Human Activity Recognition

Jake Sant[1] [117699M], Aiden Williams[2] [372001L], Ethan Zammit[3] [4802L]
Department of Artificial Intelligence
University of Malta
jake.sant.18@um.edu.mt[1], *aiden.williams.19@um.edu.mt*[2], *ethan.zammit.19@um.edu.mt*[3]

***Abstract*—Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.**

***Keywords*—Human activity, Classification, Deep learning, SVM**

I. Introduction

Human Activity

II. Aims and Objectives

The main objective our group aimed to accomplish was to understand and make use of deep learning and classical machine learning techniques to correctly classify a number of stationary and moving activities in the UCI dataset [**?**]. We also aimed to build our own dataset, consisting of many records of activities. We then wished to apply and evaluate the classification methods on our own database. The new activities in our own dataset are laying, sitting, standing, walking straight, walking upstairs, walking downstairs, driving, jumping, swimming and performing push ups.

III. Background Research

A. Sensing Devices

When considering applications in HAR, a range of sensors may be applied to recognize the activities being carried out. It depends on the application, such as using external sensors such as a camera, a microphone or as used in [A system for change detection and human recognition in voxel space using the Microsoft Kinect sensor.] using the Microsoft Kinect as a 3d sensing device applied to HAR. Unfortunately, these are usually limited to a static place, as the equipment to externally scan the activities needs a setup. On the other hand, there are wearable/portable sensing devices, such as accelerometers, gyroscopes or possibly heart rate monitors, which also have their own challenges. There were even methods such as [Emmanuel Tapia, Stephen Intille, Louis Lopez, and Kent Larson. The design of a portable kit of wireless sensors for naturalistic data collection. Pervasive Computing, pages 117-134] where the two types of sensing devices were combined, aiming to analyze intricate activities.

B. Data Acquisition

During the process of data collection, care should be taken to keep conditions as natural as possible, as large deviation from natural conditions might make the model unusable, as conditions may be drastically different. The variety must also be maintained as to aim for a robust model which can handle different conditions, noise levels, users and environments [Human Activity and Motion Disorder Recognition:Towards Smarter Interactive Cognitive Environments Jorge L. Reyes-Ortiz1,2, Alessandro Ghio1, Davide Anguita1, Xavier Parra2, Joan Cabestany2, Andreu Catal'a2].

C. Signal Processing & Feature Extraction

A robust and reliable signal processing pipeline is essential to convert raw inertial data into input to a model, as the features from the data need to be extracted and highlighted to allow the classifiers to distinguish between the activities using the reproducible characteristics, such as statistical results on windows of data. Data windowing is a very common practice when considering sensory data, as several statistical tools are only unlocked on bundled data, and allow for richer and denser information per entry. An element of overlap between windows is also often employed, as the overlap allows for smoother transitions between activities. [Human Activity and Motion Disorder Recognition:Towards Smarter Interactive Cognitive Environments Jorge L. Reyes-Ortiz1,2, Alessandro Ghio1, Davide Anguita1, Xavier Parra2, Joan Cabestany2, Andreu Catal'a2]. Albeit the importance of reliable classification one also need to consider the processing time required for the signal processing pipeline, as in some applications real-time classifications are necessary, and if the feature vector is excessively large, performance may be heavily impacted.

D. Support Vector Machines

Support Vector Machines (or SVMs) are a type of classifier used for classification and regression. SVM seeks to classify samples as different classes on a hyperplane in multidimensional space. The classification process generates multiple hyperplanes to find the one which minimises an error value.

Generally, support vector machines are used for binary classification problems. However, they can be adapted to multiclass problems by converting or reducing them into a set of multiple binary classification problems. However some samples cannot be classified linearly, hence SVMs can also make use of kernels to transform non-linear space into linear space, which was proposed by Boser et al. [**?**].

