Using Machine Learning for Activity Recognition in Running Exercise

Patrik Svensson and Erik Wendel

***Abstract*—Human activity recognition (HAR) is a growing area within machine learning as the possible applications are vast, especially with the growing amount of collectable sensor data as Internet of Things-devices are becoming more accessible. This project aims to contribute to HAR by developing two supervised machine learning algorithms that are able to distinguish between four different human activities. We collected data from the tri-axial accelerometer in two different smartphones while doing these activities, and put together a dataset. The algorithms that were used was a convolutional neural network (CNN) and a support vector machine (SVM), and they were applied to the dataset separately. The results show that it is possible to accurately classify the activities using the algorithms and that a short time window of 3 seconds is enough to classify the activities with an accuracy of over 99% with both algorithms. The SVM outperformed the CNN slightly. We also discuss the result and continuations of this study.**

***Sammanfattning*—Mänsklig aktivitetsigenkänning (HAR) är ett växande område inom maskininlärning då de möjliga applikationerna är stora, speciellt med den växande mängd insamlingsbar sensordata då Internet of Things-enheter blir mer åtkomliga. Detta projekt siktar på att bidra till HAR genom att utveckla två algoritmer som kan urskilja mellan fyra olika mänskliga aktiviteter. Vi samlade in data från den treaxlade accelerometern i två olika smarta telefoner medans dessa aktiviteter utfördes, och satte ihop ett dataset. Algoritmerna som användes var ett faltande neuralt nätverk och en stödvektormaskin, och de applicerades separat på datasetet. Resultaten visar att det är möjligt att med säkerhet klassificera aktiviteterna genom att använda dessa algoritmer och att ett kort tidsfönster med 3 sekunder av data är tillräckligt för att klassificera med en säkerhet på över 99% med båda algoritmerna. Stödvektormaskinen presterade något bättre än det neurala nätverket. Vi diskuterar även resultatet och fortsatta studier.**

***Index Terms*—Activity recognition, HAR, accelerometer, SVM, CNN, mobile sensor**

Supervisors: *Afsaneh Mahmoudi Benhangi*

TRITA number: TRITA-EECS-EX-2021:178

I. Introduction

Machine learning is an area of computer science that dates back to the 1950s [1]. Over the years it has improved considerably and today it can be found in many different areas of application. There are many different types of algorithms used within machine learning. One group of them are supervised learning algorithms which all have in common that their function is to classify novel data by being trained by previous data. Machine learning is a growing field and new applications for machine learning are found every year.

Activity recognition is also a field of growing interest within computing algorithms. It can be used for medical purposes and can for instance be used to recognize if an elderly person is falling to the floor, possibly from having a stroke, and alert medical staff. Collecting data and statistics in the field of sports or exercise enables the user to track progress. It can also be used to track peoples travelling behaviour and used for city planning and public transport planning.

The purpose of this project is to apply machine learning algorithms on sensor data collected from mobile phones to distinguish what the user is doing. Some boundaries had to be set on how many activities the algorithm should be able to distinguish between. The project group decided that the models to be developed should be able to recognize four different activities. The activities the algorithms should be able to distinguish between are running, walking, standing still and walking in stairs.

II. Theory

Two algorithms were chosen for this project for the purpose of comparing the end results. The project group decided to use a support vector machine (SVM) and a convolutional neural network (CNN).

The support vector machine [2], published in 1992, is one of the two algorithms that was used in this project and the other one is a neural network algorithm which concept idea dates back to 1944 according to [3].

A. Convolutional Neural Networks

Neural networks are algorithms whose structure is inspired by the human brain, and [3] describes neural networks as follows. A neural network is made up from many processing nodes which are interconnected in layers. Each layer consists of the processing nodes and different layers may have different amounts of nodes. The data is sent to the first layer, processed and then sent to the next layer and so on until it reaches the final output layer. Data does not travel backwards, rendering this a feed-forward algorithm. The final layer consists of as many nodes as there are possible outcomes.

