different single thread scores, the conclusion is drawn that one core of the Intel i7 is 1.7 times faster than one core of the Snapdragon 850. During the emulation, 1000 predictions are made using the final RNN-LSTM model, and the mean time each prediction takes is estimated.

To estimate the time per prediction and accounting for the different processor on the HoloLens 2 a calculation needs to be made:

$$PS_{model} = PS_{EMU} \cdot \frac{STR_{EMU}}{STR_{HL}}$$

PT_{model} = the estimation of the prediction time on the HoloLens 2 in seconds per prediction
PT_{EMU} = the prediction time during the emulation on the laptop in seconds per prediction
STR_{EMU} = The single thread rating of the CPU used during the emulation
STR_{HL} = The single thread rating of the CPU in the HoloLens 2

The goal is to keep the PT_{model} below the shortest time a task is ending. Since the shortest task takes 6.611 seconds, the ending of this task consists of

10% of the total duration is the smallest ending of any task: 0.6611 seconds. As long PT_{model} stays below 0.6611 seconds, the model can identify every task ending. So with the assumptions that there is 1.5% RAM available and with the assumption that there is 15% of one core available on the HoloLens 2, we can determine if a model is resource-efficient enough to run on the HoloLens 2.

5.4 Results

The project results evaluate and compare the final RNN-LSTM and CNN-LSTM model to determine if either is good enough to predict the end of any task reliably. The first mean of evaluation the models is by looking at their performance over a 10-fold cross-validation which is shown in Figure 5.18 and Figure 5.19. The light green box plots are the performance on the training data and the light blue on the test data. The boxplots indicate larger IQRs (the box part of the box plot) and larger whiskers, a larger variance in the data used to make them, in this case, the results of the 10-fold cross-validation. The clear observation from both Figure 5.18 and Figure 5.19 is that the variance of the test box plots is higher on both models and in all metrics.

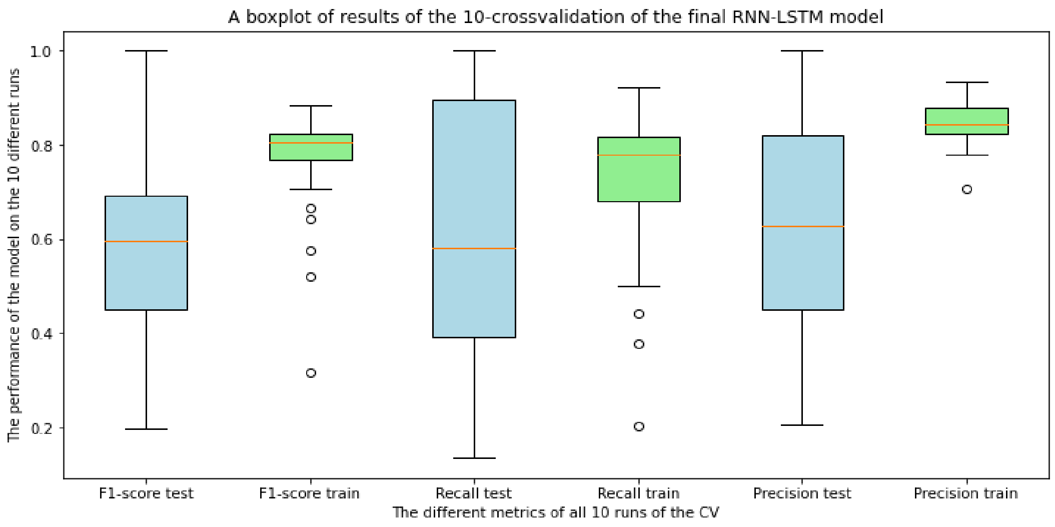


Figure 5.18: The figure visualizes the results of the 10-fold cross validation of the final RNN-LSTM model. The figure shows the difference between the performance of the 10 different folds and the comparison between the performance of the model on training and test data