E. *Convolutional Neural Networks*

The Convolutional Neural Network (CNN) is a type of deep learning neural network model commonly applied for computer vision applications. Like other deep learning neural networks, CNNs consist of an input layer, any number of hidden layers, and an output layer. Unique to the CNN is the use of the convolutional layer, these layers use a kernel that goes over the input tensor, causing the convolution. Fully connected linear layers can achieve the same results however, for large inputs FC layers will have many weights. With convolutional layers the number of these weights is reduced. Each layer has its own activation function, which acts as a filter for whether a neuron should fire or not. In this project the RELU, Softmax and LogSoftmax activations where used. After a number of convolutional layers an FC layer is usually inserted, and in this project each CNN has an FC layer at the end.

IV. Literature Review

Anguit et al. [**?**] classified activities into two different categories: static and dynamic activities. Using a One-Vs-All approach and a Laplacian kernel, they obtained highly accurate rates of classification. Demrozi et al. [**?**] compiled a number of studies on human activity recognition (Human Activity Recognition using Inertial, Physiological and Environmental Sensors: A Comprehensive Survey) and found that out of 149 papers published between January 2015 and September 2019, 53 examined deep learning models and the remaining 96 were classical machine learning models.

V. Design and Methodology

A. *Data Collection*

For the data collection, a barebones Android app was developed, which reads the triaxial accelerometer and gyroscope data at a constant rate of 50hz. Additionaly, a deviceID and timestamps were appended to each entry, along with an option to label a session before starting collection, all in aim of simplifying the processing pipeline later on, and avoiding the need to manually label all the data later on.
Using this app, several activities were carried out and recorded, whilst trying to vary the conditions as much as possible, such as putting it in the left and right pockets and having it upside down or upright, aiming to diversify the data.

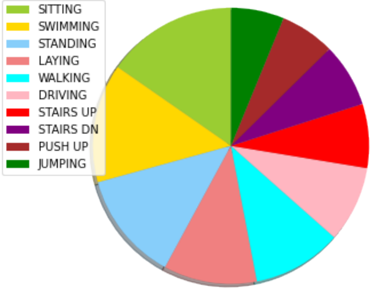


Fig. 1: Our Data Distribution

Figure **??** depicts the different amounts of data collected per activity. Unfortunately, there was a considerable difference in the amount of data per activity, where some activities such as jumping had much less data than an activity like walking. This can lead to the accuracy of classification of data being misleading as if the classifier randomly guesses walking, it is right more times than if it randomly guesses jumping.

B. *Signal Processing*

The JSON data exported from the mobile application was then ready to be imported by a python preprocessor. As is, the data is not usable by the classifiers, not only does it contain a lot of noise, but a single entry gives very little information about what is happening.
Consequently, several steps were to maximise the usability and the potential for classification of the data, such as synchronisation of gyroscope and accelerometer signals, use of a sliding window to group data, signal filtering and feature extraction using several statistical measures.

C. *Synchronisation*

Due to a limitation from the application, the gyroscope and accelerometer collections worked asynchronously, and thus the entries were likely not to be aligned properly (few ms differences). And if the phone screen was turned off, data collection would stop and restart when the screen is turned on, which would cause a substantial gap within the data of a session.
A moving average of the differences was calculated over all the data. If the average is below 10ms, the values would be considered as forming part of the same timestamp, whilst if the average value was larger, an entry was removed from the accelerometer and gyroscope data, until the average restabilized. Thie eliminates most outliers, and if there are antired out of synch by a substantial margin, they are removed. This was considered quite harshly, as several techniques such as

Fast Fourier transform rely on constant frequencies to provide desirable outputs. The first and last 2 seconds were also removed for each session, as time needs to be accounted for whilst the user puts the device in his/her pocket, which would provide inaccurate data.