The processing in each node is done by giving each node input a weight, and calculating the node output from the weighted inputs. All weights in all nodes determine what the algorithm as a whole will output when given an input. The algorithm is trained by first randomizing the weights and then inputting data that has been labelled manually, which is called training data, so the algorithm is able to see if it has arrived at the correct answer. The weights get adjusted during the training process so that the algorithm performs better than it

initially did. The node output is then sent to nodes in the next layer of the network.

In a convolutional neural network, some layers are convolutional layers, where a convolution is applied to the data. Time discrete convolution can be written as

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{1}$$

where n is the sample, x is the input signal, h is the impulse response and y is the output from the convolutional operation. Convolutional layers are used to extract specific features and recognizing patterns in the data.

A problem when designing neural networks is overfitting. Overfitting occurs when a network is too large in proportion to the complexity of the task to be performed. This results in that the network learns all of the training data's every detail and misses the bigger picture of the problem at hand, often referred to as having poor generalization capabilities. Training the machine model over the training data in high detail results in poor performance over the test data. Test data is the data that gets sent to the algorithm after the training process to validate how well the machine learning model performs over unseen data. There are many different techniques to reduce overfitting. One way is to use a regulator which regulates how the nodes learn. One commonly used regulator is a dropout regulator [4] which is applied on a layer of nodes to randomly suppress the activation of a set percentage of the nodes. This forces the network to learn new pathways to solve the problem which creates a more robust model. However, using the dropout regulators slows down the training process.

B. Support Vector Machine

Support vector machines are the most well known algorithm in a larger class of algorithms known as kernel-based methods, and the concept behind SVMs are, according to [5], described as follows. The idea is to separate values by creating a hyperplane that separates datapoints in the so-called feature input space. The features are features of the data that are relevant for classification. Examples of features for SVMs are mean value and standard deviation, and the features get extracted from the raw data beforehand; the algorithm never sees the raw data. The coordinates for a datapoint in the input feature space are the extracted features that describe the datapoint. The input feature space has a dimensionality of $\mathbb{R}^n$ where n is the number of features, and the hyperplane will have a dimensionality of $\mathbb{R}^{n-1}$ and will divide the feature input space into two subspaces. SVMs are only able to classify between two types of data at a time. It is however possible to make multiple SVMs to solve classification problems with more than two classes.

If the datapoints in the input space are not linearly separable, which is common, the data is mapped by a so-called kernel function to another space where it is linearly separable. This space always has higher dimensionality than the input feature space. It is always possible to find a space where the input data is linearly separable if the dimensionality of the space the data gets mapped into is high enough. It then becomes a problem of finding the hyperplane that is dividing the datapoints in an optimal way by maximizing the distance to the points and handling noisy values as described in [6]. This maximum margin hyperplane is developed through the training process. When the algorithm is exposed to new data after the training process is done, the new datapoints are inputted and the algorithm classsifies every datapoint according to which side of the hyperplane it is placed. The new data has to be handled in the same way as the training data for the algorithm to be effective. If the same features are not identically extracted from the training data and the novel data, SVMs cannot be expected to work well.

As mentioned above, if the data is not linearly separable in the input space, a kernel function has to be utilized which maps the points to another space. A kernel that can be used with SVMs is the radial basis function (RBF) [7] which uses the Gaussian form

$$K(x, y) = e^{-\frac{1}{\sigma^2}||x-y||^2}, \tag{2}$$

where x is the input, y is the target value and σ is the width of the function according to [8]. A relevant parameter for SVMs with RBF kernels is

$$\gamma = \frac{1}{\sigma^2}, \tag{3}$$

as it decides the flexibility of how the hyperplane can be drawn. Too large values of γ cause overfitting which leads to a poor generalization and performance on new data, similar to CNNs. Too small values of γ could cause an ineffective hyperplane fit to the data, resulting in a poor performance accuracy.

Other examples of kernels are linear ones and polynomial ones of different degrees. When choosing which kernel to use for a SVM, the trial and error approach is often the only realistic one according to [9].

