UiA Fakultet for teknologi og realfag

IKT450-G - Deep Neural Networks

Classification of human activities

Autumn 2023

Haas Fabian, Reichl Markus, Weingartshofer Florian

December 3, 2023

Mandatory Group Declaration

Each student is solely responsible for familiarizing themselves with the legal aids, guidelines for their use, and rules regarding source usage. The declaration aims to raise awareness among students of their responsibilities and the consequences of cheating. Lack of declaration does not exempt students from their responsibilities.

1.	We hereby declare that our submission is our own work and that we have not used other sources or received any help other than what is mentioned in the submission.	Yes
2.	**We further declare that this submission:** • Has not been used for any other examination at another department/university/-college domestically or abroad. • Does not reference others' work without it being indicated. • Does not reference our own previous work without it being indicated. • Has all references included in the bibliography. • Is not a copy, duplicate, or transcription of others' work or submission.	Yes
3.	We are aware that violations of the above are considered to be cheating and can result in cancellation of the examination and exclusion from universities and colleges in Norway, according to the Universities and Colleges Act, sections 4-7 and 4-8 and the Examination Regulation, sections 31.	Yes
4.	We are aware that all submitted assignments may be subjected to plagiarism checks.	Yes
5.	We are aware that the University of Agder will handle all cases where there is suspicion of cheating according to the university's guidelines for handling cheating cases.	Yes
6.	We have familiarized ourselves with the rules and guidelines for using sources and references on the library's website.	Yes
7.	We have in the majority agreed that the effort within the group is notably different and therefore wish to be evaluated individually. Ordinarily, all participants in the project are evaluated collectively.	No

Publishing Agreement

Authorization for Electronic Publication of Work The author(s) hold the copyright to the work. This means, among other things, the exclusive right to make the work available to the public (Copyright Act. §2).

Theses that are exempt from public access or confidential will not be published.

We hereby grant the University of Agder a royalty-free right to make the work available for electronic publication:	Yes
Is the work confidential?	No
Is the work exempt from public access?	No

Disclaimer

In this project, we have used the language model GPT-4 [1] to generate ideas and arguments for our project, and used it to help us debug our code. We have not used GPT-4 to write entire paragraphs or chapters for our report.

Contents

1 Abstract **1**

2 Introduction **1**

3 Related Work **1**

4 Methodology **2**

4.1 Dataset 2

4.2 Data Exploration 3

4.3 Preprocessing 5

4.4 Architectures 5

4.4.1 Feedforward Neural Network 6

4.4.2 Convolutional Neural Network 6

4.4.3 Long Short-Term Memory 7

4.4.4 Transformer 7

4.5 Training 8

4.6 Evaluation 8

5 Result **8**

5.1 Feedforward Neural Network 9

5.1.1 Simple Feedforward Neural Network 9

5.1.2 Tuned Complex Feedforward Neural Network 9

5.2 Convolutional Neural Network 12

5.2.1 Simple Convolutional Neural Network 12

5.2.2 Tuned Convolutional Neural Network 12

5.3 Long Short-Term Memory 14

5.4 Transformer 15

6 Conclusion **17**

1 Abstract

This report details our study on classifying human activities using accelerometer data, specifically utilizing the HAR dataset, where smartphones, placed at the waist, recorded data during six distinct activities. We aimed to develop basic neural networks to effectively classify these activities, with a focus on achieving good performance with models of minimal complexity. We discovered that minimalistic neural network designs could still yield impressive results, and the hyperparameter optimization techniques employed significantly enhanced these outcomes without substantial increases in complexity.

2 Introduction

Classifying human activities is a task performed daily by various devices such as smartwatches and smartphones. These devices offer a lot of sensor data that can be used in machine learning contexts. With this project, we focus on using accelerometer data to classify human activities. We selected this project because it aligns well with the degree we are currently pursuing at our home university called *Mobile Computing* and helped us increase our knowledge of machine learning in applications for mobile devices.

The goal we set ourselves for this project was to develop basic neural networks for classifying human activities. We wanted to explore with how little complexity we could achieve a good result and how much impact optimization techniques had on the result. We also wanted to experiment with more complex neural networks and see how much better they performed compared to the simpler ones, to see if the complexity is needed. We did not want to focus on developing a model that could be used for real-time classification, but rather focus on the classification itself.

The following sections detail our approach to the project such as related work, the methodologies we employed, the results we obtained, and the conclusions drawn from our findings.

3 Related Work

Human Activity Recognition (HAR) is an expanding area of research within mobile computing, which is attributed mainly to the rising availability of sensors embedded in various devices such as smartphones and wearables [2], [3]. This expansion led to growing research on this topic, providing us with many papers to use as reference. Research on this topic shows, that human activity can be recognized by applying relatively simple machine learning methods like K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) with considerable success [4], [5]. However, such machine learning methods for HAR require the selection of features by human intervention and might not even achieve sufficient performance [2]. Recent years witnessed a growing shift in HAR research towards more powerful deep learning models as a remedy for these challenges [6].

Convolutional Neural Network (CNN) are renowned for their excellent capability to han-

dle high-dimensional data and robust feature extraction ability, and have become popular for HAR tasks. They possess the intrinsic ability to conduct automatic feature extraction and selection from raw data, which greatly benefits the handling of complex sensor data. Their robustness to local translations and distortions has made them a good fit for recognizing activities from sensor data with spatial hierarchies or where the spatial positions of patterns play a crucial role. [5], [7]

Newer research documented great success with more complex and sequence-sensitive models like the Long Short-Term Memory (LSTM). LSTM are a type of Recurrent Neural Network (RNN) that are naturally suited to handle time-series data, making them ideal for HAR, given the temporal dependencies involved. Especially the works of Zhao, Yang, Chevalier, *et al.* have shown promise in achieving high recognition results with LSTM for HAR tasks [5], [6].

Deeper architectures, like Deep Convolutional Neural Network (DCNN), have also found relevance in HAR, demonstrating their capability to handle complex recognition tasks. creagh2021dcnn's use of a DCNN for distinguishing healthy individuals from participants with MS showcases an innovative use of deep learning in health-related HAR applications. This highlighted the potential of DCNN to deliver superior results over traditional SVM methods in a challenging HAR context. [8]

Another line of research has been the use of hybrid deep models for HAR. One such instance is the EMI-RNN model, which merges the strengths of CNN and LSTM. This combination model expands the potential of deep learning in HAR by utilizing the depth of CNN for feature extraction and the sequential understanding of LSTM for temporal interpretation. [9]

Furthermore, several studies have sought to examine the comparative strengths and usage of different deep learning models for HAR [10]–[12]. These studies serve as practical guides for understanding the application of these models in specific HAR contexts and would eventually become a basis for our own work. As our work focused on our personal understanding of the topic at hand, we aimed to maintain a focus on simplicity in our own model design.

4 Methodology

4.1 Dataset

In our project, we utilized the *Human Activity Recognition Using Smartphones* dataset, referenced in our literature as [13]. This database is built from recordings of 30 subjects performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors.