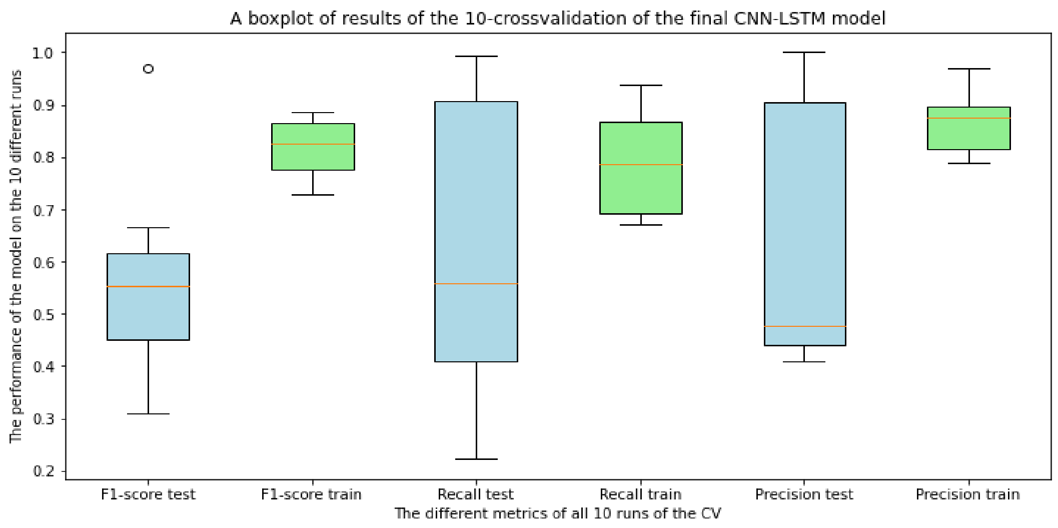


Figure 5.19: The figure visualizes the results of the 10-fold cross validation of the final CNN-LSTM model. The figure shows the difference between the performance of the 10 different folds and the comparison between the performance of the model on training and test data

To identify possible overfitting the loss plots are visualized in Figure 5.20 and 5.21. In these plots, the trends of the loss of the different folds from the 10-fold cross-validation are shown in two subplots. The main observation of Figure 5.21 is two trends in the loss plots, which shows the inconsistency between the different folds of the cross-validation. The left plot shows that the validation loss curves over the different runs are very different from the more traditional training loss curves. The validation losses of example 1 are increasing and very volatile. The second example of Figure 5.21 shows the second trend of loss curves during the 10-fold cross-validation. Two main observations are that the validation and training loss curves are much closer in example 2 and that there are fewer runs in example 2. Another observation of Figure 5.21 is that the starting validation loss of the folds in example 2 is lower than compared to the starting loss of the folds in example 1. If you compare Figure 5.20 and 5.21 the first main observation is that there are more folds out of the 10-fold cross-validation of the RNN-LSTM model with loss curves in example 2 then for the CNN-LSTM model. The example 2 validation curves of Figure 5.20 display a more similar shape to the example 2 training curves then in Figure 5.21. The loss curves of example 1 of Figure 5.20 also are less volatile, and one of the validation loss curves of example 1 looks a lot like an example 2 curve until the rise in loss after 40 epochs. The reason for these trends could be learning of the noise instead of the signal.

In certain folds, the chance of the model learning noise instead of the signal increases due to a large difference in signal between the training and unseen validation data in these folds.

