Assessment 2

Machine Learning
CS5062

Yassin Dinana
52094480

29th December 2020

1. Introduction:

In this paper, multi-class sequence classification is used to solve a Human Activity Recognition (HAR) problem, using two types of sensors, the accelerometer and the gyroscope which sense the human movement signals in six different states, those signals are walking, walking upstairs, walking downstairs, sitting, standing, and laying.

The problem is solved by implementing a sequential classification algorithm using the Long Short-Term Memory, also known as LSTM, which is a recurrent neural network architecture which includes a feedback allowing it to solve different sequence of data.

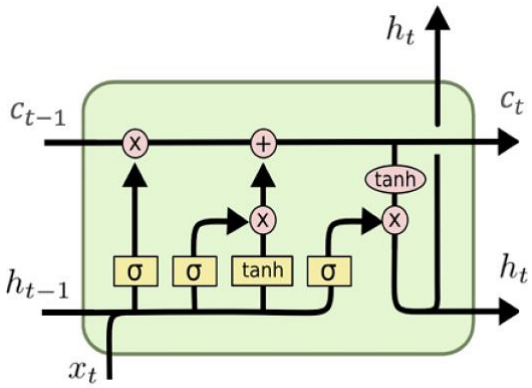

Figure 1: Long Short-Term Memory (LSTM)

Figure 1 above, shows the LSTM architecture cell, it can be seen that the output *Ct* is the cell state where at each timestep the next LSTM can either read, write, or reset it using the gating mechanism [1].

There are three different gates in the LSTM cell:

1- Input Gate: Controls weather the memory cell is updated

$$i^t = \sigma\left(W^i\left[h^{(t-1)}, x^t\right] + b^i\right) \qquad (1)$$

2- Forget Gate: Controls if the memory cell is reset to zero

$$f^t = \sigma\left(W^f\left[h^{(t-1)}, x^t\right] + b^f\right) \qquad (2)$$

3- Output Gate: Controls whether the information is visible

$$o^t = \sigma\left(W^0\left[h^{(t-1)}, x^t\right] + b^0\right) \qquad (3)$$

The given dataset includes pre-recorded videos converted into matrices including different features such as three features of 3-axial linear acceleration (x, y, and z) which is measured using the accelerometer and three other features of 3-axial angular velocity (x, y, and z) measured using the gyroscope, the two sets of data are used for training and testing.

2. Importing Data & Model Building:

The given dataset includes different factors, and it is important to import all the training and testing files correctly in order to train and test the model efficiently. At first, we need to understand and analyse the given dataset before importing the data.

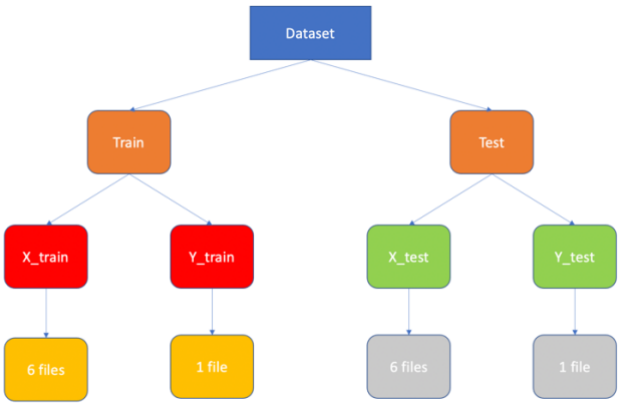

Figure 2: Dataset Architecture

Looking at the figure 2, it shows the dataset architecture, it can be seen that there are various files that need to be imported in order to use it in the task. When looking at the last nodes in figure 2, it is can be seen that there are 6 files in X_train and in X_test, those six files are analysed to be divided into two sections – the output of the accelerometer sensor in three dimensions (x, y, and z) one for each file and the other three files are the output of the gyroscope sensor in all three dimensions (x, y, z). As there are many files, it is more efficient to import all together than importing file by file, it is needed to specify the labels and signals that will be needed for the outputs.

2.1 Data Import:

```
[7] import pandas as pd
    import numpy as np
    import os
    from sklearn import metrics

    ACTIVITIES = {
        0: 'WALKING',
        1: 'WALKING_UPSTAIRS',
        2: 'WALKING_DOWNSTAIRS',
        3: 'SITTING',
        4: 'STANDING',
        5: 'LAYING',
    }

    def confusion_matrix(Y_true, Y_pred):
        Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
        Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

        return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])

    SIGNALS = [
        "acc_x",
        "acc_y",
        "acc_z",
        "gyro_x",
        "gyro_y",
        "gyro_z",
    ]
```

Figure 3: Specifying the activity, confusion matrix, and signals

Looking at the figure above, those are the first lines in the program. Firstly, the main libraries in machine learning are imported which are Pandas, Numpy, and SKlearn where those libraries are used throughout the program.