D. *Sliding Window*

As carried out int [https://www.sciencedirect.com/science/articl a sliding window was used to 'group' of size 128 with a stride of 64 (taking 128 entries (2.56 seconds of data) for each window, and overlap each window by 1.28 seconds).
This is one of the most crucial steps for data processing, as one single entry does not describe a whole lot about what a user is doing. Before any preprocessing, each entry simply has six values (two triaxial vectors), which are at a single point in time. When the sliding window is introduced, we group a window's worth of entries into one, out of which we get a representation of what happens over a period (2.56 seconds in our case), where each window will represent 128 raw entries, containing 6 features (triaxial data) each. Which when feature mapped, summarizes features of this two-dimensional data frame, back into a one-dimensional entry, whilst still retaining the fact that an entry has now the added dimension of time.

E. *Signal Filtering*

A set of steps was applied to each window of entries, which will filter and prepare the data for the feature extraction.

1) *A low-pass Butterworth filter with a corner frequency of 20hz:* This removes most of the noise, as most changes in the acceleration happened at much lower frequencies, and through this separation, the most relevant changes in acceleration were kept. In fact according to [D.M. Karantonis, M.R. Narayanan, M. Mathie, N.H. Lovell, and B.G. Celler. Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring. IEEE Transactions on Information Technology in Biomedicine, 10(1):156–167, 2006.] most bodily movements are contained below 15Hz

2) *Another low pass Butterworth filter was applied with a corner frequency of 0.3hz:* This filter separates gravitational acceleration from body acceleration, since changes in gravitational acceleration happen relatively slowly, a low corner frequency can distinguish and extract these changes as gravitational acceleration.

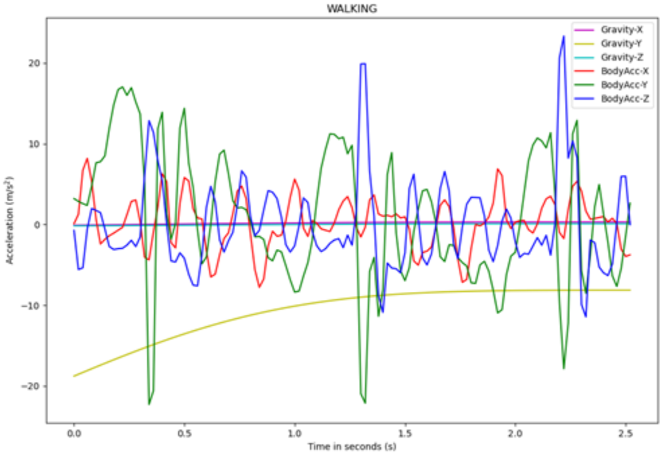


Fig. 2: The Acceleration in a single window

Figure **??** shows the different forces of acceleration experienced were plotted. The Body forces show the filtered acceleration when separated from the gravitational acceleration. At idle, these forces should be zero. The strides of the walking may also be seen as the green line fluctuates, quickly dropping when the wearer's foot descends and hits the floor. The other three stable plots are the gravitational forces experienced, these should not change as much, as the gravitational force experienced do not change much. At idle the vector of these three forces should point downwards with a magnitude of ˜9.81m/s^2.

3) *The Jerk was Calculated:* Jerk is defined as the change in acceleration and was calculated by differentiating the acceleration values by time(1/(50hz)=0.02s). This gives a better idea of how drastically the accelerations change, such as when one is jogging and the leg where the phone is sitting stops sharply as it hits the ground.

4) *The Magnitude:* The magnitude is the measure of how much the values differ from zero. This comes useful when one simply needs an idea of the amount of force applied, without considering the effects of directionality.

5) *Fast Fourier Transform:* FFT enabled the calculation of frequency components based on the time-varying signals. When considering Human Activity, one can note that there is a lot of repetitive patterns, in which case transformations such as FFT are used to calculate the discrete Fourier transform, and give a rich statistic for summarizing a window of these repetitive signals.

$https : //www-proquest-com.ejournals.um.edu.mt/docview/23017$

F. *Feature Extraction*

Excluding the label for each activity, each session in the dataset contains 589 features. These features were extracted by applying a number of different statistical measures to the different extracted signals in each sliding window.

Each signal has the statistical measures in Table 1 applied to them.