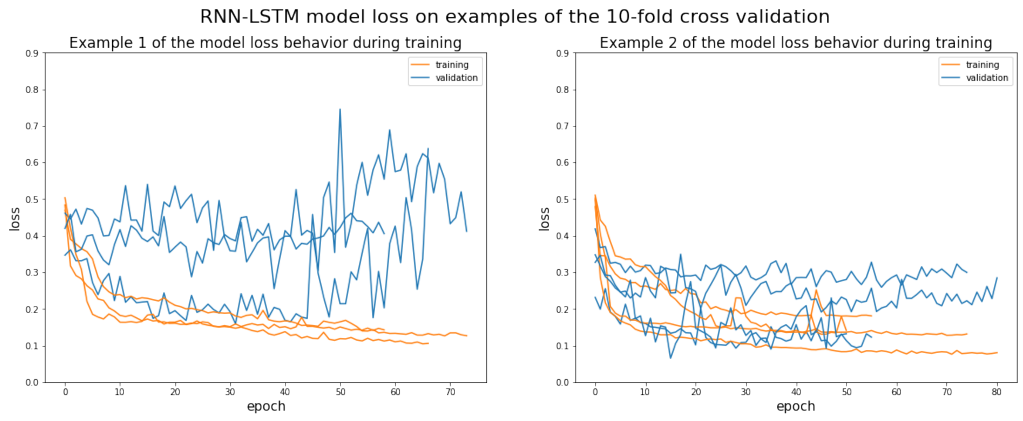


Figure 5.20: The Figure visualizes the loss behavior of the 10-fold cross-validation of the final RNN-LSTM model. The Figure shows two different behaviors of the loss-progression on the validation data during the training of the different folds in the 10-fold cross-validation. Not all 10 folds of the cross-validation are displayed in this Figure due to them having higher peaks in loss which would reduce the interpretability of the trend shown in the plots.

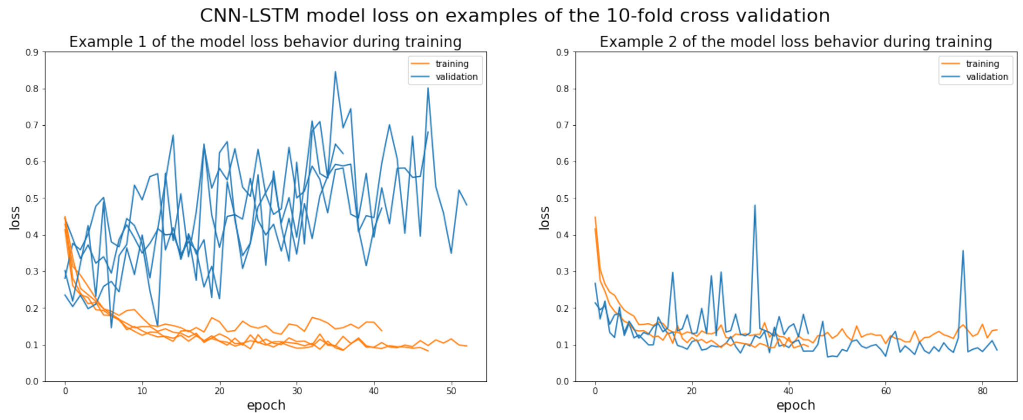


Figure 5.21: The Figure visualizes the loss behavior of the 10-fold cross-validation of the final CNN-LSTM model. The Figure shows two different behaviors of the loss-progression on the validation data during the training in the 10-fold cross-validation.

The normalized confusion matrices of Figure 5.22 show the type of mistake the different models are making. All models make relatively more mistakes in classifying when the true label is ending. The main observation of the CNN-LSTM model from the baseline model is the reduced number of false positives. The RNN-LSTM makes fewer mistakes in general, but main less false negative as shown in Figure 5.22.