As stated in the introduction, there are six important activities to be visualised and analysed by the neural network for the human activity recognition task; those activities are introduced in the beginning of the program in figure 3, representing all the six activities in a python dictionary.

The next step is specifying a confusion matrix in a new function, a confusion matrix is simply a two dimensions table that will show the true output as well as the predicted output, it describes the performance of the classification model on the given dataset.

Finally, the signals represent the files in the X_train and Y_train folders. By looking on the text files in the folders, it will be seen that the dataset includes the output of the accelerometer in the three dimensions (x, y, and z) and the same goes for the output of the gyroscope in the same dimensions. Those are represented as the signals – which will be used when importing the dataset to the program in the next step [2].

```
[12] DATADIR = '/content/drive/MyDrive/Masters Degree/Machine Learning/Assessment 2/Dataset/MyData'

[15] pd.read_csv(DATADIR + '/train/X_train/acc_x_train.txt', delim_whitespace=True, header=None)

[16] pd.read_csv(DATADIR + '/test/y_test.txt', delim_whitespace=True, header=None)
```

Figure 4: Importing a sample of the data

Looking at figure 4, there are imported files using the pandas library function. The first imported directory identified as “DATADIR” which stands for data directory, imports the folder that includes the both the train & testing folders.

The second command, we import a sample of the X_train file using pandas + the pre-called directory DATADIR, it is also specified that the difference between the text in the text file is whitespace and there are no headers.

Finally, using the same pandas’ library, we import a sample of the Y_test file, it has the same specification as the X_train file imported regarding the white space and header.

The reason why only those two sample files are imported is because in the next step those samples will be used to import all other files at once using different functions for the files to be used to training and testing the model.

Before importing all the other files, it is necessary to make sure that the imported samples are imported correctly and therefore make sure that it has the correct number of columns and rows for both the training and the testing files.

In all machine learning models, the training sets should have more data than the testing set.

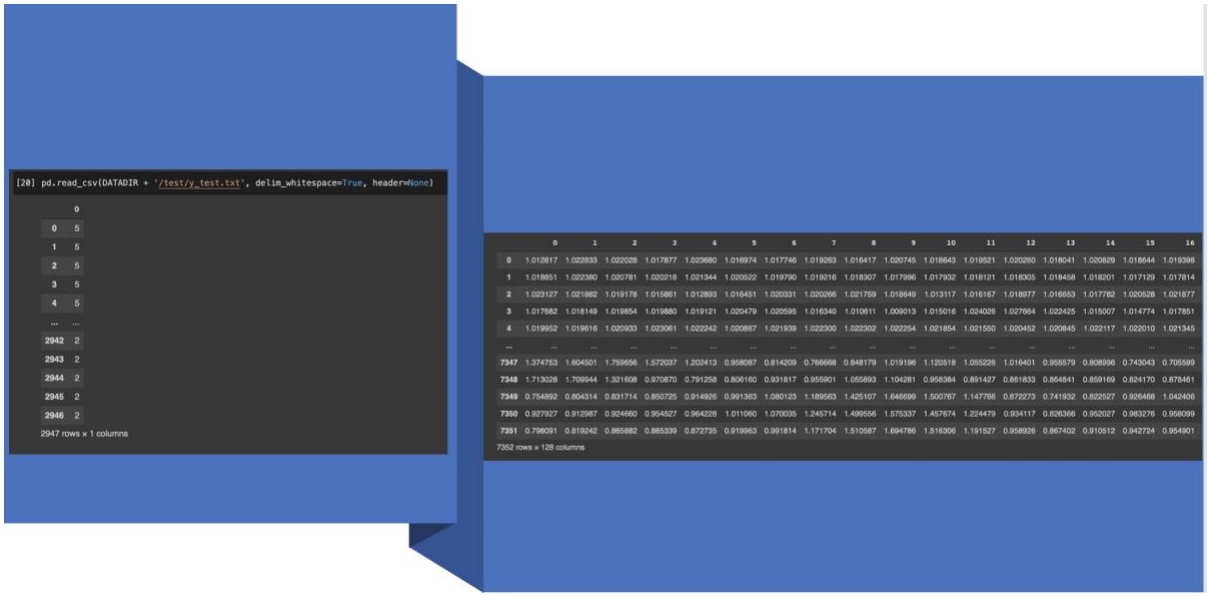

Figure 5: Output after importing sample of X_train and Y_test

The figure above shows the output after running the code snippets in figure 4, it prints the table and most importantly the shape of table, which is important to visualise to make sure it is important correctly.

