

Win Barua

Registration number 100277378

2022

Sign Language Recognition using Leap Motion and Machine Learning

Supervised by Jaejoon Lee



University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

This document details the process of designing, developing and testing a sign language interpretation system using the Leap Motion Controller and Machine Learning methods, as well as showing off the outcome and results of the overall application, discussing what went well, what I could have improved on and justifying my reasons for using certain methods instead of others. I have tested the system using different methods of constructing my machine learning methods, mainly looking at alternative ways of using preprocessing to prepare the data before training the model, however testing showed that more preprocessing does not necessarily mean a better performance as it may skew the data. In terms of the Leap Motion Controller, which I am using for data collection, it can somewhat accurately detect hands and fingers and track their movements but it is more specialised towards virtual reality use cases. Because of this I had some problems with it when developing the system, especially relating to lighting conditions, hand tracking consistency and loss of accuracy when parts of the hand came together. This is why, although my findings show that the device has potential, it would definitely need further updates and development to be fully suitable for commercial or public use as a sign language interpretation system.

Contents

1. Glossary	5
2. Introduction	6
2.1. Objective and context	6
2.2. Related work	7
3. Design and methodology	8
3.1. Project structure	8
3.2. Class prediction	11
3.3. Development of machine learning model	12
3.3.1. Feature extraction	12
3.3.2. Data collection	13
3.3.3. Model 1: No feature scaling	14
3.3.4. Model 2: Normalisation (feature scaling)	17
3.3.5. Model 3: Standardisation (feature scaling)	18
3.3.6. Model 4: Data Sampling	18
4. Outcome and analysis	21
4.1. Using the application	21
4.2. Code showcase	23
4.3. Results of machine learning model	29
4.4. User documentation	31
4.4.1. Minimum hardware and software requirements	31
4.4.2. How to install and run application	32
4.4.3. How to use the system	33
4.4.4. Possible error messages	33
5. Discussion and Evaluation	34
5.1. Testing	34
5.1.1. Normalised model	35
5.1.2. Standardised model	36
5.1.3. Sampling model	37
5.2. Justifying the choice of model	38

5.3. Limitations	39
6. Conclusion	41
References	42
A. Code for the testing script	43
B. British sign language numbers 0-20	46
C. Leap Motion coordinate system	47
D. Palm normal visual	47

1. Glossary

Activation function In neural networks, decides if a neuron should be activated.

Class/label The thing to be predicted, the output of the machine learning model.

Data balancing Stripping the dataset so all classes have the same amount of data.

Features The attributes we are using to make a valid prediction.

Feature scaling Bringing down all features of dataset to a fixed range.

Feature extraction Deciding the data that is relevant from the available dataset.

Layer Building block of model, receives input, processes it and outputs to next layer.

Neural network Algorithms used to recognise patterns in dataset by mimicking the human brain.

Node Components of a layer, containing input, process function and output.

Normalisation Scaling technique where data is shifted to range between 0 and 1.

One-hot-encoding Way of preprocessing classes, creates binary feature for each class.

Overfitting When the model focusses too much on the exact numbers instead of the patterns, making it ineffective with new sets of data.

Relu function The most common activation function in machine learning, linear function that outputs input directly if positive, otherwise outputs zero.

Sigmoid function Type of activation function, ensures the output of its unit is between 0 and 1.

Softmax function Used on output layer, the main function used when doing multi-class classification.

Standardisation Scales data around a mean value with a unit standard deviation.

2. Introduction

2.1. Objective and context

This paper will present my research, development and testing processes as well as the results given from a system designed to interpret British Sign Language (BSL) when the user performs them to the Leap Motion Controller (LMC). This device is a small and portable USB peripheral that can be connected to a computer in order to collect hand tracking data using the provided API which comes with support for many programming languages, I have decided to use Python as it provided easy implementation with my machine learning model, although I was forced to use Python 2.7 to connect to the LMC API as it's the latest version supported and Python 3.90 for the machine learning model as it provided me with the most updated version of TensorFlow and Keras, the machine learning libraries I am using.

This project aims to perform the required research and development of a basic application that could potentially help hearing impaired people to communicate more easily or be used as a means to train or test people's knowledge of hand signs, I have constructed and tested the system only for the numbers 0 to 10 of BSL but it can easily be further developed to teach it to recognise more signs using the material and findings from this work, also they would be able to modify it to teach it other dialects altogether by collecting the appropriate sets of data. The main reason I have chosen to use only numbers for now is because it provided me with an easy and linear way to develop and test the system while doing the research on what kind of model would need to be constructed to optimise its accuracy, without having too many signs to work with and collect data for.

This project could have some potential especially in the teaching industry as it could be used to help teach young children who are born deaf, which would have a big impact since 90% of deaf children are born to hearing parents who are not experienced in sign language (NDCS, 2016). In these cases it is important for the parents to have the right kind of tools to prepare themselves when helping their children which should be done at an early age, so they can learn to perform signs as accurately and freely as possible. Preparing young deaf children has become a slight concern as NDCS (2016) states that there is a continuous decrease in qualified Teachers of the Deaf in England, reporting their decrease in 4% and increase of 18% in the number of deaf children in 2016.

The use of the system for easier communication is a less obvious one, since it is not catered to it by nature, the LMC needs to be wired to a computer or laptop and the process of performing the sign to the device to it translating the sign can take some time, about 10-15 seconds, since the sign needs to be held for about 10 seconds to collect enough data for the machine learning model. This lack of portability and speed would make the system unsuitable for use as a proper communication device, as opposed to using it for teaching and testing purposes as discussed previously.

2.2. Related work

Some related work has been done using similar methods, many have used other devices and hardware to collect hand movement data and process them, some have also used the LMC alongside computer vision or machine learning. They have varying advantages and drawbacks which will be detailed in this section.

One use of other hardware was a project (Ambar, 2018) to use **flex sensors** to collect gesture data in three dimensions, the flex sensors are designed to have a varying resistance depending on how they are bent, using this feature, the user's hand positions could be collected and processed to translate and output the sign performed. The main issues that arise with this system is having to use specialised gloves with lots of hardware and circuitry which might not be very elegant to the end-user, making them less likely to buy into the item. On top of this, the system can be costly to make which might hurt the availability and accessibility of the product as many people might not be able to afford it. This issue is made worse by the fact that testing has shown that the sensors can easily break after a short term usage. On the other hand this system was extremely accurate with a reported accuracy of 91.91% which makes this method much more accurate and consistent than software solutions that use machine learning or computer vision.

A different approach was used in this project (Kshitij Bantupalli, 2018) which uses **computer vision** to recognise American Sign Language, here the authors are using an iPhone 6 camera to capture a video and process each frame, extracting the features to train a Convolutional Neural Network. They are also making use of transfer learning in order to reduce the training data size without affecting the performance by utilising previous training. This paper reported an incredible accuracy of 99% on the training set, which means, hypothetically it would rarely make a mistake when trying to recognise a sign, however this accuracy is overshadowed by some well-known issues of computer

visions systems. One being the fact that the quality of the image depends on background colours and object which could mislead the vision system, this would greatly reduce the accuracy of the system unless the user's environment is restricted to certain lighting conditions and backgrounds which could make it more frustrating to use on a regular basis. This is further evidenced in the paper as they had issues with the accuracy dropping when the singer's face was included in the video as the model would extract the wrong features from the video. Some other problem the authors detailed was that the model struggled with skin tones it had not been trained with as well as instances where the user had differences in clothing.

Finally, one system that resembles mine closely is this project, Caroline Guardino (2014), which attempts to use the LMC to recognise American Sign Language using the k-nearest neighbour and support vector machine algorithms, mapping the 26 letters of the alphabet using some original and derived features from the sensor data of the LMC. Having used a similar approach to my own project to collect data from the device they have add a similar level of success, getting about 72% to 79% accuracy. Furthermore, this project was focussed on making a more portable and easy to use system for sign language recognition, citing other solutions such as flex sensor gloves to be somewhat cumbersome, also making sure to make the system much more affordable and therefore accessible to more people.

My aim was to produce a system that can bring the LMC and machine learning modules together to work smoothly and try to get an accuracy of at least 75% in order to show the initial potential of the system which could then be improved in future work.

3. Design and methodology

3.1. Project structure

The project has two main parts: a data collection module that uses the Leap Motion API to connect to the controller so it can grab the requested features from the device, and a machine learning module which reads the data extracted and makes a prediction on what sign the user has performed. These two modules are connected together in one Python script which will extract the data, save it in a file and wait for another user input before calling the machine learning model to predict on the given data. There is also a research and testing aspect to the project which will be detailed in later sections.

Although this project is not an object oriented one I have made a module diagram (similar to UML) which shows the modules and the methods they contain as well as the parameters they take and what they return, this gives an overview of how the project is structured and what each module does.