It was sourced from a diverse group of 30 volunteers aged 19 to 48; the dataset captures six common activities: walking, walking upstairs, walking downstairs, sitting, standing, and laying. Each subject wore a Samsung Galaxy S II smartphone on the waist, enabling the capture of 3-axial linear acceleration and 3-axial angular velocity at a 50Hz rate. The dataset, a multivariate time-series collection in the computer science domain, is suitable

for tasks such as classification and clustering.

Instead of the raw time-series data, we utilized the processed features of the *Human Activity Recognition Using Smartphones* dataset. The dataset provided a comprehensive set of features derived from the accelerometer and gyroscope sensors, which were more aligned with our project objectives.

These features include various statistical measures estimated from the processed sensor signals such as but not limited to mean, standard deviation, median absolute deviation as well as frequency domain features derived from the Fast Fourier Transform (FFT). In total, the dataset provided 561 features for each sample. Which of these features were used in our project is discussed in Section 4.3.

The authors of the dataset also provided a partitioned dataset, which was split into two sets: a training set and a test set. The training set consists of 70% of the data, while the test set consists of the remaining 30%. Which comes to 7352 and 2947 samples respectively.

The authors also provide a short YouTube video [14] that shows the data collection process. This video shows that the data was collected by having the subjects perform the activities while wearing the smartphone on their waist. The subjects were also asked to perform the activities multiple times, which is why the dataset contains multiple samples for each subject and activity.

4.2 Data Exploration

In machine learning, it is important to understand the data you are working with, since this can help you decide which features to use and which machine learning approach to use. Therefore, we analyzed the dataset with plots to get a better understanding of the data we were working with.

Since looking at all the features was not feasible, we decided to only look at features that had less than 10% correlation with other features. Doing this ensures that we look at features that had the least amount of redundancy and should therefore be the most informative.

Figure 1 shows a pair-plot of the features we decided to look at. A pair-plot shows a scatterplot for each feature combination, as well as histograms of the features along the diagonal. From the pair-plot, we can see that there are some features that could be used to distinguish between a few classes. The features `tBodyAcc-std()-X` and `tBodyAcc-correlation()-X,Z`, for example, show a good separation between the different `WALKING` activities and the more stationary activities.

This is further confirmed by looking at the boxplot in Figure 2, which shows the distribution of the features for each class, and the feature `tBodyAcc-std()-X` also shows a good separation between the stationary activities and the `WALKING` activities. The boxplot further shows that some features have little variance, such as `tBodyBodyGyroJerkMag-maxInds`, which has a small interquartile range and a lot of outliers. The feature `tBodyAcc-correlation()-X,Z` on the other hand shows a strong overlap between the classes, which is not surprising since it is a correlation between two

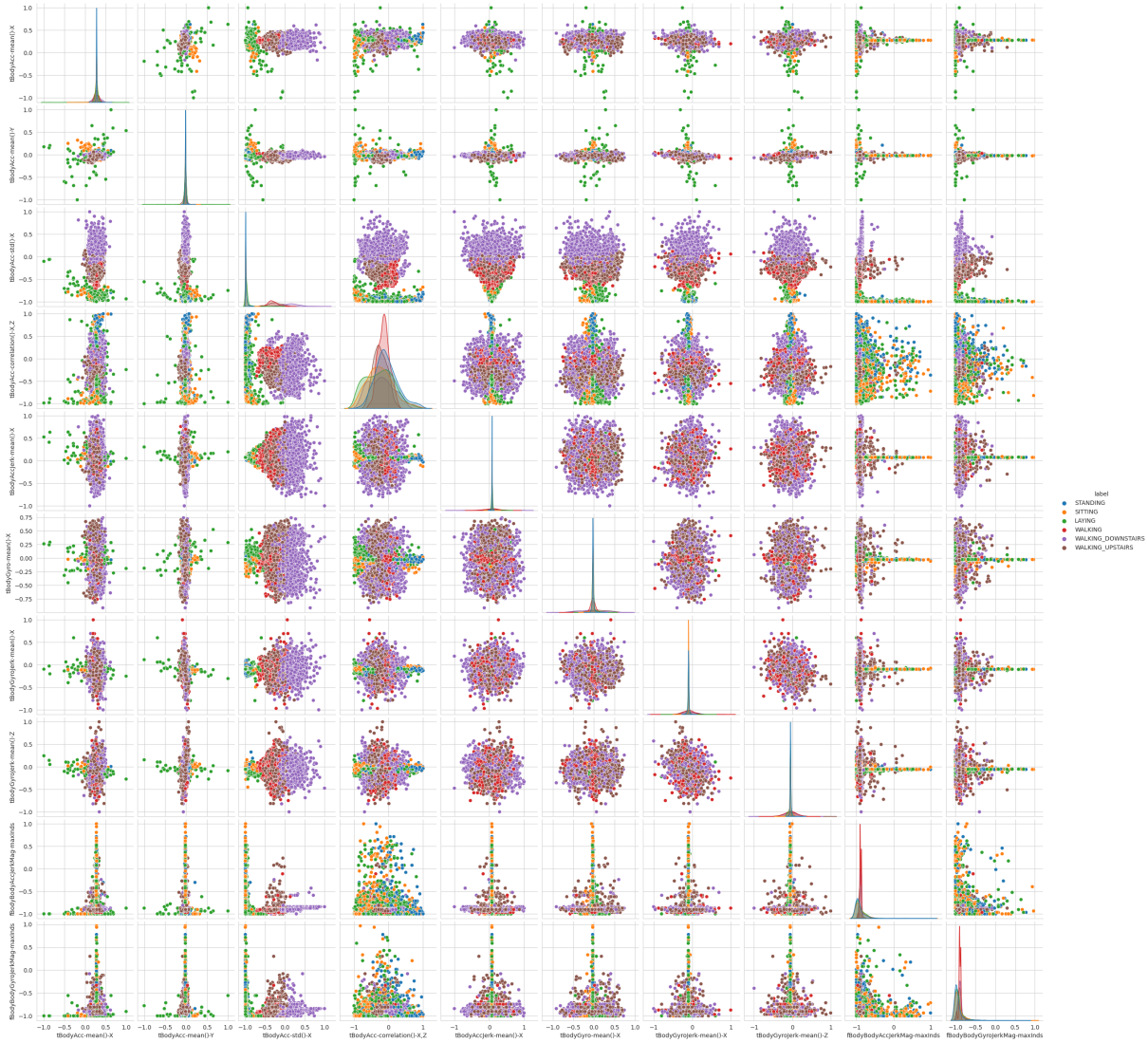

Figure 1: Pairplot of highly uncorrelated features

features that are already used to distinguish between the classes.

The plot that showed the most promise was the line-plot in Figure 3, which shows the total acceleration over time in each axis. This graph shows that the accelerations can definitely be used to distinguish between the different activities, since they have very different patterns with little overlap. The activity `LAYING` can be distinguished from the other activities, as it has no overlap with the other activities. The activities `SITTING` also shows no overlap with the other activities in the Z-axis and Y-axis, apart from itself. But the graph even allows us to clearly identify `STANDING` which has a lot of overlaps in the X and Y-axis, but can still be distinguished from the other activities in the Z-axis.