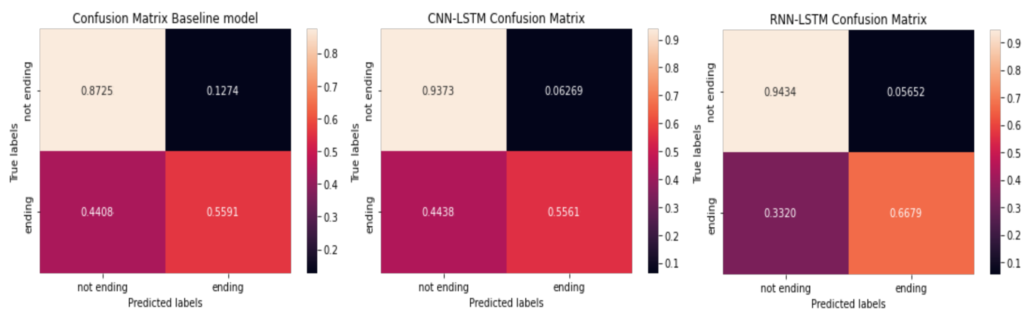


Figure 5.22: The comparison of the confusion matrix of the baseline model, final RNN-LSTM model and final CNN-LSTM model to compare the kind of misclassifications all model are making over the combined 10 folds of the cross validation.

The metrics discussed in Section 5.1 give a quantitative evaluation of the results seen in the confusion matrices of Figure 5.22. The goal is to get the score as close to one as possible. The first observation is that the RNN-LSTM score higher in every metric. The CNN-LSTM scores are higher than the baseline except in recall, where the score is approximately even.

The final part of the Results Section is the results from the resource analysis. The observation from the two bar charts in Figure 5.24 a straightforward: the RNN-LSTM model has a higher time it takes per prediction but needs a lower amount of working memory. The observation for the CNN-LSTM is vice versa (faster predictions, needs more working memory). The estimated prediction time and the working memory goal is to get as close to 0 as possible.

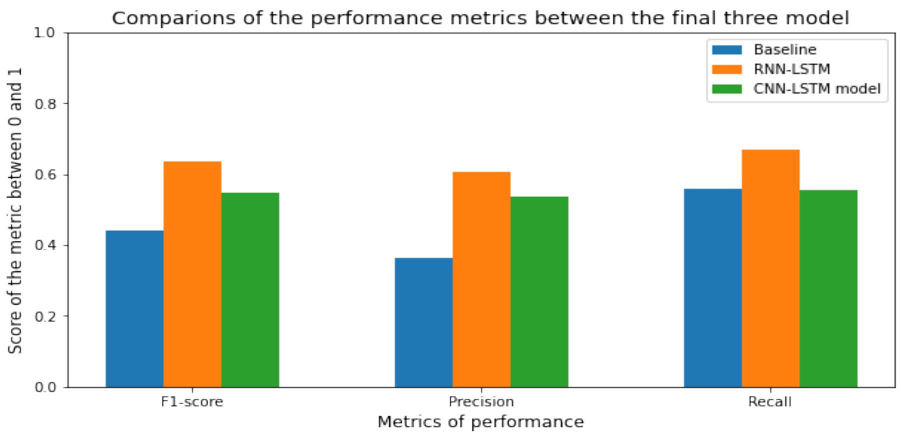


Figure 5.23: The comparison of the different aspects of the metrics of the baseline model, final RNN-LSTM model and final CNN-LSTM model to compare the performance of both model over the combined 10 folds of the cross validation.

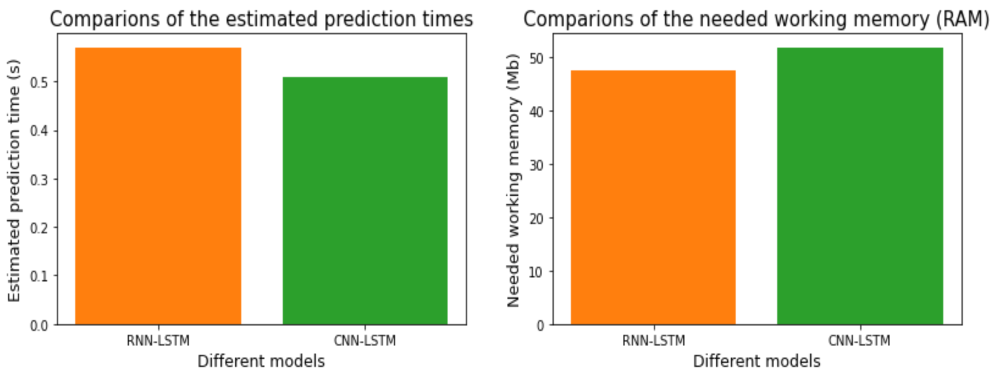


Figure 5.24: The comparison of the different aspects of the metrics of the baseline model, final RNN-LSTM model and final CNN-LSTM model to compare the performance of both model over the 10 folds of the cross validation.