Statistical Measure	Description
mean	Mean value in the window
min	Smallest value in the window
max	Largest value in the window
std	Standard deviation
entropy	Entropy
mad	3
iqr	Inter-quartile Range
energy	3
sma	Signal magnitude area
arCoeff	3
correlation	3
angle	Angle between 2 vectors of signals
band energy	3

Table 1

The statistical measures in Table 2 are applied only to FFT signals.

Statistical Measure	Description
maxInds	1
skewness	2
kurtosis	3
meanFreq	3

Table 2

G. Signal Processing

H. Support Vector Classifier

1) Data Pre-processing: Firstly, the training and testing datasets were converted to Pandas dataframes and the labels for each activity were separated into their own variables. Using a LabelEncoder, each activity label is converted into a numerical value.

2) Comparing different classification models: In total, four different classic machine learning classifiers were used on the dataset developed by Anguita et al [**?**]. These classifiers are Gaussian Naïve Bayes, AdaBoost, Stochastic Gradient Descent and a Support Vector Classifier. These were trained and tested using the respective datasets, and their accuracy, F-beta, precision and recall scores were recorded.

Classifier	Accuracy	F-Beta	Precision	Recall
Gaussian Naïve Bayes	0.7134	0.7252	0.7555	0.7134
AdaBoost	0.4065	0.2520	0.4289	0.4065
Stochastic Gradient Descent	0.9600	0.9603	0.9605	0.9600
Support Vector Classifier	0.9668	0.9676	0.9682	0.9668

Table 3: Performance scores of each classifier

As can be seen in Table 3, the Support Vector Classifier had the highest scores, each result being over 0.96. It is for this reason that the SVC was chosen to classify the UCI dataset [**?**] and, later on, the dataset built by ourselves.

I. Classification

The RBF kernel is defined as the exponential function $exp(-\gamma|x - x'|)^2$ A primer on kernel methods, JP Vert et al, where x and x' are two feature vectors, and is the gamma parameter in the classifier. Gamma's value is scale, meaning that the parameter is the reciprocal of the number of features multiplied with the variance of the input data. For this implementation, we opted for a One-Vs-All approach. A One-Vs-All approach divides the data points into just two classes: a certain activity X and the other classes. Therefore records labelled as *SITTING* are a single class, and the other activities are treated as having a single label.

J. Convolutional Neural Networks

In the research done numerous CNN implementations designed for the UCI dataset like [include numerous] were checked. From these, [paper] presented the choice and implementation of a CNN use for HAR the best. Hence, the implementation described in this report is based on the architectures described in [the paper]. Unlike [paper authors] we did not implement a first stage dynamic-static split model and instead opted to split the labels manually. The Pytorch [reference it] library was used to implement the CNNs, and the sklearn [reference it] library to evaluate the results.

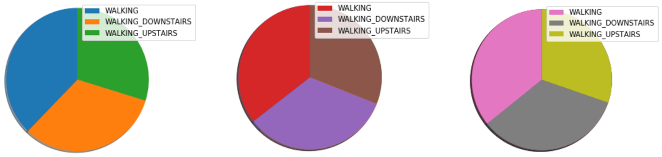
1) Data Split: The train/valid/test split was an 80/20 train/test split, then 80/20 train/validation split for the UCI dataset, and a 90/10 train/test split, then 90/10 train/validation split for Our Dataset.

2) Dataset Object: To facilitate this dynamic/static split we defined a Python dictionary with each label having a number assigned to it, starting from 0. The numbering needs to start from 0 as the output of neural networks implemented in Pytorch always start from 0. Another requirement for implementing neural networks in Pytorch is to implement a custom Dataset object. The function of this object is to read the data from a source and define its X, data and Y, labels counterparts. The Pandas [reference it] library was used at this stage due to its use of the numpy [reference it] library and efficient data management. It is important that the X component is in the shape: length(data.columns), 1, length(data.rows)).