The data exploration showed that the data is definitely separable and can probably also be solved with less complex machine learning approaches. However, since the focus of the project was on deep learning, we did not explore other machine learning approaches such as knn, decision trees or support vector machines.

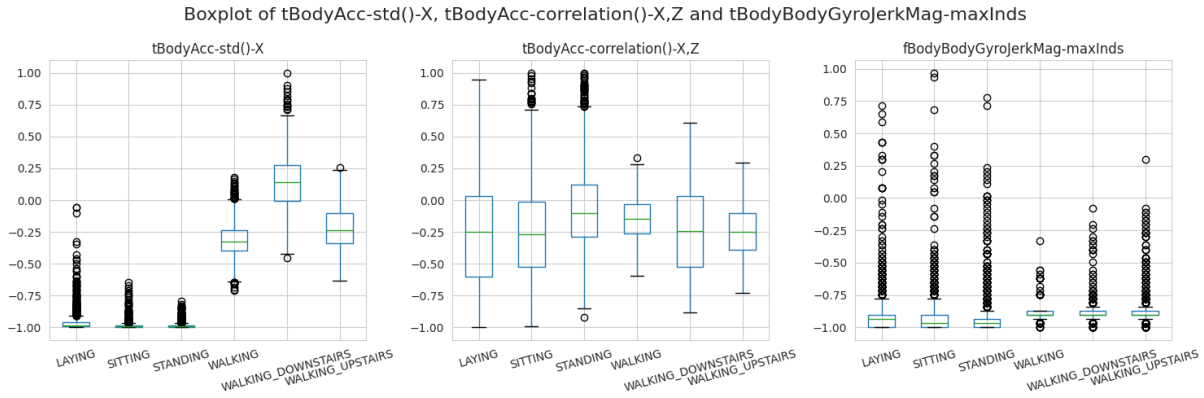

Figure 2: Boxplot of tBodyAcc-std()-X, tBodyAcc-correlation()-X,Z and tBodyBodyGyroJerkMag-maxInds

4.3 Preprocessing

Since the dataset was already processed, we did not have to perform any preprocessing steps to generate features, but we did some preprocessing to reduce the number of features.

For this, we used a correlation matrix to find features that were highly correlated with each other. Features that have a high correlation with each other imply that they are redundant and can be removed. Redundant features can be removed since the information they provide is already captured by other features. This reduces the number of features, which in turn leads to models that are less complex, as they need to learn fewer features.

Figure 4 shows the correlation matrix before preprocessing, while Figure 5 shows the correlation matrix after preprocessing. As a threshold for removing features, we used a correlation coefficient of *0.95*. This reduced the number of features from 561 to 253, effectively reducing the number of features by more than half.

We also decided to use the test set provided by the dataset as our held-back test set. For training, we split the training set into a set used for actual training and a set used for validation. This was done to be able to evaluate the performance of the model during training, and to ensure that the model did not over fit to the training set. The validation set was 30% of the training set, which comes to around 2206 samples for the validation set and 5146 samples for the training set.

4.4 Architectures

This section discusses the architectures we tried and evaluated in our project. For different architectures, we tried to first build a simple model, and then a more complex one that where we also tuned the hyperparameters.

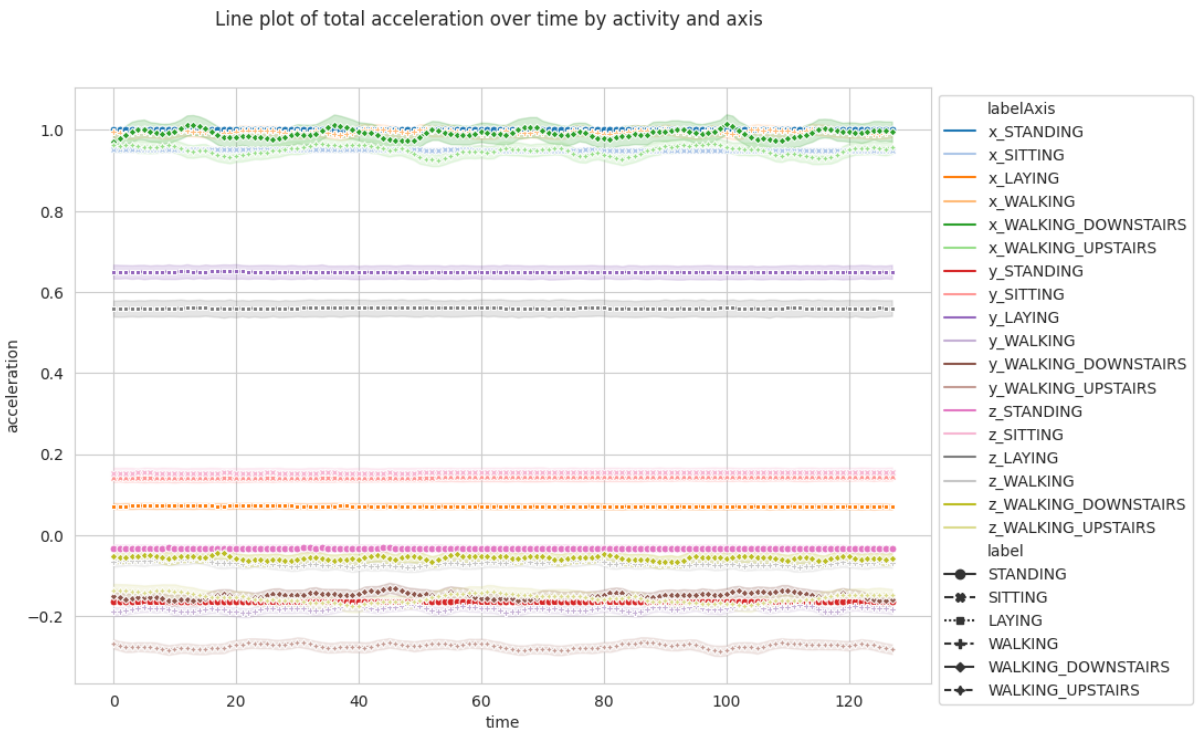

Figure 3: Lineplot of total acceleration over time in each axis

4.4.1 Feedforward Neural Network

Simple Feedforward Neural Network

Our project initiated with a simple Feedforward Neural Network (FF). This model consisted of an initial dense layer with 512 units, a hidden dense layer with 128 units using the ReLU activation function, and an output dense layer with 6 neurons. This was the simplest model we could think of. Figure 6 shows a visualization of the simple Feedforward Neural Network.

Tuned Complex Feedforward Neural Network

The results from the simple Feedforward model were already pretty good, but we wanted to see if we could improve the performance by increasing the complexity of the model and tuning the hyperparameters. For this, we utilized Keras Tuner, employing a RandomSearch strategy. This approach allowed us to automatically fine-tune the hyperparameters of our model. Figure 7 shows the network and what parameters are tunable, such as the number of layers, the number of units in each layer and the dropout rate. The goal was to identify the most effective configuration for our dataset, enhancing the model's performance.