Figure 1: Module Diagram

Within the `PredictClass.py` module I am loading the the raw data saved by the extracting module and pre processing it appropriately, mainly extracting the necessary features and balancing the data. The processed data is then fed through to the `predict()` method which will load in the saved machine learning model and output a prediction. Here I am using the final version of my model after developing and testing multiple ones(see section 3.3), choosing the one that gave the best performance when testing with real world data.

The `DataExtractor.py` module was used initially for the data collection phase for training the model, but I designed it to be easily switchable from data collection to prediction mode, if the application went into production it would have the latter version of the script. When in data collection mode it is simply connecting to the LMC, asking the user for input before runs the `startTracking()` method which will connect to the LMC and run the `onFrame()` function which is performs the task of collecting all the data from the device for every frame and appending it to a list. The program will track the hands until another input is detected after which it will save the list as a `.csv` file. The only difference with the two modes is that when in prediction mode it will run the prediction module after saving the file and output the prediction to the user.

I have also drawn out a use case diagram which shows the flow of the application as well as what actor would be involved, a signer would start the gesture tracking, perform the sign and stop tracking to reveal the translated output that a non-signer could read

if the system was to be used as a communication tool, however, the non-signer actor could be optional as the signer could just be practising or learning how to perform signs in which case they would be the only actor. The «include» statement shows when a previous action is needed as a prerequisite before performing a certain action, for example activating the gesture tracking is needed before being able to perform hand gestures. The «extend» keyword is used where there is an alternate path, here it is used when the user reads the instructions they can either click on continue and activate gesture tracking or quit the application.

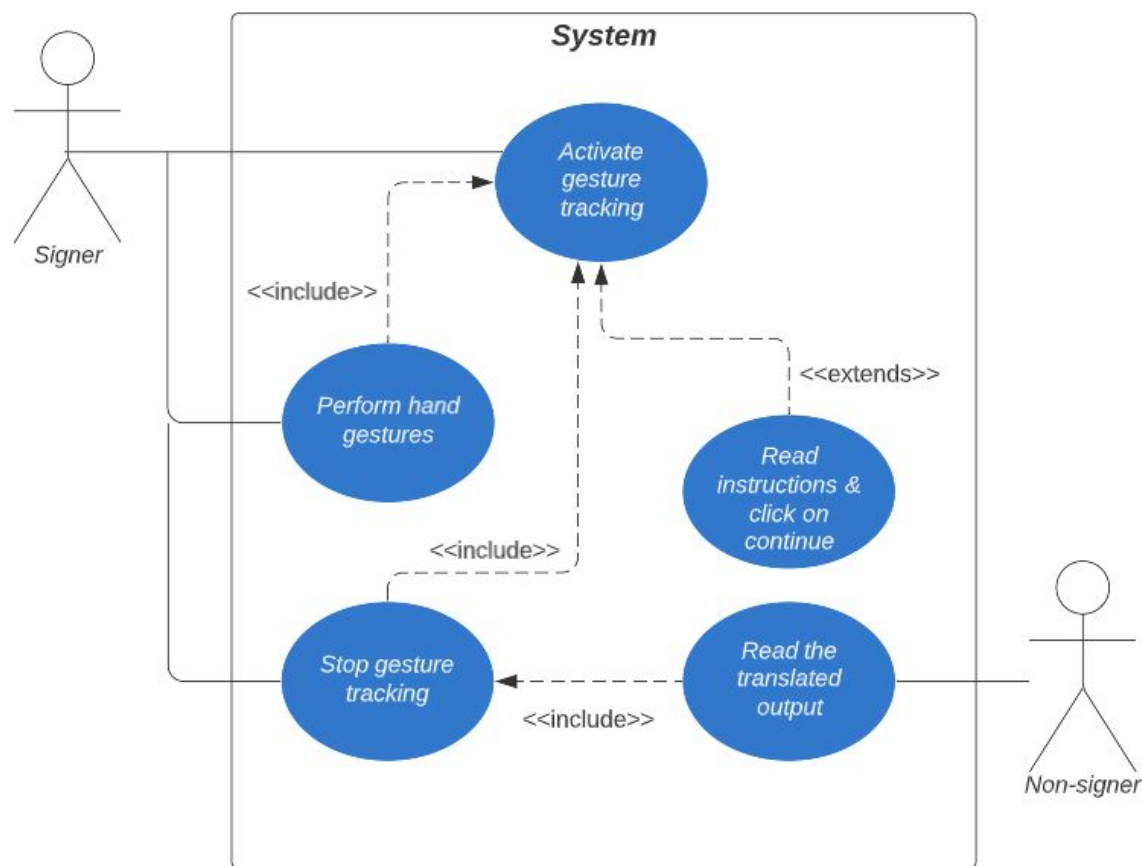


Figure 2: Use case diagram

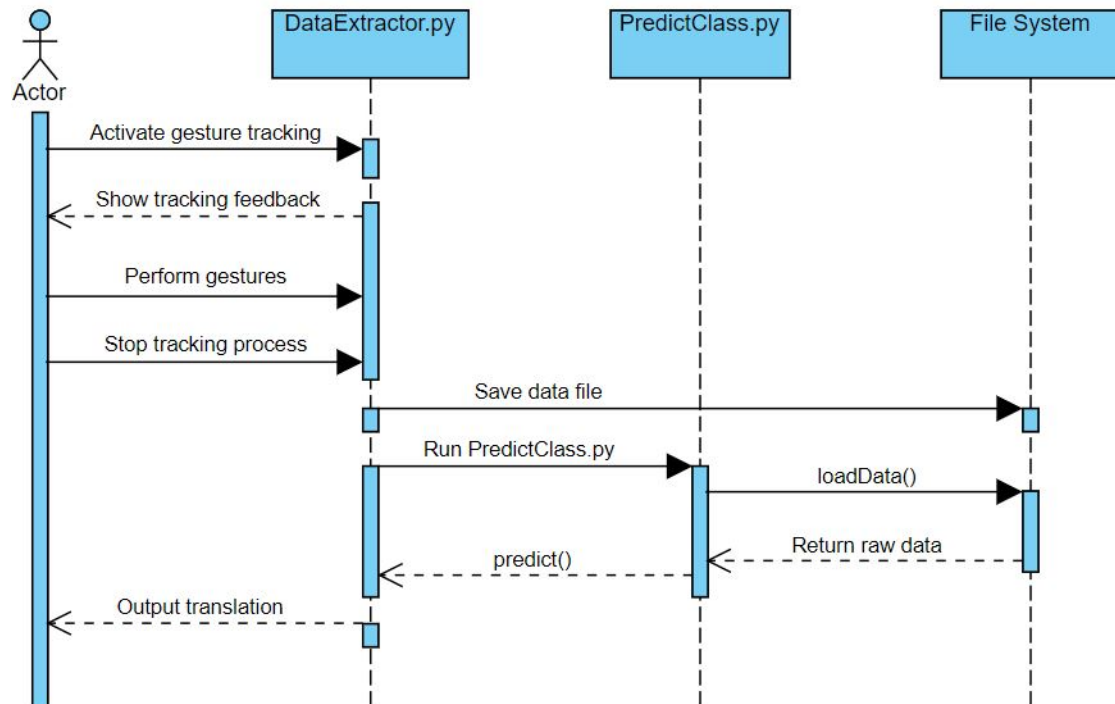


Figure 3: Sequence Diagram

3.2. Class prediction

As described in previous sections, predicting the user's sign was done with two modules, one which would collect the data for the gesture and store it in a file, then call the prediction script which will process the data according to the chosen model's requirements (see section 3.3.3), load the specified model and run it on the given data. I am using the `os` package which allows me to use `os.system()` to execute a shell command, I have to do it this way since I am using a different python version for the data collection module and prediction module, also using a virtual environment to run my prediction, the full call line is: `os.system('C:/Users/winba/anaconda3/envs/python390/python.exe predict_class.py')`.

The next section will go in detail about all the development processes of coming up with a suitable machine learning model, going through all the stages and the different models I have developed and tested before deciding which one I would be using in the final version of the system.

3.3. Development of machine learning model

3.3.1. Feature extraction

Most machine learning projects will have to go through a feature extraction stage where the developer will carefully decide what characteristics of the data collection system they want to exploit, there is no specific guideline as to how many features is suitable however, more features does not necessarily mean a better performance because adding features that are not relevant could dilute the data and skew the model's training process.

For this project I made sure to extract only features that would impact the way the model sees the gesture, including certain features relating to the hand as a whole and other features relating to each finger, a lot of these features are vector features which are divided in x, y and z. The following table 1 shows all the overarching features I have chosen for my system, every hand feature has a left and right hand sub-feature and every finger ones have a sub-feature for each finger (as well as x, y, z if it is a vector). In total the system contains 140 singular features and no derived features needed to be calculated as opposed to some other systems such as Caroline Guardino (2014).