To handle noisy or overlapping datapoints, a so-called soft margin [9] are implemented in SVMs to lessen the effect of errors in the data. This effectively means that some datapoints are allowed to be on the wrong side of the hyperplane. It is up to the designer of the SVM to decide how many datapoints are allowed on the wrong side and how far into the wrong side the data are allowed to be. There is a trade-off between the size of the margin and how well SVMs handle errors in the data. The parameter that decides how far on the wrong side the datapoints are allowed to be are decided by the soft-margin parameter according to [10], and it is usually denoted C. The size of the flexibility parameter together with the soft-margin parameter C are of importance when designing a RBF SVM.

Due to the binary nature of SVMs they can only distinguish between two groups of data, in our case two activities. They can however be modified in multiple ways to be able to classify more than two activities according to [5]. One way is to compare data from two activities at a time (one-against-one). Another way is to construct a number of SVMs that is equal to the number of classes to be distinguished, where each compares one class against all other data (one-against-all). This means each SVM answer the binary question if a datapoint belongs to that SVMs class or not according to [9].

A SVM approach will always converge to the same end result when presented with identical train and test data. This is not the case with neural networks, as the weights in the neurons are randomized before each time it trains, meaning that the final trained network might behave differently between executions.

C. Data Processing Theory

Data processing is manipulation of data. A part of data processing is to make sure the data set is balanced. An imbalanced data set occurs when different classes in a data set have different amounts of samples. Training machine learning models on highly imbalanced data sets can lead to the models having difficulties classifying the classes with less samples because of the unequal amount of training samples trained on. One way to balance an imbalanced data set is to use random undersampling. This works by randomly removing samples from the class with the most amount of data until a more balanced data set is acquired. Another technique used for balancing the data is a SMOTE (Synthetic Minority Oversampling Technique) oversampler. SMOTE sampling works by plotting every sample from a class in a space and draws lines in to its closest neighbors. It then creates new synthetic samples placed on the lines, as described in [11].

If magnitudes in the data set varies highly, machine learning models may exhibit problems learning from the low magnitude data values correctly, and could be biased to learning more from the high magnitude data. A scaler or normalization function can be implemented and applied to the input data to prevent this. One way to do this is to subtract the mean value of the input data from the input and dividing it by the standard deviation of the data. This ensures that the standard deviation of the magnitudes are of the same order which will result in better performance from many machine learning algorithms according to [12].

III. Method

Before developing the algorithms, a few steps had to be taken. Initially, a data set had to be obtained which was done by collecting smartphone accelerometer data. Then the data were processed to enable extraction of features that the algorithms would be able to take it as input. The two algorithms were then designed and applied. The programming language used was Python together with the TensorFlow library and scikit-learn as well as support libraries like Pandas, NumPy and Matplotlib.

A. Data Collection

To collect consistent data, a standardized way of collecting the data was developed. The phone was placed vertically in the trouser pocket of the person collecting the data, and the activity were then performed for an unrestricted amount of time and the recordings were stopped afterwards. Two different phones were used by two different people for collecting data. The same data collection app was used with the same collector settings on both devices. The data values collected were the x-, y-, and z-axis from the accelerometer and it was collected at 50 Hz and saved locally in a comma separated value (CSV) file which were later used in the processing. The amount of data collected for each activity can be seen in table I and an example of the data can be seen in fig 1.

TABLE I
Amount of data per activity

Activity	Time collected (minutes)	Number of snippets
Running	65	1300
Walking	56	1120
Standing	48	960
Stairs	53	1060

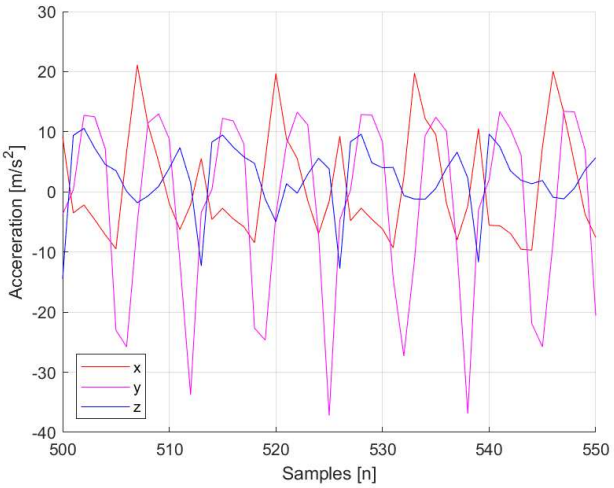

Fig. 1. Example of collected data for one second of the running activity. See appendix for examples of all four activities.