4.4.2 Convolutional Neural Network

Simple Convolutional Neural Network

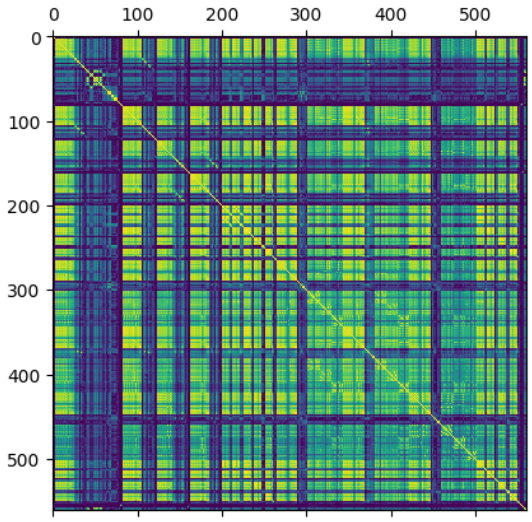

Figure 4: Correlation matrix before preprocessing

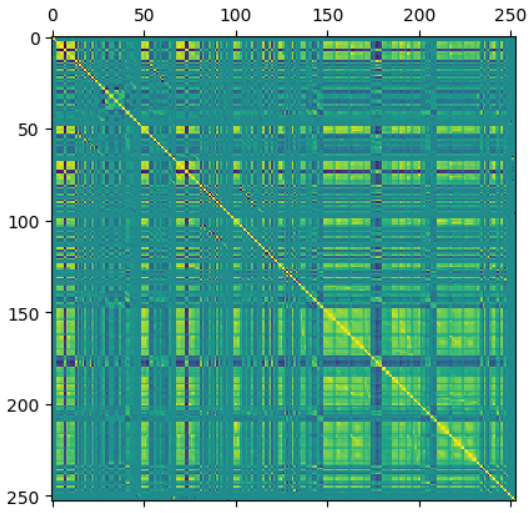

Figure 5: Correlation matrix after preprocessing

Figure 6: Visualization of simple Feedforward Neural Network architecture

We also developed a Convolutional Neural Network model, comprising convolutional layers, max pooling, global max pooling, a dropout layer for regularization, and a softmax output layer. While such models are typically used for image classification, we wanted to explore whether they could be used for time-series classification as well. This also did not need any real preprocessing apart from reshaping the data to fit the input shape of the network. Figure 8 shows a visualization of the simple Convolutional Neural Network.

Tuned Convolutional Neural Network While the simple Convolutional Neural Network model performed well, we still wanted to explore whether we could improve its performance by tuning its hyperparameters. This involved tuning the number of filters in the convolutional layers and the dropout rate, as shown in Figure **??**.

4.4.3 Long Short-Term Memory

We wanted to also try classification by using the windowed time series values instead of the processed features. We decided to try a Long Short-Term Memory (LSTM) model, which is a type of Recurrent Neural Network (RNN) that is commonly used for time series classification. We did not tune the hyperparameters of this model, since we wanted to see how well it would perform without tuning.

4.4.4 Transformer

Since transformers are currently very popular in the field of natural language processing, we wanted to see if they could also be used for time series classification. Again, we did not tune the hyperparameters of this model to see how well it would perform without

Figure 7: Visualization of tunable Feedforward Neural Network architecture

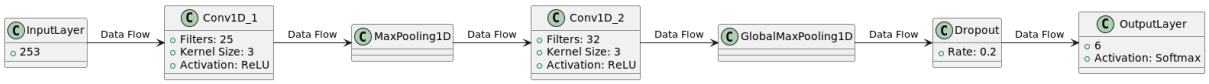

Figure 8: Visualization of simple Convolutional Neural Network architecture

tuning.

4.5 Training

As mentioned in Section 4.3, we split the training set into a training set and a validation set. The validation set was used to evaluate the performance of the model during training, and to ensure that the model did not over fit to the training set.

We trained every model for 100 epochs and used five trials for the Keras Tuner with 3 models per trial, so we tuned a total of 15 models for each model we tuned. For the models, we tuned, we used `RandomSearch` as the tuner and optimized for validation accuracy.

4.6 Evaluation

For models that were tuned, we retrained the model with the best score on the training set and evaluated it on the test set. This was done to plot the accuracy and loss graphs for the tuned models.

For evaluating the performance of the models, we used the test set provided by the dataset. This is also commonly referred to as the held-back test set, since it is not used during training or validation. Having a held-back test set is important since it allows us to evaluate the performance of the model on unseen data.

For evaluating the models, we used the accuracy, precision, recall and F1-score, as well as the confusion matrix. These metrics are commonly used for classification tasks, and they allow us to evaluate the performance of the model on a per-class basis.

5 Result

This section presents the results of the models described in Section 4. The results are presented in the form of graphs and tables. The graphs are plotted using the `seaborn` [15] and `matplotlib` [16] library. Each evaluation includes the following graphs and tables:

- **Accuracy and loss graph**: The accuracy and loss graph shows the accuracy and loss of the model during training and validation using the `accuracy` and `loss` metrics provided by Keras.

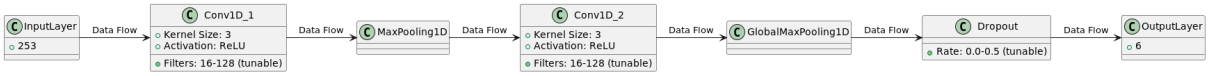

Figure 9: Visualization of tunend Convolutional Neural Network architecture

- **Performance metrics**: The performance metrics table shows the performance metrics of the model on the held back test set calculated using the `classification_report` function from `sklearn` [17]. Its metrics include precision, recall, f1-score, and support, as well as the overall accuracy of the model.
- **Confusion matrix**: The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives calculated using the `confusion_matrix` function from `sklearn` [17]. The labels 0–5 correspond to the classes `WALKING`, `WALKING_UPSTAIRS`, `WALKING_DOWNSTAIRS`, `SITTING`, `STANDING`, and `LAYING`, respectively.

5.1 Feedforward Neural Network

5.1.1 Simple Feedforward Neural Network

Figure 10 shows the accuracy and loss graph for the simple feedforward neural network. The graph shows that the training accuracy started at around 0.85 and increased to around 0.98 after 100 epochs, which is a steady increase matching the training loss. The validation accuracy started at around 0.95 and ended at about the same value after fluctuating between 0.9 and 0.975, with one spike at around 0.825. That the validation accuracy started so high indicates that the problem is not very difficult, however, the fluctuation indicates that the model is not very stable and potentially overfitting.

This is further supported by the performance metrics in Figure 11 which evaluate the model on the held back test set. The metrics indicate that the model is good at identifying the classes `WALKING`, `WALKING_UPSTAIRS`, `WALKING_DOWNSTAIRS`, and `LAYING`, but struggles at identifying `SITTING` and `STANDING`. The overall accuracy of the model is 0.89, which is pretty good for a simple model that was not tuned.