Table 1: Extracted Features

Hand		Fingers	
Name	Type	Name	Type
no. of hands	integer	direction	vector
no. of fingers	integer	tip position	vector
palm position	vector	is extended	boolean
normal	vector	tip velocity	vector
direction	vector		
sphere center	vector		
sphere radius	double		
wrist position	vector		
palm velocity	vector		

These features have been selected because I believed they would provide the model with the best data to differentiate between the signs, they are defined as follows - in documentation (Ultraleap, 2014):

no. of hands how many hands are currently being detected, can be 0, 1 or 2.

no. of fingers how many fingers are currently being detected, can be 0, 5 or 10.

palm position a vector of the center position of the palms in the LMC coordinate system (see C).

normal a directional vector pointing out from your palm. (see D)

palm direction the direction of the palm through the fingers.

sphere center the center of a sphere to fit the curvature of the hand, as if the hand was holding a ball.

sphere radius the radius of the same sphere.

wrist position position of the wrist.

palm velocity rate of change of palm position in millimeters per seconds.

finger direction direction in which the finger is pointing.

tip position position of fingertip in millimeters from Leap Motion origin.

is extended boolean of whether or not the finger is straightened.

tip velocity rate of change of the fingertip position in millimeters per second.

3.3.2. Data collection

The data collection phase was a crucial part of developing the system as it would greatly dictate the quality and accuracy of the overall system, I needed to collect a reasonable amount of data to feed the machine learning model and find a good balance, as too little data would render the model ineffective and too much data would make it difficult for my computer to handle the load. I also needed multiple iteration/variations of data for the same sign to account for slight differences in hand positioning when performing the gestures.

To preface this, note that I have tested the models with the testing data used for training as well as collecting new data that was unseen by the model, normally the model would need to have a 100% or near 100% with the testing data because it has

already seen it, if it is lower then the model clearly has something wrong with it and is unusable, otherwise I should be able to move forward and test the model with new data.

I devised a first plan that would have me perform each sign twice for 30 seconds each, giving me a total of 11 files which I combined into one file to feed to the model. This phase of data collection provided a lot of data for each trial but did not provide enough variation and did not allow the model to account for slight changes each time the sign is performed because each sign is only performed twice. Because of this, the model did not perform well at all when confronted with unseen data,

The second set of data I collected was a lot more exhaustive as I collected data for each sign 5 times, holding the sign for 30 seconds at a time, this gave the data a lot more diversity while still being relevant to the appropriate sign, however I stayed as strict as possible with performing the sign, trying to not move during the 30 seconds and making sure they were accurate. Following the data collection, this gave me a total of 55 files which I just combined together to prepare the data to be processed for training the model.

3.3.3. Model 1: No feature scaling

This version of the model is the one with the least amount of processing, and **all models have data balancing** which was the first step in every model and is where I am stripping each dataset per class to have the same amount of data to avoid having the model be biased towards one or two specific classes. The purpose of this model is to see how performant it could be without any complex types of processing as data balancing is fairly simple and only has a light load on the system, and some types of data naturally do not need much processing to be viable for a machine learning model which is what I am testing here.

After the data balancing I am simply using one-hot-encoding to put the class labels in a format the model will understand. The next step is to split the data into training and testing data, then construct the model. The model itself is made up of 5 Dense layers, with the first layer having an input dimension equal to the number of features (140) and the last layer having 11 nodes which is the number of output classes, all the layers use a `relu` activation function except the last one which uses `softmax`, according to Baheti (2022) this is the most commonly used function for multiclass classification and provides the best results in most of these cases.

Listing 1: Model Structure

```
INPUT_DIM = 140 # number of features
OUTPUT_DIM = 11 # number of classes

model = Sequential()
model.add(Dense(8, input_dim = INPUT_DIM , activation =
    'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(OUTPUT_DIM, activation = 'softmax'))
```

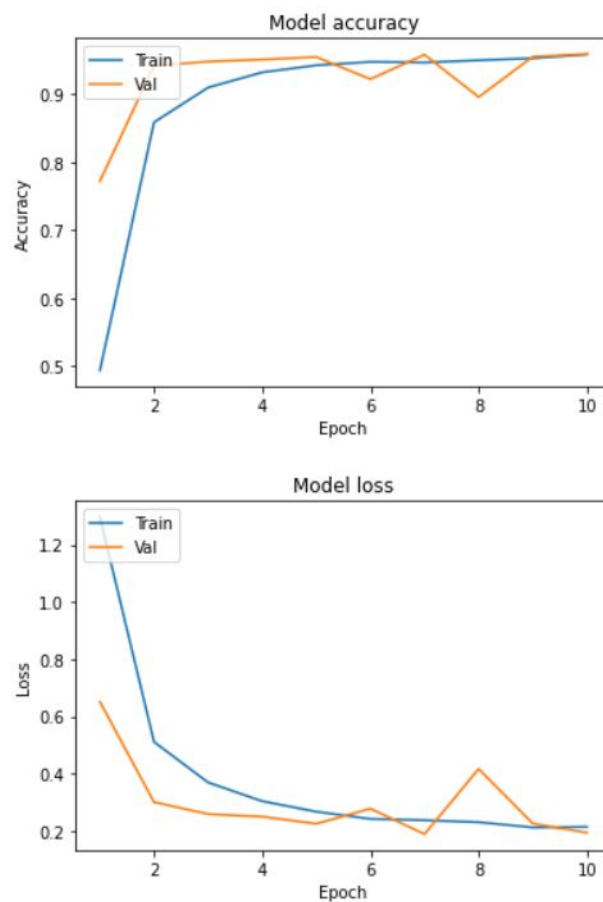


Figure 4: Model accuracy and loss graphs

(<Figure size 504x504 with 1 Axes>,
 <AxesSubplot:xlabel='predicted label', ylabel='true label'>)

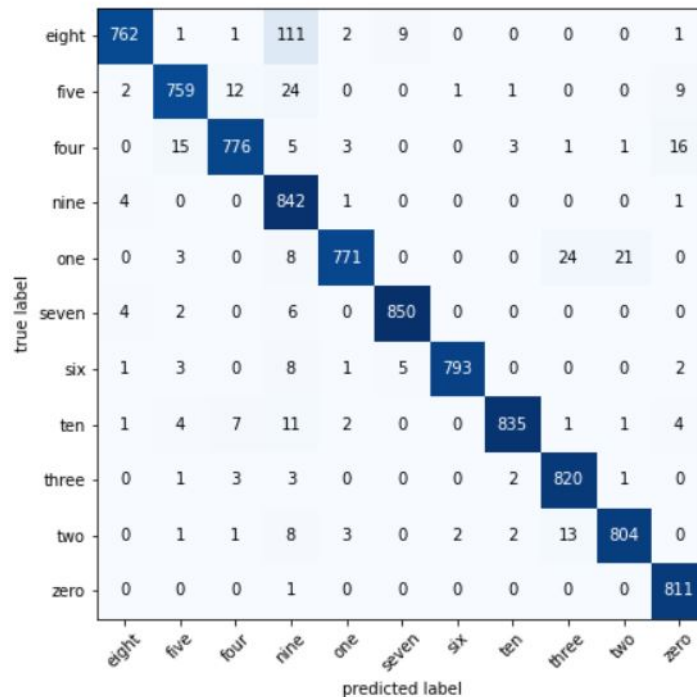


Figure 5: Confusion matrix

Figure 4 is showing the levels of accuracy and loss for training and value that the system has experienced throughout the training process, whenever the value line goes over the train line it means there is some overfitting occurring which is where the algorithm learns from exact data and its noise instead of learning from patterns, too much overfitting can cause the model to not perform well when being fed new data. Apart from the slight instances of overfitting the graphs do not look bad as the accuracy and loss are smoothly going up and down respectively.

The confusion matrix in figure 5 shows the predicted labels against the expected labels for each prediction, it can provide a very good overview of how the model performs with each class and we can see which classes it especially struggles with. Here we can see that 111 times the model predicted a “nine” instead of “eight”, from this we can understand that the model struggles with differentiating and eight sign from a nine sign. Although 111 is higher than some of the other wrongly predicted instances, it is much less than the amount of times the model made the correct prediction.

3.3.4. Model 2: Normalisation (feature scaling)

This model relied on a type of feature scaling, also known as min-max scaling where all the data is scaled to range between 0 and 1. Feature scaling makes the training faster and optimises it to reduce noise and variance, it can definitely be very useful for certain machine learning projects but some types of model will be completely indifferent to it or perform even worse.

As usual, I have balanced the data first before moving to the normalisation step which involves using the `MinMaxScaler()` function given by the `sklearn.preprocessing` package, as shown in listing . This is followed by encoding the data and constructing and fitting the machine learning model which has the same layers as the previous one. This version of the model gives a very similar accuracy of 95% during training however it did not do well at all when actually making predictions even on test data and turned out to be completely unusable.