5.5 Discussion

The main observation was that the RNN-LSTM model outperformed both the baseline and the CNN-LSTM models in every aspect. However, all the models are severely overfitting which was especially clear from the comparison of the metrics on the training and validation data of Figure 5.19 and Figure 5.18. A very clear sign of overfitting was also displayed in Figure 5.21 and Figure 5.20. These figures showed a clear inability for the models to learn the signal instead of the noise by the different shapes of the loss curves on the different folds compared to the training data. Figure 5.21 and Figure 5.20 also showed that the RNN-LSTM loss curves over the 10-fold cross-validation showed more signs of the model starting to learn the patterns and separate the signal from the noise by looking more like a traditional loss curve. The emphasis in the previous sentences is on ”starting”, the RNN-LSTM model still overfits significantly. The confusion matrix in Figure 5.22 and the bar chart in Figure 5.23 did show that the RNN-LSTM model classifies every category better and is the best model resulting from this project. Assuming the assumptions defined in Section 5.3.2 about the resources left on the HoloLens 2 hold, all models are lightweight enough to run online on the HoloLens 2.

Chapter 6

Conclusion

6.1 Conlusion and limitations

The main research question of this project is: Can we reliably classify the end of a high-level task using a RNN-LSTM or CNN-LSTM model based on the little data given by KIT-AR? The answer to the research question is that we cannot reliably classify the end of a high-level task using a RNN-LSTM or CNN-LSTM model and the KIT-AR data. This conclusion resulted from the evaluation results discussed and interpreted in Section 5.4 and Section 5.5. The main conclusion of these sections is that even though the final RNN-LSTM and CNN-LSTM models do improve over the baseline model, both models cannot reliably and accurately predict the end of a task. This conclusion was based on the large degree of overfitting, which led to decreased performance and a decreased robustness (reliability) of the models. As a result, the model performance is poor on unseen data, as proven in the results (Section 5.4). Therefore, the network fails to generalize the features or patterns present in the training dataset. Vidhya [8] mentions two main causes of overfitting on neural networks (NN) are:

- small training dataset, which causes the NN to learn too many details (including the noise) instead of the underlying patterns in the data.
- Overly complex models can be caused by a noisy dataset, many features in a dataset, or deeper NN based on the model architecture and different hyperparameters.

The cause of overfitting in this project is the combination of a small and a noisy training dataset relative to the nuanced classification task at hand. The NN complexity is not architecture or hyperparameter related.

All the models tried in this project are very shallow models. During the hyperparameter tuning, different hyperparameters attempt to reduce model complexity: max pooling size, number of neurons, kernel size, and filters. The hyperparameters are extensively tuned using k-fold cross-validation in the process of making the final RNN-LSTM and CNN-LSTM so that they won't cause an increased model complexity. Model complexity is also limited by the early stopper implemented during the training of the different models.

Possible solutions for the overfitting observed during the project besides collecting more data are data augmentation, different regularisation techniques, and feature selection to reduce model complexity. However, these solutions are also where the limitation of this project start. The project has a limited time span, which means the time needed to be used efficiently, and it means that not all possible solutions to answer the research question can be tried. Instead of using the LOUO splitting strategy, we used the standard randomized training, validation, and test split. So this mistake cost a lot of time and reduced the number of different solutions tried during this project.