B. Data Preprocessing and Feature Extraction

The data were then preprocessed. The first and last eight seconds from every recorded CSV-file were cut away to account for the time it takes from starting the recording until the phone is in place in the pocket. Then each collected 50Hz datapoint in each file is labeled with the activity it contains and were then merged into one CSV-file.

As mentioned in the theory, the performance will be better with smaller differences in magnitude in the data so a rescaling of the data was made.

As different amounts of data were collected from the different activities, an oversampler using SMOTE was implemented to make up for the unbalanced data. It was set up to create new data samples for the activities with the least recorded data until all activities have the same number of data samples as the one with the most recorded data. This was only applied to the training data so the test data would consist of recorded data without any created data.

All the data in the CSV-file was then portioned into snippets, each snippet containing 200 samples which equals four seconds of data. The initial length of four seconds was chosen arbitrarily. Every snippet that included more than one activity

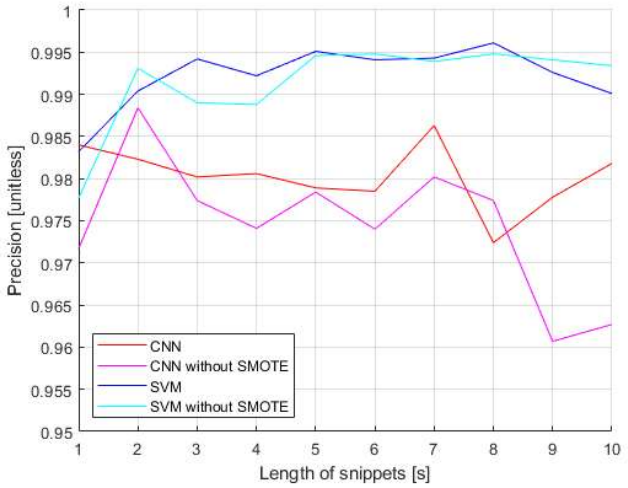

Fig. 2. Graph over the performances with and without SMOTE oversampling using different snippet lengths

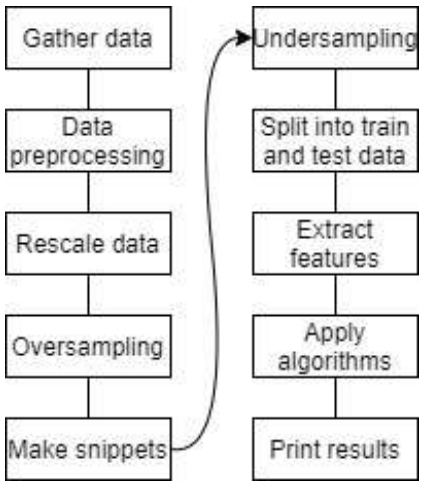

Fig. 3. Block diagram of program

was removed to clean up the data and help the network only associate the right data to its corresponding activity. The best value for the length of the snippets were found later when we had algorithms that were working. It was found experimentally by trying various lengths. The lengths that were tested were 1 to 10 seconds long and the results from the experiment can be seen in figure 2. The graphs that use the SMOTE oversampling were the ones considered when deciding the length, as the oversampling is implemented in the algorithms. As dividing the data into longer snippets gives fewer snippets, a short time was preferred to give more snippets for the algorithms to train and test on, giving higher resolution in the result. Following this logic, the length of snippets for the final algorithms were decided to three seconds long. The length was kept the same for both algorithms to enable an objective comparison of the final result.