Listing 2: Model Normalisation

```
# normalise data
norm_scaler = MinMaxScaler()
norm_data = norm_scaler.fit_transform(X)
# re-construct dataframe with the label
norm_data = pd.DataFrame(data = norm_data)
norm_data['label'] = y.values
norm_data.columns = columns
```

As stated before, using already seen data to make the model predict the label is a useful indication of whether or not the model is worth considering, however for the sake of completeness I have created a script which would run tests for me, going through every hand data file in a folder and run the model on them, I have used this script for every model using test data as well as new data. For this model the test data accuracy was 9.09% and it was the exact same when using new data, telling me there is something wrong in the way the model sees the data, I will talk more about what the reasons might be in section 4 where I will be discussing the strengths and weaknesses of each model as well as justifying my decision for which model I used in the final version of the system.

3.3.5. Model 3: Standardisation (feature scaling)

This is another type of feature scaling where the data is scaled around a mean value, usually 0 with a unit standard deviation of 1 without a lower or upper limit, this method is useful when the data we have follows a Gaussian distribution, which is where all the data is distributed equally above and below a mean value, following a bell-shaped curve if plotted, however in some cases it can also be useful where the data does not follow Gaussian distribution. This was also implemented using `sklearn.preprocessing` with its `StandardScaler()` function.

Listing 3: Model Standardisation

```
# standardise data
scaler = StandardScaler()
X = scaler.fit_transform(X)
# re-construct dataframe with the label
scaled_data = pd.DataFrame(data = X)
scaled_data['label'] = y.values
scaled_data.columns = columns
```

Again, this model uses the same layers and structure as the previous models and gave very similar graphs and confusion matrix. In terms of results, it was very similar to the data normalisation model, yielding an accuracy of 18.18% when using test and new data which means the model is unusable.

3.3.6. Model 4: Data Sampling

Along with the no scaling (see section 3.3.3) method, this was the other method that I would consider as usable looking at the results, this pre-processing method is fundamentally different to the other ones because it requires a different construct for the machine learning method. Every step is the same except instead of sending data line by line, I am sampling the data to send a block of data at a time (20 lines) where I am assigning the most common label in that sample to the block of data. The aim of using this method is to combine and correlate some of the data together, having the model look at multiple lines at a time to train on those patterns instead of having it look at the information separately line by line.

Listing 4: Sampling pseudocode

```
function get_frames(dataframe, frame_size, increment){
    frames = []
    labels = []

    for (int i = 0; i <= dataframe.length - frame_size;
        increment) {
        # sample all 140 features and store in variables
        hands = dataframe['hands'].values[i: i + frame_size]
        fingers = dataframe['fingers'].values[i: i + frame_size]
        ...
        rh_wrist_pos_y = dataframe['rh_wrist_pos_y'].values[i:
            i + frame_size]
        rh_wrist_pos_z = dataframe['rh_wrist_pos_z'].values[i:
            i + frame_size]

        # get most common label and append to frames and labels
        label = most common label in sample
        frames.append(hands, fingers, ..., rh_wrist_pos_y,
            rh_wrist_pos_z)
        labels.append(label)
    }

    return frames, labels
}
```

Listing 5: Sampling ML model

```
model = Sequential()
model.add(Conv2D(16, (2, 2), activation = 'relu',
    input_shape = X_train[0].shape))
model.add(Conv2D(32, (2, 2), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(11, activation='softmax'))
```

Listing 5 shows the structure I have used for the sampling model, it starts with a `Conv2D` layer because I am sending a 2 dimensional frame of data which has a label assigned to it, before using a `Flatten()` layer to convert it into a 1 dimensional array to pass through some `Dense()` layers, with the final layer having 11 nodes equivalent to the number of output classes. The structure of the model had to be changed in order to adhere to the dimensions of the sampled frames that I am sending through which would have been incompatible if I was just using `Dense()` layers.

When testing the model I received an accuracy of 98.18% when using the test data and an accuracy of 70.91% when using new data, this shows the model has potential and would be somewhat useable however the first model I tested using no feature scaling or extra pre-processing still performed the best when using new data, with an accuracy of 75.45%. This is why I have chosen to use the no scaling model as my final model looking purely at its accuracy on real world data, in the next section I will be showing evidence for the results and outcome of this model as well as analysing why it worked as opposed to all the other models.

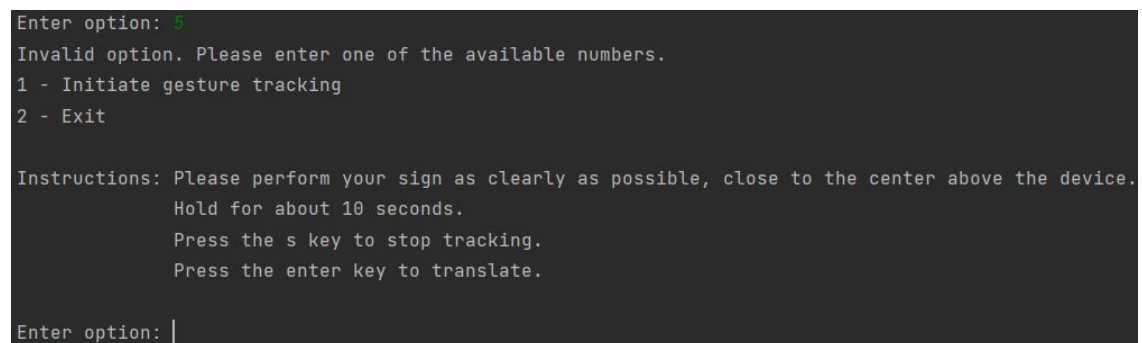
All of these models were mentioned in the **project notebook**, along with the associated graphs, confusion matrices and testing evidence (which can also be found in the appendix) to show my progress when researching and developing the final version of my model.

4. Outcome and analysis

In this section I will focus on the main, final version of my system which uses the no feature scaling model, evidencing its results to the end-user when it is finished running. The exact flow of the application goes as follows:

1. User starts the application.
2. App displays options (1 - Start tracking; 2 - Exit) and instructions.
3. IF option 1:
 - 3.1. Starts tracking, allows user to perform sign.
 - 3.2. User does sign and presses on the `s` key.
 - 3.3. Tracking stops and displays message for next step.
 - 3.4. User presses the `Enter` key.
 - 3.5. Translation is shown and application loops back to step 2.
4. IF option 2: application exits.
5. ELSE: feedback to the user saying “invalid value”.

4.1. Using the application



```
Enter option: 5
Invalid option. Please enter one of the available numbers.
1 - Initiate gesture tracking
2 - Exit

Instructions: Please perform your sign as clearly as possible, close to the center above the device.
              Hold for about 10 seconds.
              Press the s key to stop tracking.
              Press the enter key to translate.

Enter option: |
```

Figure 6: Invalid input

```
C:\Python27\python27.exe C:/Users/winba/Desktop/TYP_ALT/scripts/DataExtractor.py
1 - Initiate gesture tracking
2 - Exit

Instructions: Please perform your sign as clearly as possible, close to the center above the device.
             Hold for about 10 seconds.
             Press the s key to stop tracking.
             Press the enter key to translate.

Enter option: |
```

Figure 7: Ask for user input

```
Enter option: 1
Connected
1.80572283268
1.82755064964
1.85151302814
1.90119540691
2.00579714775
2.03238201141
2.05133748055
2.04911231995
```

Figure 8: Start tracking

```
12.7952642441
12.8265705109
12.8520059586
sTracking stopped... press ENTER to show translation
two
```

Figure 9: End tracking, show translation

This shows how the application runs and asks for user input before starting to track and ending it, also when tracking I am showing the x position of the palm, mainly because this allows to overcome an issue with the LMC where it randomly stops detecting the hand, this might seem like an unpolished way of giving user feedback but unfortunately since this is a hardware/software issue from Leap Motion themselves I cannot do much about it. However this way the user knows that the device is actually detecting the hand as it will just display 0.0 if no hand is being detected, usually slightly moving the hand allows the device to re-detect the hand but if it doesn't work then they would have to just restart, see the limitations section(5.3) for more details.

4.2. Code showcase

Apart from the Leap Motion API class which is required to connect to the device, the system itself is classless and does not make use of object-oriented programming, instead opting for modules for the two major tasks which use different versions of Python due to the compatibility requirements of the API and the machine learning framework. In order to be able to import the Leap API I had to add a lib folder with certain files provided by them, then importing the required things using `import lib.Leap as Leap`.