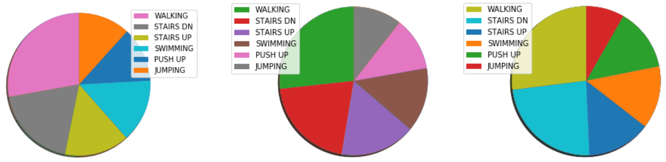
The dataset object was initialised 3 times, for the Train, Validation and Testing data.

The pie charts included below show the training, validation and testing label distribution in order.

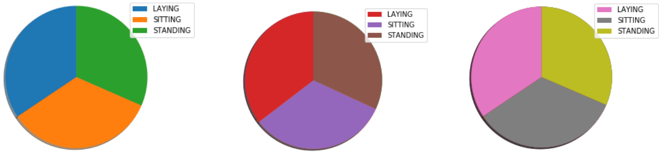
In development, it was noted that the data distribution of the dynamic activities in our dataset was unequal. This was causing issues with the CNN's accuracy. To combat this added functionality was added with the aim of increase the quality of the model's output. In this project the number of rows for each



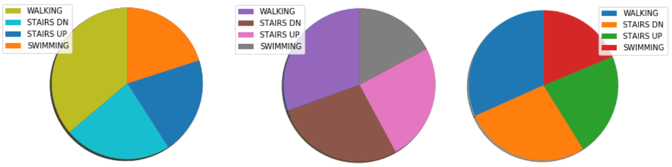
UCI Dynamic Label Distribution



OD Dynamic Label Distribution



UCI Static Label Distribution



OD Static Label Distribution

eligible label was stored. Using the functionality provided by the Pandas library, the index for each label was stored in a list linked to its respective label. Using Python list slicing these lists where cut down to their lowest common length. These indices where then converted into a new Pandas dataframe and this balanced dataset was used for the dynamic model.

Finally the dataset objects are used to initialise a Dataloader object. This object gives us the functionality of delivering batches as inputs to the models at training and testing. It was found that a batch size of 32 worked best for the Static models, and a batch size of 64 worked best for the dynamic models.

3) Model Creation: The CNN models where implemented following the design of the below diagrams. Every model used the Cross Entropy as their loss function and the ADAM optimizer. The learning rate for the UCI Dataset Models, and Our Dataset Static model was set at 0.0005 and were trained for 10 epochs. The learning rate for the Our Dataset Dynamic model was set at 0.00005 and were trained for 11 epochs.

As seen in figures **??** and **??** the four models follow the a similar structure having a Convolutional layer that is then followed by a fully connected layer. The OD models differentiate at the final activation function where it was found that the Softmax activation function was not activating properly. Seeing this another activation function was used, the LogSoftmax function. It is also worthwhile to note that the final layer fully connected layer is different since the UCI dataset and Our dataset have a different number of total features. These models accept a tensor of the shape: (batchsize, 1, number_of_features). The output of item in this batch is a number that corresponds to the encoded label. Dropout is present at the fully connected layer, set to 50% to combat overfitting. Dropout is useful since it applies a 50% chance of ignoring a neurons outputting, which forces the model to learn redundancy.

VI. Evaluation

A. *Evaluation Metrics*

In order to determine which classifier provided the most accurate predictions, four different performance metrics were recorded and acted as a score. These metrics are **accuracy**, **f-beta**, **precision** and **recall**.

Accuracy is the ratio of the true labels y on the set of predicted labels y', where $accuracy(y, y') = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(y'_i = y_i)$.

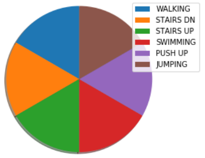
Precision is the ratio of true positives during prediction, and it is calculated as $P = \frac{T_P}{T_P + F_P}$, where T_P is the number of true positives and F_P is the number of false positives.

Recall is the ratio of correctly identified positive labels, and it is calculated as $R = \frac{T_P}{T_P + F_n}$, where T_P is the number of true positives and F_n is the number of false negatives.