An undersampler was implemented to make the amount of snippets equal for each class after some of the values had been removed when combining the samples into snippets. This was only a few samples but it made sure that all the data was completely balanced.

The snippets were then split up into training and test data, with 80% of the data being training data and the rest being test data. Which 20% of the snippets used as test data were decided randomly each time the program were executed. The training and test data were the same for both of the algorithms.

The input to the CNN was the data in the snippets (x-, y- and z-axis acceleration). This is an acceptable input for CNNs as they need strings of information to be able to perform the convolution calculation, see equation (1), in the convolutional layers.

Other features were used for the SVM as the input space would be of very high dimensionality if the raw samples were used and likely to be less optimal due to the algorithm having to map every sample in every snippet via the kernel. Before finding what features would be best for the SVM, a basic SVM was constructed which could take input data and produce a result. Then features were extracted from the snippets and the features were the statistical measures mean value, standard deviation, kurtosis and skew. The values were extracted for each coordinate axis from each snippet. Different combinations of the values were used with the SVM and the results recorded, and it showed that the SVM performed best when only mean value and standard deviation were used as inputs. Therefore, the other features were discarded and mean value and standard deviation were the features used for the final SVM. Because both of the features were extracted for the x-, y- and z-axis separately, the final SVM had six input features in total.

The full flow of data can be seen in figure 3.

C. Algorithm design

The CNN used was a network consisting of two convolutional layers and two artificial neural network layers, which are not convoluting. A dropout was implemented between every layer to reduce overfitting. The output from the first two layers were then transformed to get the right dimension for the rest of the layers. The first, second, third and fourth layers contained 16, 32, 64 and 4 nodes respectively.

For the SVM, the kernel decision was made through trial and error, as recommended in the theory. The kernels that were tried were linear, polynomials of various degrees and RBF and the RBF showed to produce the best results.

To find good values for the flexibility parameter γ and the soft margin parameter C, multiple executions were carried out with different values for the parameters. The test results can be seen in table II. This gave an understanding of what values were appropriate. The same technique were then applied for values around the optimal ones in the table. The highest precision was attained with the parameter values $C = 30$ and $\gamma = 3$.

The algorithms were then considered finished and were executed one last time and the result printed out as confusion matrices.

TABLE II
TABLE OVER PRECISION FOR SOFT MARGIN PARAMETER AND FLEXIBILITY PARAMETER FOR THE SVM

γ \ C	0.1	1	10	100	1000
0.01	90.2%	92.6%	97.1%	97.5%	97.6%
0.1	96.4%	97.3%	97.6%	97.9%	98.4%
1	97.5%	98.1%	98.8%	99.5%	99.2%
10	98.8%	99.4%	99.4%	99.4%	99.4%
100	88.9%	98.3%	98.4%	98.4%	98.4%