These are some of the code required to setup connect to the device as well as gather data from it in order to store it into a dataframe to be stored as a file:

```
# Setup leap
controller = Leap.Controller()
controller.set_policy(Leap.Controller.POLICY_ALLOW_PAUSE_RESUME)
```

Figure 10: Leap API setup

```
# set the listener onto leap motion and activate the hand detection
listener = LeapListener()
def start_tracking():
    global is_menu_active
    is_menu_active = False           # remove menu display
    controller.set_paused(False)     # start the tracking
    controller.add_listener(listener)
    sys.stdin.readline()
```

Figure 11: startTracking() function

```
def stop_tracking():
    global hand_data
    global is_data_collected
    is_data_collected = True        # set state as 'data_collected'
    controller.set_paused(True)      # stop the leap motion tracking
    controller.remove_listener(listener)
    hand_data = hand_data.loc[(hand_data != 0).any(1)] # add all data to main dataframe
    hand_data['label'] = 'ten'       # add label (only used for training model)
    hand_data.to_csv('data_to_be_read.csv', index=False) # store data file

    # wait for user input before starting the translation
    print('Tracking stopped... press ENTER to show translation')
    global is_menu_active
    is_menu_active = True
```

Figure 12: stopTracking() function


```
##### LISTENER CLASS #####
class LeapListener(Leap.Listener):
    # send message when connected to controller
    def on_connect(self, controller_arg):
        print('Connected')
    # runs every frame of tracking
    # send data from frame of data to data list
    def on_frame(self, controller_arg):

        frame = controller.frame()

        right_hand = frame.hands[0]
        left_hand = frame.hands[0]
```

Figure 13: Listener class

```
# detect which hand the user is using to use the correct data
if len(frame.hands) == 2:
    right_hand = frame.hands.rightmost
    left_hand = frame.hands.leftmost
elif len(frame.hands) == 1:
    if frame.hands[0].is_right:
        right_hand = frame.hands[0]
    else:
        left_hand = frame.hands[0]
```

Figure 14: Right-hand detection

```
# grab all the data into variables to be stored into list
rh_palm_pos = right_hand.palm_position
rh_palm_normal = right_hand.palm_normal
lh_palm_pos = left_hand.palm_position
lh_palm_normal = left_hand.palm_normal
lh_direction = left_hand.direction

rh_thumb = right_hand.fingers[0]
rh_index = right_hand.fingers[1]
rh_middle = right_hand.fingers[2]
rh_ring = right_hand.fingers[3]
rh_pinky = right_hand.fingers[4]
```

Figure 15: Retrieve Leap data

```
# Create row of data
data_row = {'hands' : len(frame.hands),
            'fingers' : len(frame.fingers),
            'lh_palm_pos_x' : lh_palm_pos.x,
            'lh_palm_pos_y' : lh_palm_pos.y,
            'lh_palm_pos_z' : lh_palm_pos.z,
            'rh_palm_pos_x' : rh_palm_pos.x,
            'rh_palm_pos_y' : rh_palm_pos.y,
            'rh_palm_pos_z' : rh_palm_pos.z,
```

Figure 16: Add to data row

These screenshots show some of the methods and logics the program goes through to gather the user's hand tracking data, all of this is ran from the `main()` function where the user option routing resides, simply calling the `startTracking()` method when the user chooses the first option. This method will add the listener to the leap motion which will automatically start the tracking and go into the `on_frame()` which takes

all the necessary data from the API to add it to a row of data which will then be added to a dataframe and stored onto the file system. As seen in figure 14, this algorithm is required because the API does not provide an easy function to get the right or left hand, but it stores all the hands in an array, the system will detect how many hands are in view, if there are two hands then it can just assign the correct hands to appropriate variables by looking at the leftmost and rightmost hand. If there is only one hand then it will detect whether or not it is a right hand and store it in the appropriate variable by looking at the 0th index which will be the only element in the hands array. This was an important step during the development of the system as this allows to store the data in the appropriate columns since most signs only require your dominant hand and not dealing with the other hand would cause errors.

```
# stop tracking when s key is pressed
keyboard.on_press_key("s", lambda _:stop_tracking())
while is_menu_active:
    # if data is in collected state call the prediction model then display menu
    if is_data_collected:
        os.system('C:/Users/winba/anaconda3/envs/python390/python.exe predict_class.py')
        time.sleep(1)
        is_data_collected = False
    display_menu()
```

Figure 17: Inside `main()` function, call the prediction model

The above figure shows the code I have used when calling the other module, in figure 12 there is a boolean variable called `is_data_collected` which checks on the state of the application flow, if the data has just been collected then this variable turns to `TRUE` and when the menu is reactivated it will first send a system call to run the `predict_class.py` script and show the translation before displaying the menu again.

The second line show the code I used to listen for user input, the program will run the `stop_tracking()` method when the `s` key is pressed which is responsible for storing the data file and removing the listener from the leap motion controller. I am using a specific package called `keyboard` which allows the python script to wait for a specific user input, and I am implementing it using an inline function call with a lambda expression.

The next few figures will show the code contained in the `predict_class.py` module:

```
# load and split data
data = pd.read_csv('data_to_be_read.csv')
dataset = data.values
X = dataset[:, 0:140].astype(float)

# load and process model
model = tf.keras.models.load_model('../model_training/saved_models/no_scaling/')
probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
```

Figure 18: Loading the data and model

```
# make predictions on data and find most common prediction
predictions = probability_model.predict(X)
prediction_indices = []
for item in predictions:
    prediction_indices.append(np.argmax(item))

# get the most common prediction
counter = 0
max_num = prediction_indices[0]
for item in prediction_indices:
    current_frequency = prediction_indices.count(item)
    if current_frequency > counter:
        counter = current_frequency
        max_num = item
```

Figure 19: Make predictions and get the most common prediction

```
# get label
labels = ['eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'ten', 'three', 'two', 'zero']
print(labels[max_num] + '\n')
```

Figure 20: Get the label for output

This code is all contained in the `main()` function as each task does not take that many lines of code so I believe it is unnecessary to overload the module with functions, the aim of this module is to prepare the data before it is inserted into the model which will give us a label in a one-hot-encoded format which is why we need to transfer it back to a normal string using a list of the labels in the correct order which was noted when developing the system by using `labelEncoder.classes_` which returns all the labels as they were encoded, because the label encoder does not return the encoded labels in the order they were first inputted.

As shown in figure 18, I am stripping the data to only take the 140 features and omit the label column since it would be irrelevant when making a prediction, this is followed by loading the model which has previously been saved onto the computer, this feature allows to develop, test and optimise a model on Jupyter Notebook and use it externally. Another thing I am doing is getting the most common prediction, because I am sending in the data line by line, each line of code will be looked at against what the model has already trained with and a prediction will be made on each row, so I need to find the label that is predicted the most amount of times in the whole process as it will most likely be the number that was signed throughout the 10 second duration.

4.3. Results of machine learning model

As stated in earlier sections I have created a script specifically to test the performance of a model depending on the data given to it, it will go through a list of data files where each sign is performed a certain amount of times and display the actual label and predicted label for each data file as well as calculate the accuracy per class and rank them so I can see what signs it specifically struggles with. In my case I have tested it with 10 of each sign (see figure 21), I performed each sign for 10 seconds to make it similar to real-use and also made sure the device was properly tracking my hand the entire time because I know the device to sometimes have trouble with consistent tracking for extended periods of time (more on this in section 5.3). The following figures shows parts of the data files folder used for testing and the output text file that is created, which shows the outcome of the main no feature scaling model used in the final system and these results will be reflected when using the full system since it is just importing the model and running it on the data, which is similar to what the testing script is doing.


