The shape of the sample X_train file is given to be (*7352 rows x 128 columns)* which is the correct shape needed for a sample training file, looking on the Y_test file shape, it shows (*2947 rows x 1 column*) which is the shape needed for a sample test file.

```
[17] def _read_csv(filename):
        return pd.read_csv(filename, delim_whitespace=True, header=None)


     def load_signals(subset):
         signals_data = []

         for signal in SIGNALS:
             filename = DATADIR + '/'+subset+'/X_'+subset+'/'+signal+'_'+subset+'.txt'
             signals_data.append(
                 _read_csv(filename).to_numpy()
             )
         print(np.array(signals_data).shape)

         return np.transpose(signals_data, (1, 2, 0))
```

```
[18] def load_y(subset):

         filename = DATADIR + '/'+subset+'/y_'+subset+'.txt'
         y = _read_csv(filename)[0]

         return pd.get_dummies(y).to_numpy()
```

```
[ ]  def load_data():

         X_train, X_test = load_signals('train'), load_signals('test')
         y_train, y_test = load_y('train'), load_y('test')

         return X_train, X_test, Y_train, Y_test
```

Figure 6: Importing all remaining files of the dataset

As stated above, it is now important to import all the other remaining files from the dataset, looking at the first cell in figure 6 above, two functions are created that will load all the data files using the sample file imported in figure 4.

First, the read csv function is defined to read the filename that will be specified in the next function, the filename is read using the same specifications of the white space and headers as when importing the sample files.

The function after that loads the signals that were specified in figure 3 which are either x, y, or z for each type of sensor.

When specifying the filename in the For-Loop, it can be seen that the same as the imported sample training file is added, there is a slight change in the path explained below:

- Replacing "Train" with "+Subset+": Where this function will be able to import the path when including "Train" or "Test" and therefore, it will be running through all the files in the dataset.

- Replacing the signals "Acc_x" with "+Signals+" which was specified before and therefore going thought the dataset and importing files with different signals such as "Acc or Gyr" (Accelerometer or Gyroscope) as well as the different dimensions X, Z, or Y.

After the files imported, it can be seen in the program saves the data in an NumPy array using the NumPy library.

The resultant shape is given to be (7352 train / 2947 test for the columns, 128 timesteps for the rows, and finally there are 6 signals which were pre-defined earlier in the program and in the report. At the end of the cell, it is important to transpose in order to change the dimensions of the output.

When looking at the dataset, it can be seen that there are two different files with the name of Y_test and Y_train. Those files which have only 1 column also need to be imported, looking at figure 6, the Load_Y, loads those files using the pre-imported sample by the same method used to import the X_test and X_train files.

Finally, after loading all the data in the program, the data is saved into different variables that will be used later that are named X_train, X_test, Y_train, Y_test.

Dataset	***Dimensions***
X_train	(7352, 128, 6)
X_test	(2947, 128, 6)
Y_train	(7352, 1)
Y_test	(2947, 1)

2.2 Building the model:

After importing all the data in the program and making sure all the dimensions are correct for the imported files, is it now possible to start importing the necessary libraries to start building the LSTM model and specifying how the model should operate.

```
[67] np.random.seed(42)
     import tensorflow as tf
     from tensorflow.keras.models import load_model, Model
     from tensorflow.python.keras import backend as K
     from keras.models import Sequential
     from keras.layers import LSTM
     from keras.layers.core import Dense, Dropout
     from tensorflow.keras import layers
     from tensorflow.keras.layers import InputLayer

[65] # Utility function to count the number of classes
     def _count_classes(y):
         return len(set([tuple(category) for category in y]))

[51] #Loading All Test & Trainging Data
     X_train, X_test, Y_train, Y_test = load_data()
```

Figure 7: Importing libraries for building the model and counting classes function

As TensorFlow is the main library used in this task, it is imported as well as its sub-libraries which will be used when building the model. Keras is also important as it is an important aspect of machine learning as it provides important blocks and essential abstractions in machine learning. As stated before, LSTM can deal with sequential data, therefore, Sequential is imported from Keras as well as the classification Algorithm used in this task which is LSTM. Finally, as the project involves a neural network which will include different layers.