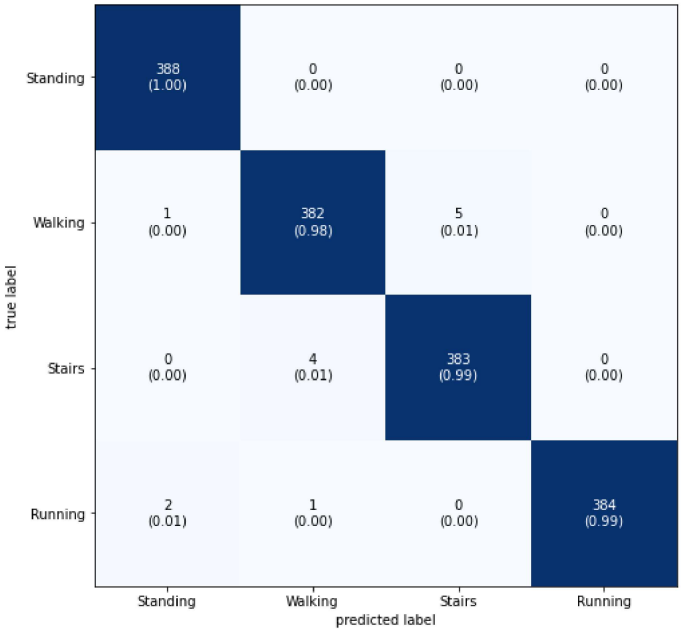

Fig. 4. Confusion matrix for the CNN algorithm

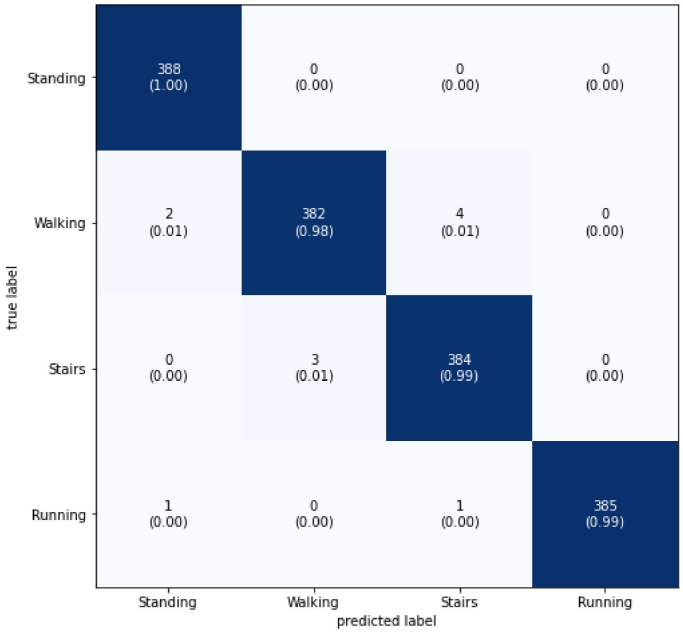

Fig. 5. Confusion matrix for the SVM algorithm

IV. Results

The confusion matrices with the final results can be seen in figure 4 and figure 5. The confusion matrices display how well respective algorithm performed for the four different activities. The rows show what activity a snippet contained was and the columns show what the algorithm predicted it to be, giving a visual way of inspecting how the algorithm performed for each activity. It also shows how many snippets that were wrongly classified and which activities that were most misclassified.

The mean accuracy for the SVM were 99.29% and the mean accuracy for the CNN were 99.16%.

V. Discussion

The results of this project shows that four different activities can be classified with over 99% mean accuracy using machine learning. In this project, the SVM outperformed the CNN slightly. The overall performance was better than anticipated. The high accuracy could be explained by a clean dataset which might be a negative thing for the final products, as the algorithms have not been trained on noisy data and it is not certain how they will perform on imperfect data. Different people have different speeds of walking, running and walking in stairs and collecting data from more than two people is likely to lead to better generalization. It should also be noted that the SMOTE oversampler makes new data points from the existing ones, making the training data even more homogeneous, which might contribute to poor generalization. However, all of the test data were real data with no oversampling applied, implying some generalization in the result. To improve the result, collecting data from another group of people and testing the algorithms on that data would be of interest to see how well the algorithms perform on general data.

Another notable result of this project is that a time window of three seconds was enough to classify the activity with high accuracy, and even one second is enough to classify with relatively high accuracy as seen in figure 2. This implies that it should be possible to design a mobile application that collects the data and continuously computes the activity recognition directly on the smart phone, with a delay of just the length of the snippet plus the short time it takes to portion the data into a snippet and pass it through an already trained algorithm. In the application in running exercise, as was the scope of this project, this real time feature might not be of high importance. But it indicates that it would be possible to develop a mobile application that almost instantly recognizes if the user is falling to the floor, which can be used in elderly care and for alerting medical staff if this were to happen, possibly reducing the number of deaths caused by stroke and other falls in the elderly. Another continuation of this project could be an application that recognizes if a car driver experiences a very sudden retardation, indicating a dangerous impact, and alert medical help.

From figure 2 it is clear that the length of the snippets did not have a significant effect on the accuracy of the algorithms. One thing to note is that because of the random properties of the train test split and the SMOTE sampling the result

will change slightly between executions. The CNN network also has a random nature as mentioned in the theory. This inconsistency in the result could be lessened by executing the algorithms multiple times and calculating the mean of the results for each snippet length.