Name	Date modified	Type	Size
 0-0.csv	23/05/2022 01:13	Microsoft Excel Co...	630 KB
 0-1.csv	23/05/2022 01:13	Microsoft Excel Co...	656 KB
 0-2.csv	23/05/2022 01:13	Microsoft Excel Co...	977 KB
 0-3.csv	23/05/2022 01:14	Microsoft Excel Co...	1,168 KB
 0-4.csv	23/05/2022 01:14	Microsoft Excel Co...	1,067 KB
 0-5.csv	23/05/2022 01:15	Microsoft Excel Co...	1,002 KB
 0-6.csv	23/05/2022 01:15	Microsoft Excel Co...	1,120 KB
 0-7.csv	23/05/2022 01:16	Microsoft Excel Co...	1,057 KB
 0-8.csv	23/05/2022 01:17	Microsoft Excel Co...	806 KB
 0-9.csv	23/05/2022 01:18	Microsoft Excel Co...	772 KB
 1-0.csv	23/05/2022 01:19	Microsoft Excel Co...	807 KB
 1-1.csv	23/05/2022 01:20	Microsoft Excel Co...	806 KB
 1-2.csv	23/05/2022 01:21	Microsoft Excel Co...	885 KB
 1-3.csv	23/05/2022 01:21	Microsoft Excel Co...	863 KB
 1-4.csv	23/05/2022 01:22	Microsoft Excel Co...	874 KB
 1-5.csv	23/05/2022 01:22	Microsoft Excel Co...	1,010 KB
 1-6.csv	23/05/2022 01:27	Microsoft Excel Co...	790 KB

Figure 21: Part of data files folder

```
Overall model accuracy
=====
correct: 75.45 %
incorrect: 24.55 %

Predicted labels
=====
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: one - True class: one
Predicted class: three - True class: one
Predicted class: three - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: three - True class: one
Predicted class: one - True class: one
```

Figure 22: Part of overall accuracy section


```
Per class rankings
=====
Class: six - correct: 0.00 % - incorrect: 100.00 %
Class: seven - correct: 30.00 % - incorrect: 70.00 %
Class: one - correct: 70.00 % - incorrect: 30.00 %
Class: two - correct: 70.00 % - incorrect: 30.00 %
Class: ten - correct: 80.00 % - incorrect: 20.00 %
Class: eight - correct: 90.00 % - incorrect: 10.00 %
Class: nine - correct: 90.00 % - incorrect: 10.00 %
Class: zero - correct: 100.00 % - incorrect: 0.00 %
Class: three - correct: 100.00 % - incorrect: 0.00 %
Class: four - correct: 100.00 % - incorrect: 0.00 %
Class: five - correct: 100.00 % - incorrect: 0.00 %
```

Figure 23: Per class accuracy rankings

Figure 22 shows the overall accuracy of the model at 75.45%, this is using new data that was unseen by the model so it is the equivalent to real-life cases so it is a good indication of how good it will perform when being used by the end user, provided it is utilised properly. The next figure shows how the algorithm ranked the classes depending on how well the model performed when it encountered each sign, in order of worst performing to best performing, having the worst performing classes at the top allows me or anyone testing the system to be able to easily see the classes it is having most trouble with and assess what the issue is.

4.4. User documentation

4.4.1. Minimum hardware and software requirements

Hardware

- Leap Motion Controller device from Ultraleap (formerly Leap Motion, Inc.)
- Processor: minimum 1.50 GHz; Recommended: 2GHz or more
- Memory (RAM): Minimum 4GB; Recommended: 6GB or more
- Storage: 600MB
- I/O: USB 2.0 port

Software

- Windows 10 or newer, Mac OS X or Linux (Ubuntu)
- Leap Motion Orion 3.2.1, <https://developer.leapmotion.com/releases/?category=orion>
- Python 2.7
- Python 3.9
- Any TensorFlow and Keras compatible with Python 3.9

4.4.2. How to install and run application

Make sure your system is in line with the minimum requirements.

Install the Leap Motion SDK (as per their documentation)

1. Install the client software (contains drivers and service/daemon software).
 - On Windows: run “Leap_Motion_Installer_3.2.1.exe”
 - On Mac OS X:
 - a) Open “Leap_Motion_Installer_3.2.1.dmg”
 - b) Run “Leap Motion.pkg”
 - On Linux (Ubuntu):
 - For 32-bit systems, run: `sudo dpkg --install Leap-3.2.1-x86.deb`
 - For 64-bit systems, run: `sudo dpkg --install Leap-3.2.1-x64.deb`
2. Copy the LeapSDK folder to a suitable location on your computer

Run application

1. Place downloaded folder of project in suitable place on computer.
2. Open command line (open as administrator if you encounter any problems).
3. Navigate to project folder using `cd` command.
4. Run application using command: `python DataExtractor.py`.
5. Follow on-screen instructions.

4.4.3. How to use the system

Navigate through application to get to hand tracking stage, the main thing to focus on is to perform on of the implemented signs for 10 seconds, holding your hand above the center line of the leap motion controller, not directly above the controller but slightly closer to your body. Then press the `s` key which will save the data to the project folder, now press `Enter` in order to start the sign prediction process which will take about 5-10 seconds.

4.4.4. Possible error messages

The only error message you might receive is when choosing an option at the start of running the application, before the hand tracking starts; if you enter a number/option other than those available the program will give you an error message: “Invalid option. Please enter one of the available numbers.” The program will loop you back to the main menu screen where you can re-enter your option.

5. Discussion and Evaluation

5.1. Testing

Although this project's main goal is to create an application, it is also research heavy due to the machine learning aspect, which is why a lot of the model testing was covered during the development phase when trying to decide what model to use (see section 3.3). However that section of the report only showed graphs and confusion matrices of the different models that I have developed, this section will focus on the results of testing the other 3 models.





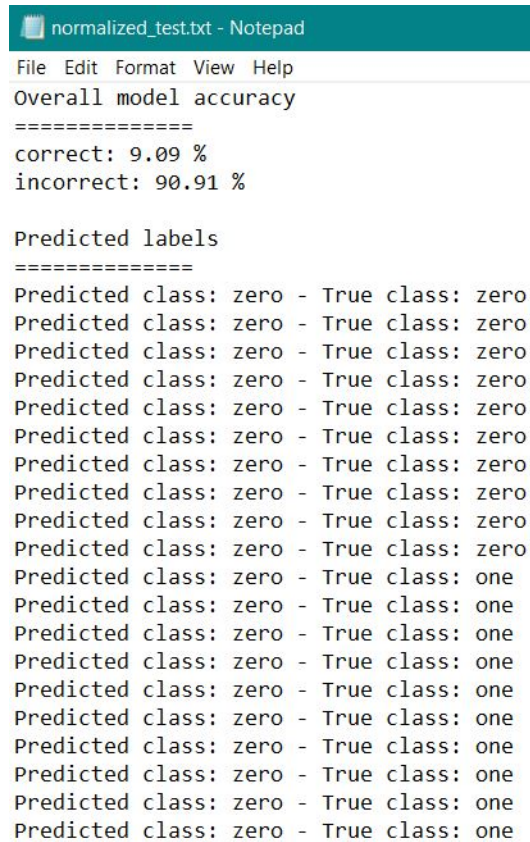
Name	Date modified	Type	Size
 no_scaling_test.txt	23/05/2022 14:22	Text Document	6 KB
 normalized_test.txt	23/05/2022 14:25	Text Document	6 KB
 sampling_test.txt	23/05/2022 14:55	Text Document	6 KB
 standardized_test.txt	23/05/2022 14:38	Text Document	6 KB

Figure 24: Model testing folder

The above figure shows the folder where all the test outputs are stored, each model is tested separately on the data files showed in figure 21 (note: figure 21 only shows part of the folder, there is a total of 110 files). As the no feature scaling model was the one chosen for the final system, it's testing output were already shown in the previous section which is why I will be showing the outputs for the other 3 models.

5.1.1. Normalised model



```
normalized_test.txt - Notepad
File Edit Format View Help
Overall model accuracy
=====
correct: 9.09 %
incorrect: 90.91 %

Predicted labels
=====
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: one
Predicted class: zero - True class: one
Predicted class: zero - True class: one
Predicted class: zero - True class: one
Predicted class: zero - True class: one
Predicted class: zero - True class: one
Predicted class: zero - True class: one
Predicted class: zero - True class: one
Predicted class: zero - True class: one
Predicted class: zero - True class: one
```

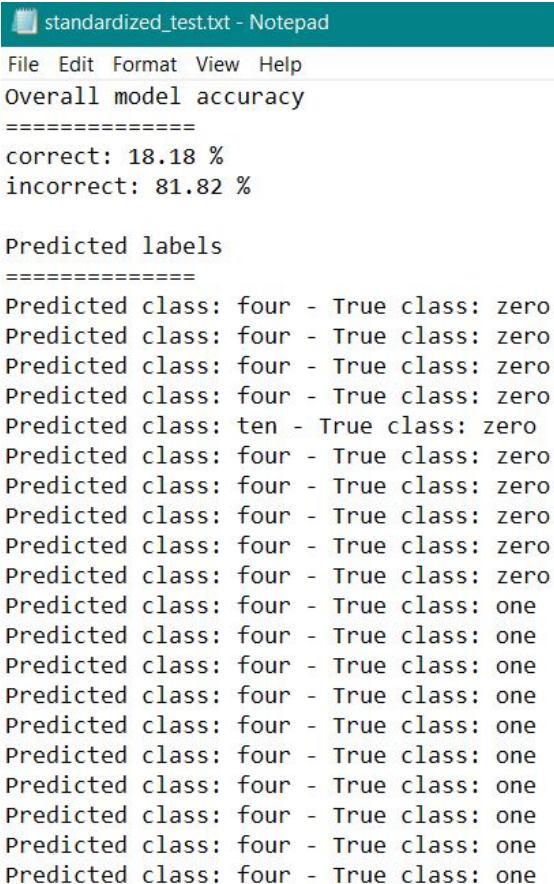
Figure 25: Normalised test output 1

```
Per class rankings
=====
Class: one - correct: 0.00 % - incorrect: 100.00 %
Class: two - correct: 0.00 % - incorrect: 100.00 %
Class: three - correct: 0.00 % - incorrect: 100.00 %
Class: four - correct: 0.00 % - incorrect: 100.00 %
Class: five - correct: 0.00 % - incorrect: 100.00 %
Class: six - correct: 0.00 % - incorrect: 100.00 %
Class: seven - correct: 0.00 % - incorrect: 100.00 %
Class: eight - correct: 0.00 % - incorrect: 100.00 %
Class: nine - correct: 0.00 % - incorrect: 100.00 %
Class: ten - correct: 0.00 % - incorrect: 100.00 %
Class: zero - correct: 100.00 % - incorrect: 0.00 %
```

Figure 26: Normalised test output 2

Figure 25 shows the overall accuracy of the model to be at 9.09% and also shows the top part of the predictions list, where there are 110 of which is the same as the number of files, the next figure shows the rankings of each class depending of how well the model was able to recognise them. This test clearly shows that the model is not effective or useable, and we can see it thinks everything is a zero .