Looking at the confusion matrix in Figure 12, we can see that it often confuses `STANDING` with `SITTING`. Which explains why performance metrics for these two classes are lower than the others.

5.1.2 Tuned Complex Feedforward Neural Network

After tuning the hyperparameters of the feedforward neural network, we end up with the architecture shown in Figure 13. Which consists of 2 dense layers with 384 and 640 neurons, respectively, and a dropout layer with a dropout rate of 0.1, followed by 6 dense layers with 32 neurons each. Making it more complex than the simple feedforward neural network.

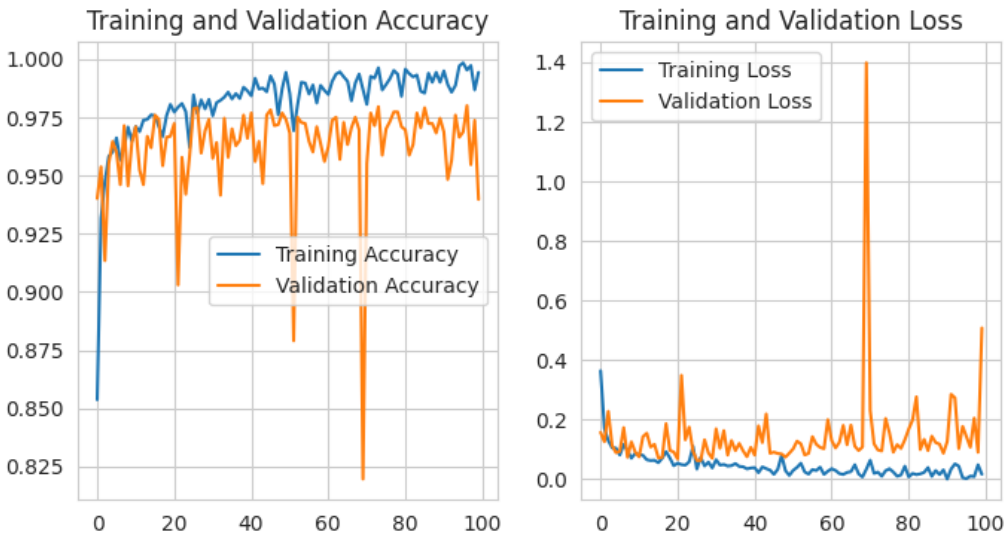

Figure 10: Accuracy and loss graph for the simple feedforward neural network.

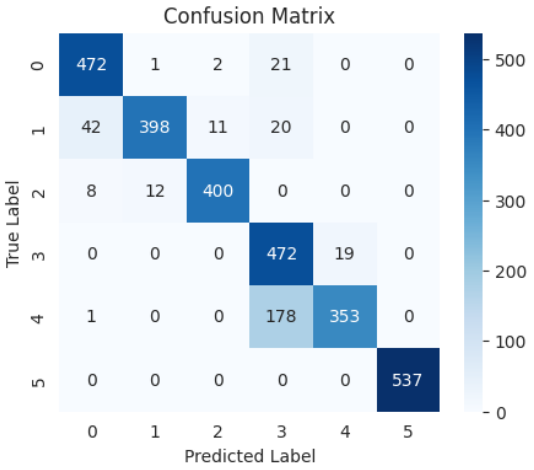

Figure 11: Performance metrics for the simple feedforward neural network.

Figure 12: Confusion matrix for the simple feedforward neural network.

This increased complexity seems to allow the model to learn more about the data, as shown in Figure 14. The training accuracy starts at over 0.99 and hits 1.0 several times during training, while the validation accuracy starts and ends at around 0.98. This indicates that the model is probably overfitting, but the validation accuracy is still higher than the training accuracy of the simple feedforward neural network, which indicates that the model is more stable. While the fluctuation might seem more intense than in the simple feedforward neural network, it is important to note that the y-axis is scaled differently, and the fluctuation is actually smaller.

The performance metrics in Figure 15 also show that the model got a lot better at identifying `SITTING` and `STANDING` compared to the simple feedforward neural network. The overall accuracy of the model is 0.94, which is a good improvement over the simple feedforward neural network.

Looking at the confusion matrix in Figure 16, we can see that it still sometimes confuses

`STANDING` and `SITTING`, but not as often as the simple feedforward neural network.

Overall, the model seems to be doing a good job at identifying the classes, and it also seems to generalize better than the simple feedforward neural network.

Figure 13: Visualization of tuned feedforward neural network architecture

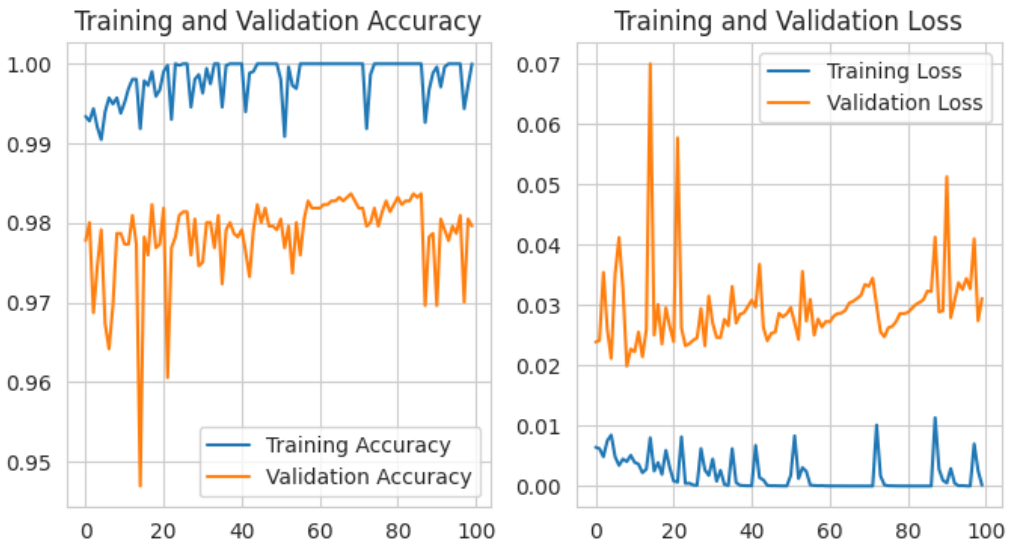

Figure 14: Accuracy and loss graph for the tuned feedforward neural network.

Figure 15: Performance metrics for the tuned feedforward neural network.

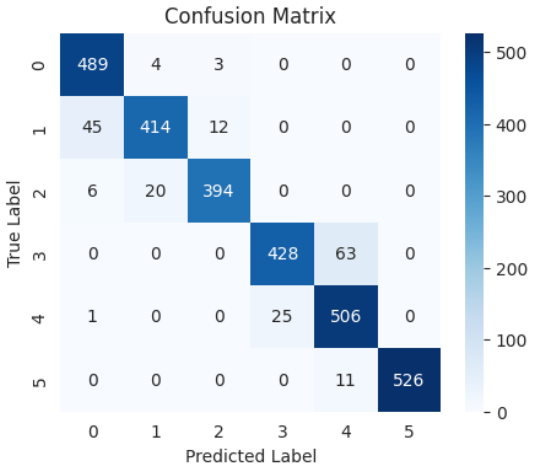

Figure 16: Confusion matrix for the tuned feedforward neural network.