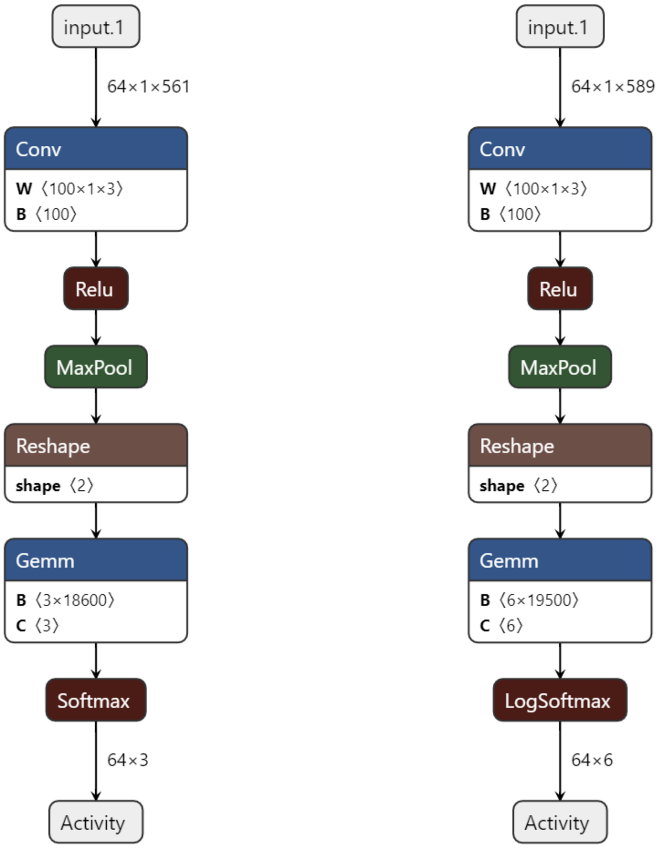
Beta (or **F-Beta**) is calculated based on the precision and recall scores of the classifier, wherein precision is multiplied by some parameter γ, thereby giving more importance to the precision value. This is calculated as $F_\gamma = (1+\gamma^2) * \frac{P * R}{(\gamma^2 * P) + R}$. For evaluation purposes, a γ value of 0.5 was used.

B. *Evaluating SVM*

Figure 1 above is the confusion matrix representing the accuracy of the classification of each activity as a percentage. The leading diagonal represents correctly labelled activities. Each activity group classified had an accuracy of 96% or higher, with half activities being classified at an accuracy of 99-100%. 1% of driving activities were misclassified as sitting. This was to be expected as during driving sessions there were times (i.e. being stuck in traffic) where the driver was stationary and in the same position as that of someone seated down. 2% of laying activities were misclassified as sitting, this is understandable as the two stationary activities have similar positions.



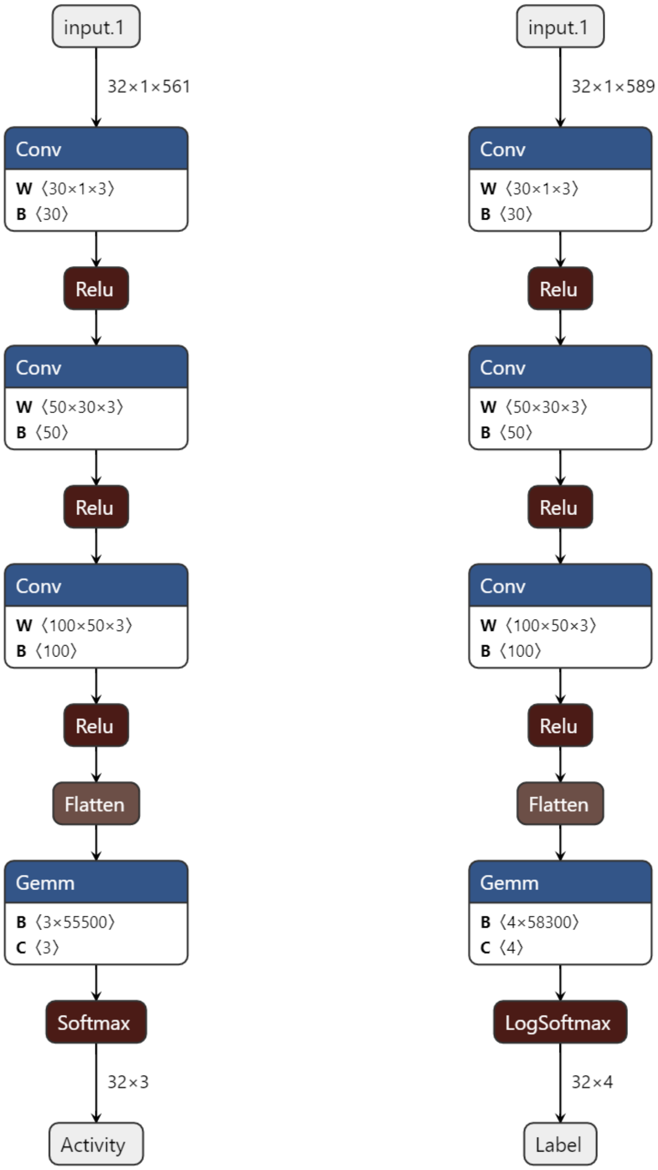
OD Dynamic Training Balanced Label Distribution