It can be discussed why the RBF kernel for the SVM performed the best. The explanation could be that it is because of the RBF kernels suitability for classifying non-linear data sets, according to [7]. We also noted that when using different kernels, some values in the confusion matrix barely changed (confusion between running and standing) while others varied heavily (confusion between stairs and standing). This implies that the choice of kernel had lower impact on the values that are relatively separated in the feature input space and higher impact on the values that are overlapping, which is in accordance with the theory.

We found that the algorithms never mistake the activities standing with running or vice versa. This in theory is not surprising, but it shows that the data is likely to not have large amounts of noisy, erroneous values.

There are notable limitations in the result. We know that the algorithms work with collected data from the specific accelerometer data collection app used, and on the two different phones. Problems might arise if one were to generalize this project on other phones or with other apps. Therefore, we cannot draw the conclusion that this accuracy result is relevant for all models of smart phones or with any data collection application.

There are also some methodological limitations as the authors are relatively inexperienced in the area of machine learning and numerical analysis and optimization. A lot of the design choices were made with a trial and error approach. With some parameters, e.g. choice of kernel function, the trial and error approach might be preferred as mentioned in the theory. However, a more methodical approach to the choice of the other parameters might have lead to more optimal values, which would result in better performance. Also, the authors have limited knowledge of the linear algebra mathematics behind machine learning, for instance Hilbert spaces and Lagrangian multipliers, resulting in a less profound understanding of how the algorithms work which might have affected the design decisions and the end result.

VI. Conclusion

This project implemented two machine learning algorithms that accurately classifies human activities from tri-axial accelerometer data when given data from three second long snippets. The methodology was trial and error and iterative optimization to find the parameters for the algorithms. The highly accurate results is possibly explained by a clean and small dataset, which might imply poor generalization. The results show that it is possible to design algorithms that accurately classify between standing, walking, running and walking in stairs with three second long snippets of accelerometer data collected from phones by two people. The SVM outperformed the CNN slightly in this project.

Continuations of this research could be into the medical field or fitness field, by developing applications that can recognize falls or efficiently track exercise data respectively.

VII. Acknowledgement

We would like to thank our supervisor Afsaneh Mahmoudi Benhangi for her help throughout this project.

Appendix A

Examples of collected data for the four activities

References

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 70–105, Oct. 1959.

[2] B. B. et al., "A training algorithm for optimal margin classifier," *In Proc. 5th ACM Workshop on Computational Learning Theory*, vol. 0, pp. 144–152, Jul. 1992.

[3] L. Hardesty. (2017, Apr.) Mit news. Massachusetts Institute of Technology, Cambridge, MA, USA. [Online]. Available: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

[4] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 8609–8613.

[5] C. Campbell and Y. Ying, *Learning with Support Vector Machines*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. San Rafael: Morgan Claypool Publishers, 2010, vol. 5, no. 1.

[6] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 09 1995.

[7] J. Suriya Prakash, K. Annamalai Vignesh, C. Ashok, and R. Adithyan, "Multi class support vector machines classifier for machine vision application," in *2012 International Conference on Machine Vision and Image Processing (MVIP)*, 2012, pp. 197–199.

[8] M. Mamat and S. A. Samad, "Performance of radial basis function and support vector machine in time series forecasting," in *2010 International Conference on Intelligent and Advanced Systems*, 2010, pp. 1–4.

[9] W. S. Noble, "What is a support vector machine?" *Nature Biotechnology*, vol. 24, no. 12, pp. 1565–1567, Dec 2006. [Online]. Available: https://doi.org/10.1038/

[10] A. Ben-Hur and J. Weston, *A User's Guide to Support Vector Machines*. Totowa, NJ: Humana Press, 2010, pp. 223–239. [Online]. Available: https://doi.org/10.1007/978-1-60327-241-4_13

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[12] F. e. a. Pedregosa, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.