5.2 Convolutional Neural Network

5.2.1 Simple Convolutional Neural Network

Figure 17 shows the accuracy and loss graph for the simple convolutional neural network. It has a steady increase in training and validation accuracy, and a steady decrease in

training and validation loss. It also seems to be pretty stable as it does not fluctuate much, however this might be because the y-axis is scaled differently, as it starts at 0.4. The starting accuracies are also more in line with what you would normally expect from a model that has not been fitted to the data. What is really curious is that the validation accuracy is consistently higher than the training accuracy, which is not something you would normally expect and might indicate that the model is underfitting.

Looking at the performance metrics in Figure 18, the model seems to be performing slightly worse than the simple feedforward neural network, with an overall accuracy of 0.88. Interestingly it seems to struggle with categorizing `WALKING` and `WALKING_UPSTAIRS`.

The confusion matrix in Figure 19 that the mistakes it makes are more distributed than the simple feedforward neural network. This could mean that the error the model makes has less bias than the simple feedforward neural network.

Overall, the model still performs decently, but it does not seem to be an improvement over the simple feedforward neural network.

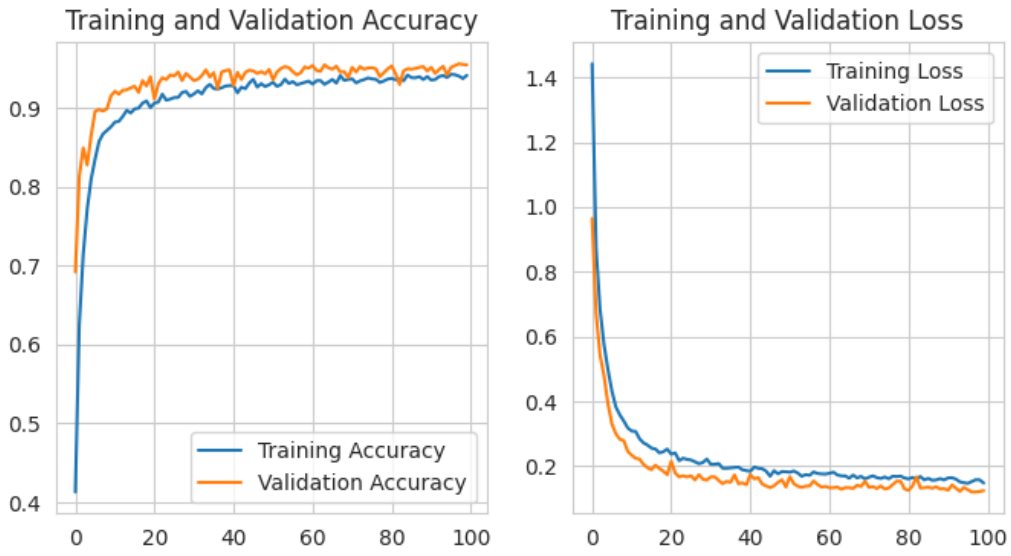

Figure 17: Accuracy and loss graph for the simple convolutional neural network.

5.2.2 Tuned Convolutional Neural Network

After tuning the hyperparameters of the convolutional neural network, we end up with the architecture shown in Figure 20. In contrast to the untuned convolutional neural network, the first convolutional layer has 112 filters instead of 25, the second convolutional layer has 64 filters instead of 32 and the dropout rate is 0.0 instead of 0.2.

As Figure 21 shows, this small change allows for the training accuracy to reach 1.0, while the validation accuracy fluctuates mostly between 0.98 and 0.97. This indicates that the model is overfitting the training data.

The performance metrics in Figure 22 show that the model is performing better than the simple convolutional neural network, with an overall accuracy of 0.94. And having pretty good scores on the metrics for all the classes.

	precision	recall	f1-score	support
WALKING	0.79	0.95	0.86	496
WALKING_UPSTAIRS	0.82	0.77	0.80	471
WALKING_DOWNSTAIRS	0.97	0.80	0.88	420
SITTING	0.88	0.83	0.86	491
STANDING	0.86	0.90	0.88	532
LAYING	0.99	0.99	0.99	537
accuracy			0.88	2947
macro avg	0.88	0.87	0.88	2947
weighted avg	0.88	0.88	0.88	2947

Figure 18: Performance metrics for the simple convolutional neural network.

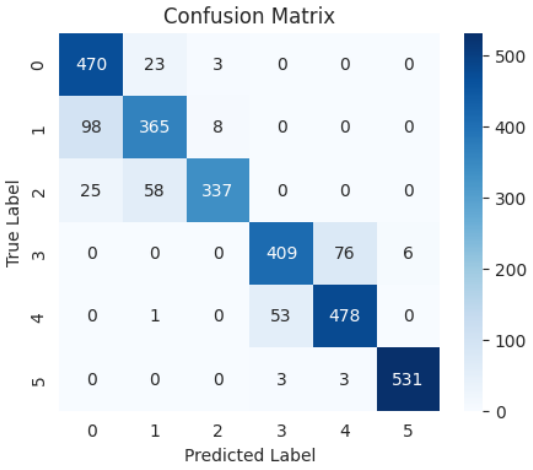

Figure 19: Confusion matrix for the simple convolutional neural network.

The confusion matrix in Figure 23 shows that the model is still making some mistakes, but it is doing a pretty good job at classifying the activities.

Figure 20: Visualization of tuned convolutional neural network architecture

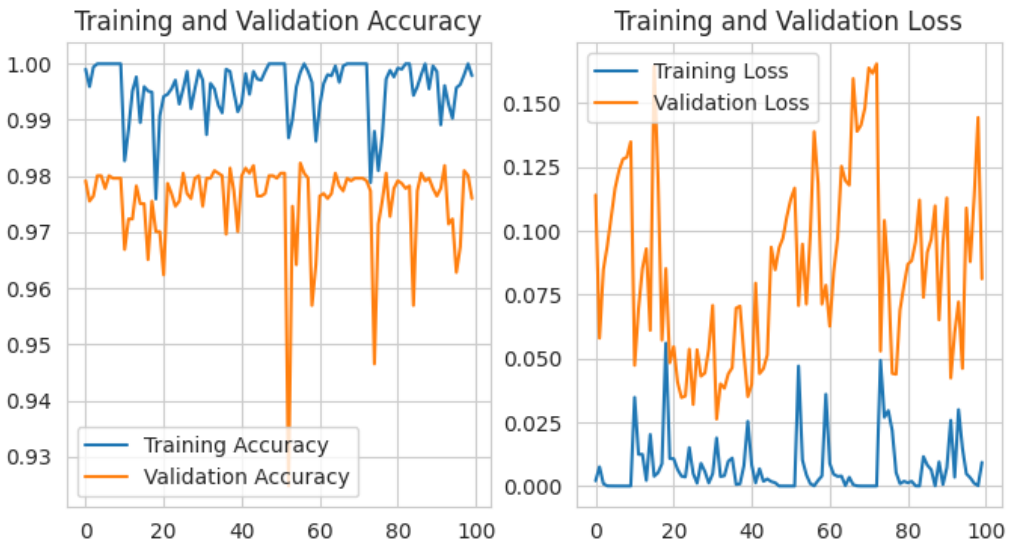

Figure 21: Accuracy and loss graph for the tuned convolutional neural network.