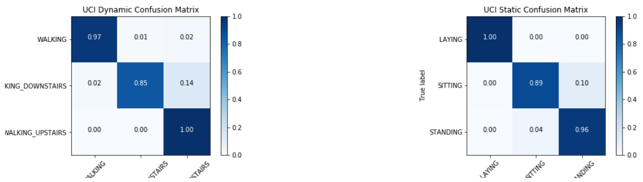
UCI(left) and OD(right) Dynamic Model Structure

C. Evaluating The CNNs

The CNN models were evaluated during training as well as after testing. During training, the training and validation loss was printed. When a hyperparameter was being amended during the development the shift in the loss output would indicate the success or failure of the amendment. In general, when the validation loss is more than the training loss, the model is considered to be overfitting, when the validation loss is less than the training loss, the model is considered to be underfitting. It was our aim to set the validation and training losses as equal to each other as possible. The hyperparameters set, as described in the methodology are a result of this process. After the testing steps where completed, the same evaluation metrics used for the classical ML techniques where implemented.



UCI(left) and OD(right) Static Model Structure



Dynamic(left) and Static(right) UCI Model Confusion Matrix

1) UCI Dynamic & Static: These models achieved very high results (¿90%) however a little overfitting was observed. To combat this overfitting, the epoch count was reduced however this started affecting general accuracy and so it was decided to keep epochs at 10 and accept a little overfitting. It was also noted that although results where mostly consistent, each new run would change the overall accuracy score by 2% on average. The Static model achieved the highest results from the CNN models. Due to their high accuracy, the f-score, precision and recall metrics resulted in similarly high results and offered little new information.

Dynamic(left) and Static(right) OD Model Confusion Matrix

2) Our Dataset Dynamic & Static: Initially the architecture for the UCI Dataset models would not function on our dataset. It was found that the activation function of the last fully connected layer had to be changed from Softmax to LogSoftmax. This affected the performance of the models as it increased the processing load. The Dynamic model achieved good results with an accuracy of ¿60%, when considering it trained on a reduced sized dataset with an equal distribution of 15% for each label. The extra effort put in to balance out the training distribution was expected to benefit the model in terms of accuracy, however there was no difference between the results of the model before the data balancing and after. The higher precision score of 71% compared to the recall of 61% shows that the model is still able to predict the relevant label. The Static model achieved very high results (¿90%). The same steps taken for the UCI dynamic & static models where taken and decided upon.

VII. Conclusion and Future Work

The purpose of this report is to discuss the problem, view the existing solutions and to present the work that has been done to date mainly the automatic user profiling prototype. Further development regarding both the improvement of such a prototype and the development of the automatic itinerary generation will take place in the upcoming weeks.

VIII. Distribution of Work

- Jake Sant
- Aiden Williams
- Ethan Zammit