Besides not being able to try more different techniques to reduce limitations and increase performance, there is also a limitation to the hyperparameter techniques used during the project. Starting at a sub-grid of hyperparameters and moving this grid based on the results over the initial sub-grid could cause finding a local maximum instead of a global maximum. A global optimum can be missed due to the effect of different hyperparameters on each other changing with the hyperparameters. For example: Out of the starting sub-grid of the first round of hyperparameter tuning, a larger batch size seems to perform better, and a smaller step size also seems to do better on average. However, the model could perform exponentially worse on the second sub-grid because of the combination of a larger batch size and a smaller step size since these effects don't have to be linear. However, if the initial grid is large enough, the consequences of these effects on the different parameters show in the results of the first round of hyperparameter tuning. The main conclusion is that there is a balance between 1) the initial grid size and the density of the grid and 2) the time it takes to run the grid search because of the large number of hyperparameters. There are also limitations to the resource analysis needed without trying the model on the actual HoloLens. The prediction time resulting from the emulation is an estimation where the accuracy is very closely related to the conversion formula discussed in Section 5.3. Even though the logic is sound, there is a possibility that the relation between the single thread rating and the prediction time is not completely linear. The most accurate way of estimating the CPU needed for the prediction time to be low enough is by using the model on the HoloLens.

6.2 Future works

So to improve the model's performance and reduce the overfitting without just collecting more data, future work should be focused on more data augmentation, different regularisation techniques, or try to make a model for each task. Random over-and undersampling was used as one data augmentation technique during the project. However, there are much more sophisticated over-and undersampling techniques like OHIT discussed in the paper written by Zhu et al. [21] about minority oversampling for imbalanced time series classification.

A second possible solution is weight regularisation which works through penalizing or adding a constraint to the loss function [8]. Otherwise, the loss will be very low during overfitting, and the weight changes will be minimal. Regularization terms are constraints applied to the optimization algorithm when minimizing loss function apart from reducing the error between the predicted value and actual value. Examples of the optimization algorithms mentioned in the sentence before are Stochastic Gradient Descent or Adam, a replacement optimization algorithm for stochastic gradient descent, used in the final models of this project. According to the paper by Carpenter et al. [4] possible regularisation techniques for small datasets to try in future work are L1 regularization (or Lasso), L2 regularization (or Ridge), and L1+L2 regularization (Elastic Net). There is a possibility that not every task has the same pattern before the end of a task, so a possible solution could be making a separate model for each task. However, this would further reduce the amount of training data for the models.

The final potential topic that could be interesting for future work is to apply the segmentation techniques and data transformation approaches discussed in the paper written by Zheng et al. [20] to the KIT-AR data to see if these techniques increase the overall performance of the project's final models. These segmentation techniques could be used to reduce the effect of the noise in the data and make the signal clearer, thereby increasing the performance of the models over the raw data. The best data segmentation and transformation technique tried during Zheng's research was the multichannel transformation method (MCT) which improved the model performance on all datasets, including the different versions of the WISDM dataset [17].

In future works, different more intelligent hyperparameter tuning methods should be explored due to the high dimensionality of the hyperparameters. This high dimensionality of the hyperparameters causes an exponential increase in the time needed for the hyperparameter tuning. Future hyperparameter tuning methods should include the different gradient descent methods, Bayesian optimization, random grid search, and halving grid search to decrease the time needed for the hyperparameter tuning. Besides the hyperparameter tuning, would it also be interesting to see the effect of changing the method of the training-validation-test split to where the test and validation data is not data from unseen subjects. This change in training-validation-test split would most likely increase the performance but change the use-case of the resulting models.

Bibliography

[1] Preeti Agarwal and Mansaf Alam. A lightweight deep learning model for human activity recognition on edge devices. *Procedia Computer Science*, 167:2364–2373, 01 2020.

[2] Jake K Aggarwal and Michael S Ryoo. Human activity analysis: A review. *ACM Computing Surveys (CSUR)*, 43(3):1–43, 2011.

[3] Ankita, Shalli Rani, Himanshi Babbar, Sonya Coleman, Aman Singh, and Hani Moaiteq Aljahdali. An efficient and lightweight deep learning model for human activity recognition using smartphones. *Sensors*, 21(11), 2021.