In order to count the number of classes in each file, the count classes function is built to count the classes' length when dealing with different categories in each file; this function is later used to count the number of classes in the file Y_train. Finally, all the loaded data imported previously is are loaded in the load.data () object to be used in the model.

```
[56] timesteps = len(X_train[0])
     input_dim = len(X_train[0][0])
     n_classes = _count_classes(Y_train)

     print(timesteps)
     print(input_dim)
     print(len(X_train))

     128
     6
     7352

[61] epochs = 10
     batch_size = 1
     n_hidden = 32
```

Figure 8: Specifying the timesteps, input dimensions and number of classes

As the LSTM model is being used for this task, it takes a 3D input, looking at figure 8, the timesteps variable is specified to the program to be 128 which is the length of the X_train file as a sample. The input dimension is specified to be 6 which are the given labels, and finally the number of classes for the training folders which will be given to the network is set to be 7352 as it was specified earlier in figure 5.

The step before implementing the model itself, the LSTM model needs to know the epochs, the batch size, and number of hidden layers, where:

- *Epochs:* Defines how many passes the model passes through the training set- it is first specified to be 10.

- *Batch Size:* It is the number of training example used in one iteration.

- *Number of hidden layers:* It is the number of layers located between the input and the output of the neural network, where all the weights are applied.

```
[ ]  model = Sequential()
     model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
     model.add(Dropout(0.2))
     model.add(Dense(n_classes, activation='sigmoid'))
     model.summary()

[ ]  # Compiling the model
     model.compile(loss='categorical_crossentropy',
                   optimizer='rmsprop',
                   metrics=['accuracy'])

[ ]  # Training the model
     model.fit(X_train,
               Y_train,
               batch_size=batch_size,
               validation_data=(X_test, Y_test),
               epochs=epochs)
```

Figure 9: Building and compiling the LSTM model [3]

In figure 9 above, the LSTM model is implemented in the first cell, where:

1- The first line of code specifies that this a sequential model able to solve sequential data tasks and not just single data using the feedback provided.

2- The model is given the input information, such as the number of hidden layers, the input shape including the timesteps and input dimensions which were specified in figure 8.

3- The dropout rate is specified to be 0.2, this rate prevents the model from overfitting, it means dropping 20% of the hidden units in the neural network.

4- The Dense layer is very imported in the LSTM, as it uses the feedback to feeds the output of the previous layer to all other neurons. The activation is also set to be Sigmoid to keep the range only between 0 and 1.

When it comes to compiling the built model, it is important to specify different aspect such that the accuracy of the model in %, specify an optimizer to be used which in our case the chosen optimizer is the “rmsprop” which is an optimizer that uses the magnitude of recent gradients to do the normalisation to other gradients in the model; by looking at its name, it uses Root Mean Squared to move towards and from gradients.

It also specified in the model compilation that it is important to visualise the training loss which is the error on the training set of data.

Finally using the two cells discussed above, it is time to run the fit on the model and visualise the output, when running the fit, both the pre-imported training sets are added which are X_train and Y_train, as well as the batch size specified before, and finally the validations set which is the testing files imported in figure 6 which are X_test and Y_test, finally the number of epochs which is due to change, it is specified as 10 as stated before [4].

3. Results:

After running the cells in figure 9, let’s visualise the output slowly.

```
#Building Model
model = Sequential()
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
model.add(Dropout(0.2))
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 32)                4992
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,190
Trainable params: 5,190
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: LSTM model summary

After building the model, the line “model.summary()” allows us to visualise the summary of the model after it is built – different layers types are seen such as the LSTM hidden layers, the dropout, and the dense layer which were discussed earlier. The summary also represents the total parameters than can be trained which are 5190 parameters.

After running the model fit function, the model will start training our data on the training set as well as the evaluation set, while running it will be showing the training accuracy, training loss, validation accuracy, and validation loss.

```
[64] # Training the model
     history = model.fit(X_train,
               Y_train,
               batch_size=batch_size,
               validation_data=(X_test, Y_test),
               epochs=epochs)