5.1.2. Standardised model



```
standardized_test.txt - Notepad
File Edit Format View Help
Overall model accuracy
=====
correct: 18.18 %
incorrect: 81.82 %

Predicted labels
=====
Predicted class: four - True class: zero
Predicted class: four - True class: zero
Predicted class: four - True class: zero
Predicted class: four - True class: zero
Predicted class: ten - True class: zero
Predicted class: four - True class: zero
Predicted class: four - True class: zero
Predicted class: four - True class: zero
Predicted class: four - True class: zero
Predicted class: four - True class: zero
Predicted class: four - True class: one
Predicted class: four - True class: one
Predicted class: four - True class: one
Predicted class: four - True class: one
Predicted class: four - True class: one
Predicted class: four - True class: one
Predicted class: four - True class: one
Predicted class: four - True class: one
Predicted class: four - True class: one
Predicted class: four - True class: one
```

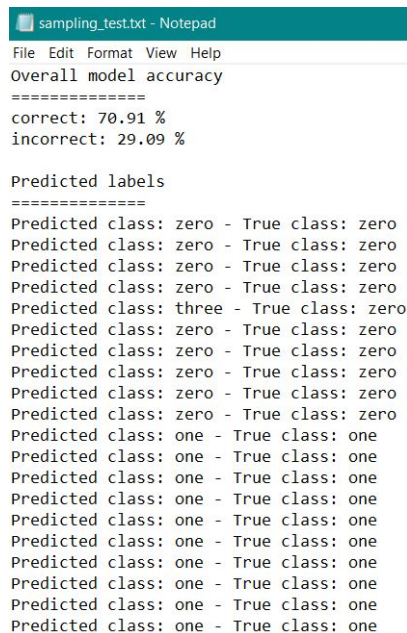
Figure 27: Normalised test output 2

```
Per class rankings
=====
Class: zero - correct: 0.00 % - incorrect: 100.00 %
Class: one - correct: 0.00 % - incorrect: 100.00 %
Class: two - correct: 0.00 % - incorrect: 100.00 %
Class: three - correct: 0.00 % - incorrect: 100.00 %
Class: five - correct: 0.00 % - incorrect: 100.00 %
Class: six - correct: 0.00 % - incorrect: 100.00 %
Class: seven - correct: 0.00 % - incorrect: 100.00 %
Class: eight - correct: 0.00 % - incorrect: 100.00 %
Class: nine - correct: 0.00 % - incorrect: 100.00 %
Class: four - correct: 100.00 % - incorrect: 0.00 %
Class: ten - correct: 100.00 % - incorrect: 0.00 %
```

Figure 28: Normalised test output 2

These figures show slightly better accuracy at 18.18% but still not useable at all and the latter figure gives an overview of accuracy per class and we can see that the model thinks everything is either a four or a ten.

5.1.3. Sampling model



```
sampling_test.txt - Notepad
File Edit Format View Help
Overall model accuracy
=====
correct: 70.91 %
incorrect: 29.09 %

Predicted labels
=====
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: three - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: zero - True class: zero
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
Predicted class: one - True class: one
```

Figure 29: Normalised test output 2

```
Per class rankings
=====
Class: nine - correct: 0.00 % - incorrect: 100.00 %
Class: seven - correct: 10.00 % - incorrect: 90.00 %
Class: five - correct: 40.00 % - incorrect: 60.00 %
Class: two - correct: 70.00 % - incorrect: 30.00 %
Class: six - correct: 80.00 % - incorrect: 20.00 %
Class: zero - correct: 90.00 % - incorrect: 10.00 %
Class: eight - correct: 90.00 % - incorrect: 10.00 %
Class: one - correct: 100.00 % - incorrect: 0.00 %
Class: three - correct: 100.00 % - incorrect: 0.00 %
Class: four - correct: 100.00 % - incorrect: 0.00 %
Class: ten - correct: 100.00 % - incorrect: 0.00 %
```

Figure 30: Normalised test output 2

The sampling model was a very close second to the no scaling/balancing only model coming at 70.91%, the second figure shows that the model did not manage to recognise any of the data files for the number nine, instead it predicted most of the nines as four or ten where the four sign is very similar to nine, have 4 of the fingers extended and the thumb tucked into the palm, the only difference is the orientation of the hand with the fingers pointed upwards for number 4 or to the side for number 9.

These were the outputs from my machine learning models displayed in an organised way, the outputs of the system as a whole were shown in the “Outcome and analysis” section (4), in terms of testing, since the data extractor module mainly implements the information from the API documentation I tested it as I went along to make sure I was implementing it properly and the application was successfully connecting to the device, I am also printing one of the features (palm position x) during the application use in order to make sure the device was tracking properly.

5.2. Justifying the choice of model

After the steps I took to test each model on new hand data it was very simple for me to decide which model to use in my final system since I calculated the overall accuracy of the model and also designed the testing script to give me an overview of what is going on behind the scenes, therefore I simply picked the model that provided with the highest accuracy. Even though this model could not recognise the number 6 I still decided to pick it because I expected there to be issues with that number since it is one of the more complex signs with an awkward angle and some fingers hidden behind the knuckles and

hand, these are some of the limitations of the LMC (see 5.3). On the other hand, the number nine which caused issues to the sampling model is not very complicated as the orientation of the hand should be a big giveaway as to what sign is being performed.

As seen by the testing stages, two of the four models would have been completely unusable and both include some type of feature scaling, one possible cause for this could be because feature scaling needs to be applied to the entirety of the dataset as it is the whole point of scaling the data so it is more uniform and/or within a certain range so the model is able to understand and spot patterns more easily as well as speeding up the training process. Having to scale the entire data means you have to process columns of data that may not be suitable for feature scaling, since my data has different types of data, I am referring specifically the vectors against the boolean data or the integer data columns, all the vector data is a linear type of data where the value can go from the minimum to maximum value as a double whereas the boolean data only has two possible values and the integer data can only have three possible values. I noticed that the algorithm for feature scaling was having issues with these integer and boolean data types, when scaling these values it would set a specific float value to represent each of the different possible values, for example 0.566666 for 1 and 0.122222 for 0 when normalising or 0.455555 for 1 and -0.244444 for 0 when standardising. By default this behaviour may not have been a problem however, whenever I would run the scaling algorithm again it would set these data types to different values which meant the model would be trained on certain values then reassign the data to different values which did not make sense to the model, for example assigning 0.844444 for 1 and 0.233333 for 0 instead of the example above. This meant the model would be a complete stranger to the values every time a new set of data was scaled and needed to be predicted on.

5.3. Limitations

The main limitations of this system come from the hardware used, the LMC, although being a small and powerful device it isn't specifically designed for the purposes of this project, rather it is designed for use cases in virtual reality and Ultra Leap has made their API available to developers. This section will explain some of the limitations or shortcomings of the device I have noticed while developing the system as well as some of the drawbacks of the system as a whole from a software point of view.

One of the first limitations I encountered that came straight from the LMC itself was

its very outdated and hard to find documentation and API download, the API is divided into many iterations and versions some of which come out as completely different product lines, the ones I had to use for my specific LMC model used the legacy APIs which can be difficult to navigate to. On top of this the documentation can be somewhat vague and scarce, making the implementation of the Leap motion API and the development of the application somewhat of a trial and error task at times, the main thing I did in order to overcome this issues was to use external resources such as YouTube videos and looking at forum posts to find answers.

Another big limitations which actually affected the performance and usability of the system is the LMC's accuracy can sometimes be inadequate and inconsistent, firstly it struggles with keeping track of the hand throughout the 10 seconds of performing the sign, although this does not occur every single time it is a problem about 20% of the time. Sometimes the device even confuses my head for my hand which changes the printed values of palm position from 10-15 to around 200-300, in these occurrences, I have to stop the tracking and redo it which slows down the use of the application and makes it less practical to use. The only way I have of circumventing this issue is to print the palm position so I know if the device is tracking properly.