[4] Marcus Carpenter, Chunbo Luo, and Xiao-Si Wang. The effects of regularisation on rnn models for time series forecasting: Covid-19 as an example. *arXiv preprint arXiv:2105.05932*, 2021.

[5] Yuwen Chen, Kunhua Zhong, Ju Zhang, Qilong Sun, Xueliang Zhao, et al. Lstm networks for mobile human activity recognition. In *Proceedings of the 2016 International Conference on Artificial Intelligence: Technologies and Applications, Bangkok, Thailand*, pages 24–25, 2016.

[6] Yixin Gao, S Swaroop Vedula, Carol E Reiley, Narges Ahmidi, Balakrishnan Varadarajan, Henry C Lin, Lingling Tao, Luca Zappella, Benjamın Béjar, David D Yuh, et al. Jhu-isi gesture and skill assessment working set (jigsaws): A surgical activity dataset for human motion modeling. In *MICCAI workshop: M2cai*, volume 3, page 3, 2014.

[7] Adel Ghazikhani, Hadi Sadoghi Yazdi, and Reza Monsefi. Class imbalance handling using wrapper-based random oversampling. In *20th Iranian Conference on Electrical Engineering (ICEE2012)*, pages 611–616, 2012.

[8] Chirag Goyal. Guide to prevent overfitting in neural networks. *Analytics Vidhya*, Jun 2021.

[9] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A field guide to dynamical recurrent neural networks*, 2001.

[10] Sönke Knoch, Nico Herbig, Shreeraman Ponpathirkoottam, Felix Kosmalla, Philipp Staudt, Daniel Porta, Peter Fettke, and Peter Loos. Sensor-based human–process interaction in discrete manufacturing. *Journal on Data Semantics*, 9:1–17, 03 2020.

[11] Felix Mannhardt, Riccardo Bovo, Manuel Fradinho Oliveira, and Simon Julier. A taxonomy for combining activity recognition and process discovery in industrial environments. In David Camacho, Paulo Novais, Antonio J. Tallón-Ballesteros, and Hujun Yin, editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2018 - 19th International Conference, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 84–93, Germany, 2018. Springer. 19th International Conference on Intelligent Data Engineering and Automated Learning, IDEAL 2018 ; Conference date: 21-11-2018 Through 23-11-2018.

[12] Saeed Mohsen, Ahmed Elkaseer, and Steffen G Scholz. Industry 4.0-oriented deep learning models for human activity recognition. *IEEE Access*, 9:150508–150521, 2021.

[13] Abdulmajid Murad and Jae-Young Pyun. Deep recurrent neural networks for human activity recognition. *Sensors*, 17:2556, 11 2017.

[14] Comparison between intel core i7-8750h vs snapdragon 850, Dec 2021.

[15] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing Management*, 45:427–437, 07 2009.

[16] Md Zia Uddin and Ahmet Soylu. Human activity recognition using wearable sensors, discriminant analysis, and long short-term memory-based neural structured learning. 2021.

[17] Gary M Weiss, Kenichi Yoneda, and Thaier Hayajneh. Smartphone and smartwatch-based biometrics using activities of daily living. *IEEE Access*, 7:133190–133202, 2019.

[18] Wikepedia. Precision and recall, Oct 2021.

[19] Cheng-Hong Yang and Po-Yin Chang. Forecasting the demand for container throughput using a mixed-precision neural architecture based on cnn–lstm. *Mathematics*, 8(10), 2020.

[20] Xiaochen Zheng, Meiqing Wang, and Joaquín Ordieres-Meré. Comparison of data preprocessing approaches for applying deep learning to human activity recognition in the context of industry 4.0. *Sensors*, 18(7), 2018.

[21] Tuanfei Zhu, Cheng Luo, Jing Li, Siqi Ren, and Zhihong Zhang. Minority oversampling for imbalanced time series classification. *arXiv preprint arXiv:2004.06373*, 2020.

List of Figures

List of Tables