5.3 Long Short-Term Memory

Looking at the accuracy and loss graph in Figure 24, the lstm seems to be able to learn the data. However, the performance metrics in Figure 25 show that the model performs really bad on the test set, with an overall accuracy of 0.33.

The confusion matrix in Figure 26 shows that it classifies pretty much everything as `WALKING_DOWNSTAIRS`.

Figure 22: Performance metrics for the tuned convolutional neural network.

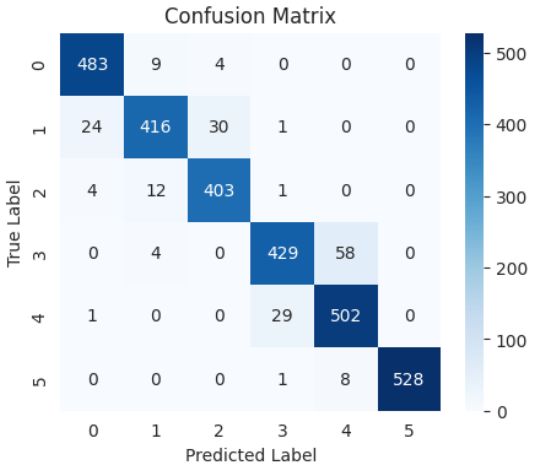

Figure 23: Confusion matrix for the tuned convolutional neural network.

We could not figure out why the models had such bad performance. The training and validation accuracy and loss graphs look fine, and the model seems to be able to learn the data. The only guess we have, is that we feed the time series data into the model in the wrong format. However, we decided not to spend more time on this and we already had models that performed well.

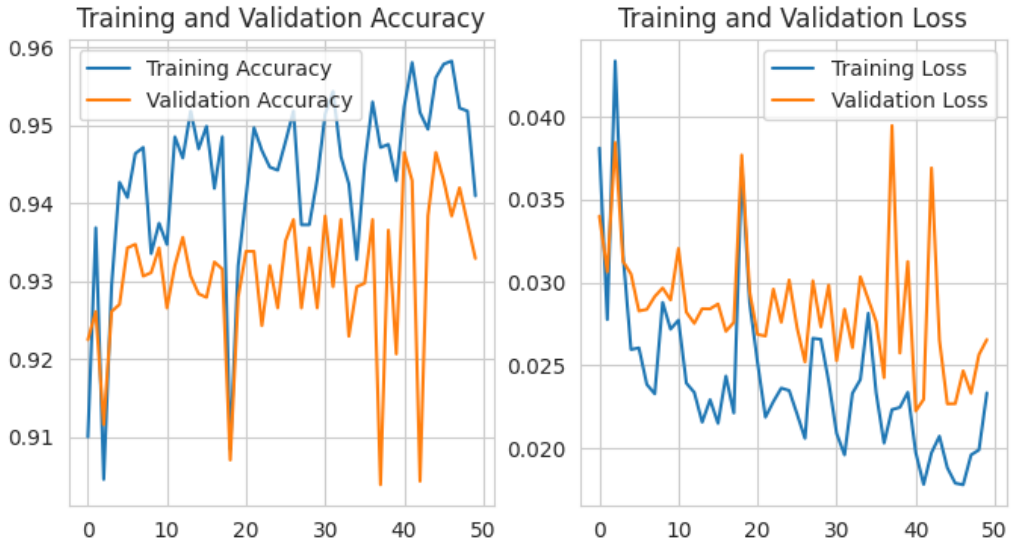

Figure 24: Accuracy and loss graph for the long short-term memory model.

5.4 Transformer

Again, looking at the accuracy and loss graph in Figure 27, the transformer seems to be able to learn the data. However, the performance metrics in Figure 28 show that the model performs really bad on the test set, with an overall accuracy of around 0.17, which is about as good as guessing. The confusion matrix in Figure 29 shows that it classifies almost everything as `WALKING_UPSTAIRS`.

	precision	recall	f1-score	support
WALKING	0.00	0.00	0.00	496
WALKING_UPSTAIRS	0.85	0.61	0.71	471
WALKING_DOWNSTAIRS	0.20	0.98	0.33	420
SITTING	0.00	0.00	0.00	491
STANDING	0.00	0.00	0.00	532
LAYING	0.87	0.54	0.67	537
accuracy			0.33	2947
macro avg	0.32	0.35	0.28	2947
weighted avg	0.32	0.33	0.28	2947

Figure 25: Performance metrics for the long short-term memory model.

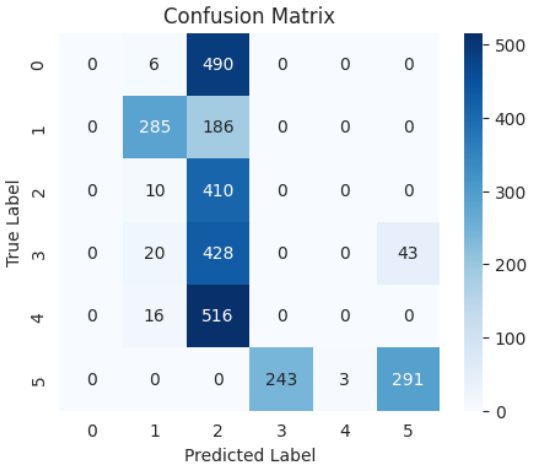

Figure 26: Confusion matrix for the long short-term memory model.

We think that we have the same problem as with the lstm model.

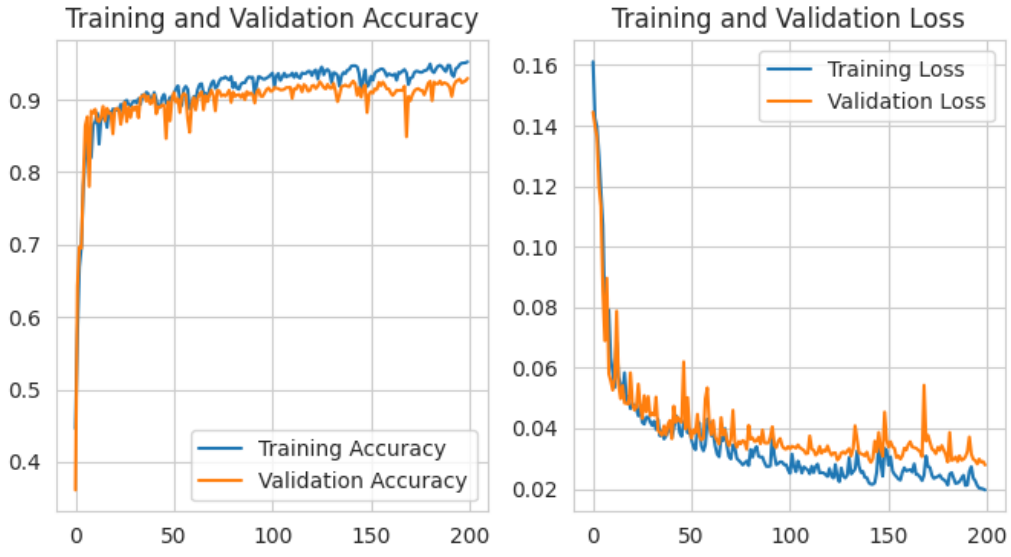

Figure 27: Accuracy and loss graph for the transformer model.