The accuracy issues of the LMC also carries over to another issue which has to do with lighting conditions in the room where the device is being used, the device's accuracy greatly falters if there is too much light coming onto it or its surroundings, this wouldn't be a big issue if the excess-light-status happened only if a lot of light hit the device which would have been unnatural and a rare occurrence. However, this was not the case as natural sunlight in the room (not directly onto the device) was enough for the device to give me a warning, this meant that the accuracy of the device would suffer as soon as it would start to get sunny which slowed down the development process.

A limitation of the software design itself could be the fact that it is using quite a lot of dependencies and external packages due to the nature of the project, this means there is a higher chance of something going wrong or a vulnerability coming up within one of the many packages and causing a bug in the application. The structure of the system also makes it much more complicated to deploy it as an executable and have a single installer to download all the packages and make it ready to use, instead I opted for it to be a folder that needed to be downloaded by the user with a `readme.txt` file to instruct the use as to how to use the application, this would make it slightly more difficult for the user to get to the application but made the development process quicker

and easier. Furthermore, the readme file provided would give the user the information needed to start and use the application properly which shouldn't take too much extra time. The readme file in question will be similar to the user documentation section given in this report and will detail the minimum hardware and software requirement as well as detailed step by step instructions on how to properly prepare, start and use the application.

Another limitation of the system itself is the fact that it uses specialised hardware, meaning you would need to buy it and then use a USB cable to connect it, although it might not be as expensive as other methods such as the glove that uses flex sensors, it is still an extra thing that the user needs to have and set up. One method that overcomes this would be to use computer vision, however this would entirely depend on the quality of the inbuilt camera, where most computer webcams may not be suitable however a lot of phone cameras are very high-end, so building a mobile app would allow to somewhat overcome this issue but it will introduce some other drawbacks associated with computer vision (see section 2.2).

6. Conclusion

To finish off this paper I will say that through this project I was able to research and develop a somewhat conclusive system that includes the use of the Leap Motion API and machine learning model to get an accuracy of slightly over 75% which is what I was aiming for especially looking at similar systems of this type. The program is easy to use and very intuitive once installed with simple on-screen instructions, I was also able to come up with different ways of preprocessing data to maximise the performance of the machine learning module and came to the conclusion that more preprocessing does not necessarily mean a better level of accuracy. With the research done in this paper, a similar system using a more accurate and consistent alternative to the Leap Motion Controller could have a much better accuracy as most of the limitations from this system came from the hardware itself.

References

- Ambar, R. (2018). Development of a wearable device for sign language recognition.
- Baheti, P. (2022). 12 types of neural network activation functions: How to choose?
- Caroline Guardino, C.-H. C. (2014). American sign language recognition using leap motion sensor.
- Kshitij Bantupalli, Y. X. (2018). American sign language recognition using deep learning and computer vision.
- NDCS (2016). Right from the start: A campaign to improve early years support for deaf children.
- Ultraleap (2014). Leap motion developer documentation.

A. Code for the testing script

```
1 def get_files(path):
2     filelist = glob.glob(os.path.join(path, '*.csv'))
3
4     # list of files that should be at the end,
5     # needed because of the order glob gets the files
6     tail = ['data\\new_data\\10-0.csv',
7            'data\\new_data\\10-1.csv',
8            'data\\new_data\\10-2.csv',
9            'data\\new_data\\10-3.csv',
10           'data\\new_data\\10-4.csv',
11           'data\\new_data\\10-5.csv',
12           'data\\new_data\\10-6.csv',
13           'data\\new_data\\10-7.csv',
14           'data\\new_data\\10-8.csv',
15           'data\\new_data\\10-9.csv']
16
17     # add the files except those in tail
18     files = []
19     for file in filelist:
20         if file not in tail:
21             files.append(file)
22
23     files = files + tail
24
25     return files
```

Figure 31: Function to get all the files

```
1 def make_prediction(data_path, model):
2     data = pd.read_csv(data_path)
3
4     dataset = data.values
5     X = dataset[:,0:140].astype(float)
6     expected_label = data['label']
7
8     # Load and process model
9     model = tf.keras.models.load_model(model)
10    probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
11
12    # make predictions on data and find most common prediction
13    predictions = probability_model.predict(X)
14    prediction_indices = []
15    for item in predictions:
16        prediction_indices.append(np.argmax(item))
17
18    counter = 0
19    max_num = prediction_indices[0]
20    for item in prediction_indices:
21        current_frequency = prediction_indices.count(item)
22        if current_frequency > counter:
23            counter = current_frequency
24            max_num = item
25
26    labels = ['eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'ten', 'three', 'two', 'zero']
27
28    prediction = labels[max_num]
29
30    return prediction, expected_label[0]
```

Figure 32: Function to make a prediction on a file

```
1 def predict_all_files(files_list, model):
2     predictions_list = []
3
4     for file in files_list:
5         if model == 'saved_models/sampling/':
6             predictions_list.append(sampling_prediction(file))
7         else:
8             predictions_list.append(make_prediction(file, model))
9
10    return predictions_list
```

Figure 33: Function to make predictions on all files

```
1 def calculate_accuracy(pred_list):
2     different_count = 0
3     for item in pred_list:
4         if item[0] != item[1]:
5             different_count += 1
6
7     percentage_wrong = (different_count / len(pred_list)) * 100
8     percentage_right = 100 - percentage_wrong
9
10    return percentage_right, percentage_wrong
11
12
13 def rank_class_performance(pred_list):
14     predictions = np.array([*pred_list])
15     predictions = np.reshape(predictions, (11, 10, 2))
16
17     per_class_accuracy = {}
18     for group in predictions:
19         for item in group:
20             perc_right, perc_wrong = calculate_accuracy(group)
21             per_class_accuracy.update({item[1]: [perc_right, perc_wrong]})
22
23     sorted_dict = {k: v for k, v in sorted(per_class_accuracy.items(), key=lambda item: item[1])}
24     return sorted_dict
```

Figure 34: Functions to calculate model accuracy and class rankings

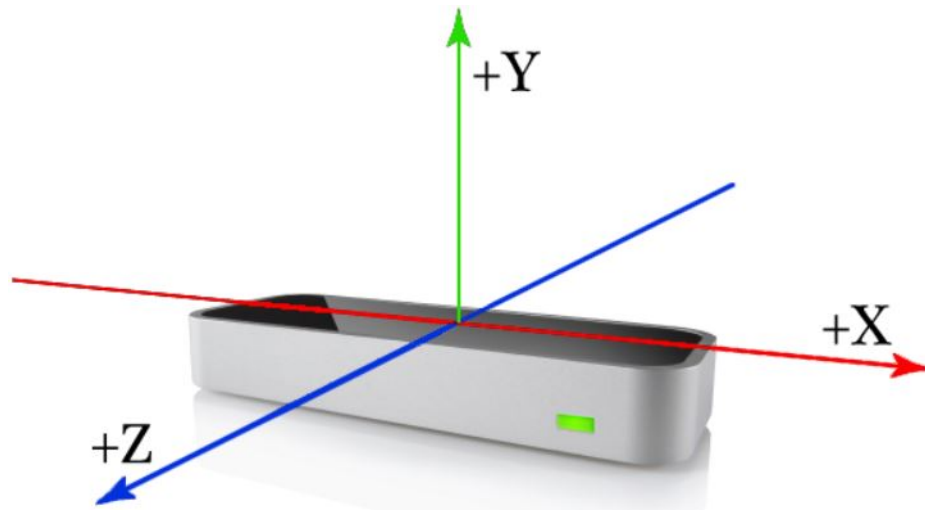
```
1 def write_to_file(pred_list, class_rankings, out_path):
2     with open(out_path, 'w') as f:
3         f.write('=====\n')
4         f.write('MODEL TESTING\n')
5         f.write('=====\n\n\n')
6
7         perc_right, perc_wrong = calculate_accuracy(pred_list)
8         f.write('Overall model accuracy\n=====\n')
9         f.write('correct: %.2f %%\n' % perc_right)
10        f.write('incorrect: %.2f %%\n' % perc_wrong)
11
12        f.write('Predicted labels\n=====\n')
13        for item in pred_list:
14            f.write('Predicted class: ' + item[0] + ' - True class: ' + item[1] + '\n')
15
16        f.write('\nPer class rankings\n=====\n')
17        for key, value in class_rankings.items():
18            f.write('Class: %s - correct: %.2f %% - incorrect: %.2f %%\n' % (key, value[0], value[1]))
```

Figure 35: Function to write information to file

B. British sign language numbers 0-20



C. Leap Motion coordinate system



D. Palm normal visual