	precision	recall	f1-score	support
WALKING	0.00	0.00	0.00	496
WALKING_UPSTAIRS	0.19	0.99	0.32	471
WALKING_DOWNSTAIRS	0.44	0.05	0.10	420
SITTING	0.00	0.00	0.00	491
STANDING	0.00	0.00	0.00	532
LAYING	0.00	0.00	0.00	537
accuracy			0.17	2947
macro avg	0.11	0.17	0.07	2947
weighted avg	0.09	0.17	0.06	2947

Figure 28: Performance metrics for the transformer model.

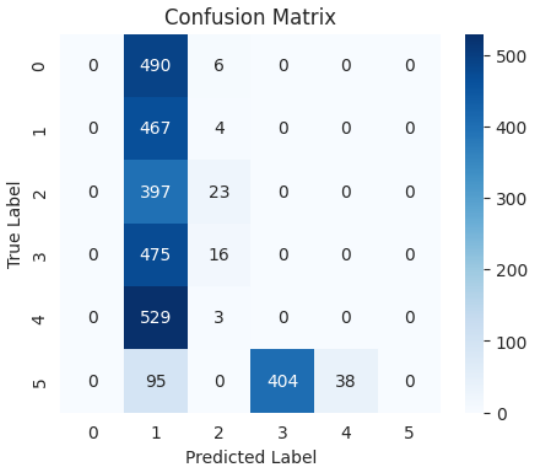

Figure 29: Confusion matrix for the transformer model.

6 Conclusion

In this project we have implemented a few simple but effective models for classifying the human activities of the UCI HAR dataset. We have shown that even the simplest models can achieve a high accuracy on this dataset and with even a little bit of tuning, we can achieve a very high accuracy.

We learned that tuning hyperparameters is crucial even for simple models, and can have a big impact on the performance. There are a lot of different ways to tune hyperparameters, and we have only used `RandomizedSearch` in this project, but it would be interesting to try out other methods such as `GridSearch` or `BayesianSearch`.

We also learned that even with a simple smartphone, we already have more than enough data to pretty accurately classify human activities. Which could be used, for example, in a fitness app to track the activities of the user. It also demonstrates to us that smartphone manufacturers or app developers could easily track our activities if they wanted to. Which is a bit scary and shows that people should be careful with what apps they install on phones and what permissions they are given. Not every app should have access to the sensors of the phone.

The authors of the dataset recorded the data with the smartphone placed on the waist of the user instead of in their pocket. It would be interesting to see how changing the position of the smartphone would affect the accuracy of the models. We assume that it would require a completely retrained model, as the position of the smartphone would affect the data collected by the sensors.

We could have invested more time in getting the long-short-term-memory network and the transformer network to work. Working with the raw data instead of the preprocessed data, which would allow for real-time classification of the activities. With the preprocessed data we used for the other models, we need to wait for the data to be collected before we could classify the activities. Which depending on the use case, could be a problem.

References

[1] OpenAI, *Chat Generative Pre-trained Transformer (ChatGPT)*, Accessed: Nov. 25, 2023; Version: 4, 2023. [Online]. Available: https://www.openai.com/chatgpt.

[2] X. Li, P. Zhao, M. Wu, Z. Chen, and L. Zhang, "Deep learning for human activity recognition," *Neurocomputing*, vol. 444, pp. 214–216, 2021, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.11.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220318063.

[3] S. Zhang, Y. Li, S. Zhang, *et al.*, "Deep learning in human activity recognition with wearable sensors: A review on advances," *Sensors*, vol. 22, no. 4, 2022, ISSN: 1424-8220. DOI: 10.3390/s22041476. [Online]. Available: https://www.mdpi.com/1424-8220/22/4/1476.

[4] I. A. Bustoni, I Hidayatulloh, A. M. Ningtyas, A Purwaningsih, and S. N. Azhari, "Classification methods performance on human activity recognition," *Journal of Physics: Conference Series*, vol. 1456, no. 1, p. 012 027, 2020. DOI: 10.1088/1742-6596/1456/1/012027. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1456/1/012027.

[5] Y. Zhao, R. Yang, G. Chevalier, X. Xu, and Z. Zhang, "Deep residual bidir-lstm for human activity recognition using wearable sensors," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–13, 2018.

[6] F. Gu, M.-H. Chung, M. Chignell, S. Valaee, B. Zhou, and X. Liu, "A survey on deep learning for human activity recognition," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–34, 2021.

[7] J. Yang, M. N. Nguyen, P. P. San, X. Li, and S. Krishnaswamy, "Deep convolutional neural networks on multichannel time series for human activity recognition.," in *Ijcai*, Buenos Aires, Argentina, vol. 15, 2015, pp. 3995–4001.

[8] A. P. Creagh, F. Lipsmeier, M. Lindemann, and M. D. Vos, "Interpretable deep learning for the remote characterisation of ambulation in multiple sclerosis using smartphones," *Scientific Reports*, vol. 11, no. 1, p. 14 301, 2021.

[9] D. Dennis, C. Pabbaraju, H. V. Simhadri, and P. Jain, "Multiple instance learning for efficient sequential data classification on resource-constrained devices," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[10] T. Watanabe, *Deep learning (and machine learning) for human activity recognition*, https://github.com/takumiw/Deep-Learning-for-Human-Activity-Recognition, 2020.

[11] U. B. R. Paila, *Human activity recognition using deep learning*, https://github.com/UdiBhaskar/Human-Activity-Recognition--Using-Deep-NN, 2018.

[12] D. Bradway and B. Himmetoglu, *Human activity recognition (har)*, https://github.com/takumiw/Deep-Learning-for-Human-Activity-Recognition, 2017.

[13] J. Reyes-Ortiz, D. Anguita, A. Ghio, L. Oneto, and X. Parra, *Human activity recognition using smartphones*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C54S4K, 2012.

[14] J. L. R. Ortiz, *Activity Recognition Experiment Using Smartphone Sensors.* [Online; accessed 25. Nov. 2023], Oct. 2012. [Online]. Available: https://www.youtube.com/watch?v=XOEN9W05_4A.

[15] M. L. Waskom, "Seaborn: Statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. DOI: 10.21105/joss.03021. [Online]. Available: https://doi.org/10.21105/joss.03021.

[16] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.