     Epoch 1/10
     7352/7352 [==============================] - 170s 23ms/step - loss: 0.9552 - accuracy: 0.5866 - val_loss: 0.8458 - val_accuracy: 0.7231
     Epoch 2/10
     7352/7352 [==============================] - 168s 23ms/step - loss: 0.5666 - accuracy: 0.8153 - val_loss: 0.6331 - val_accuracy: 0.8107
     Epoch 3/10
     7352/7352 [==============================] - 167s 23ms/step - loss: 0.4511 - accuracy: 0.8674 - val_loss: 0.7531 - val_accuracy: 0.8140
     Epoch 4/10
     7352/7352 [==============================] - 174s 24ms/step - loss: 0.3893 - accuracy: 0.8923 - val_loss: 0.6438 - val_accuracy: 0.8402
     Epoch 5/10
     7352/7352 [==============================] - 168s 23ms/step - loss: 0.3062 - accuracy: 0.9113 - val_loss: 0.8059 - val_accuracy: 0.8341
     Epoch 6/10
     7352/7352 [==============================] - 171s 23ms/step - loss: 0.2889 - accuracy: 0.9172 - val_loss: 0.6343 - val_accuracy: 0.8571
     Epoch 7/10
     7352/7352 [==============================] - 164s 22ms/step - loss: 0.2793 - accuracy: 0.9221 - val_loss: 0.8319 - val_accuracy: 0.8537
     Epoch 8/10
     7352/7352 [==============================] - 172s 23ms/step - loss: 0.2757 - accuracy: 0.9246 - val_loss: 0.6303 - val_accuracy: 0.8694
     Epoch 9/10
     7352/7352 [==============================] - 163s 22ms/step - loss: 0.2506 - accuracy: 0.9270 - val_loss: 0.7990 - val_accuracy: 0.8405
     Epoch 10/10
     7352/7352 [==============================] - 164s 22ms/step - loss: 0.2807 - accuracy: 0.9234 - val_loss: 0.8179 - val_accuracy: 0.8531
```

Figure 10: Model Fit Output

After running the model fit for 10 epochs, the results were good, as it can be seen that the accuracy is increasing gradually from the first to the 10th epoch reaching 92.3%, unlike the training loss which decreased gradually from the first till the last epoch.

As discussed before, the confusion matrix shows the both the predicted and the real output. Therefore, the confusion matrix is printed using the Y_test and X_test variables after training the model, the output be seen in the figure below.

```
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

Pred	LAYING	SITTING	...	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
True			...		
LAYING	528	1	...	2	0
SITTING	0	300	...	2	28
STANDING	0	40	...	12	14
WALKING	0	0	...	86	7
WALKING_DOWNSTAIRS	0	0	...	407	8
WALKING_UPSTAIRS	0	0	...	54	410

[6 rows x 6 columns]
<function confusion_matrix at 0x7fa77cf989d8>

Figure 11: Model Printing the confusion matrix

The printed confusion matrix can be seen in figure 11 above, which will be explained and analysed in the next section using a plot. It shows the predicted output as well as the real output after training the model for all the labels and activities specified in the beginning of the program.

4. Analysing the results:

When it comes to analysing the results, plots and graphs are used to visualize the outputs, by looking at the training of our model to the data, it shows that it is working good as the accuracy increased gradually and the training loss decreased gradually by passing on the 10 epochs.

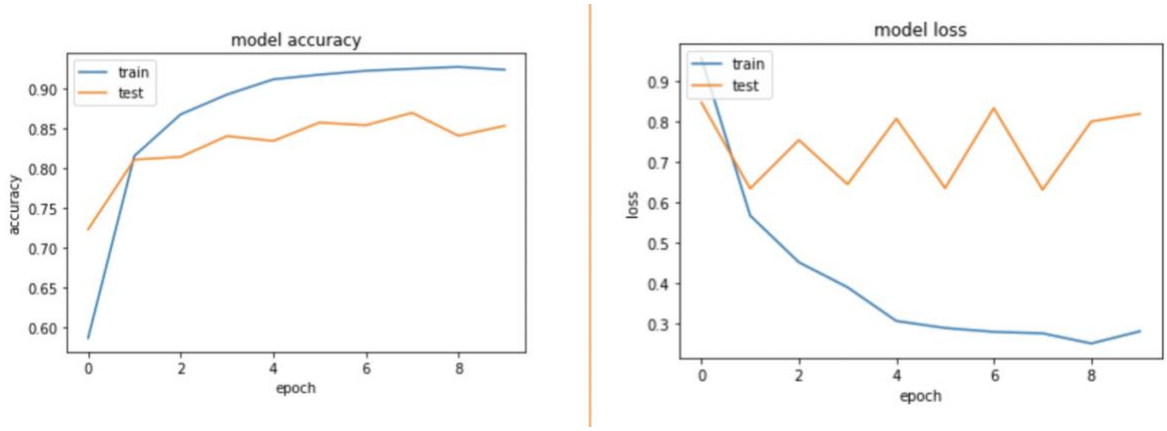

Figure 12: Plotting the training results

In figure 12 above, there are two plots available, the first plot analyses the model accuracy. By looking at the model accuracy plot, it is an accuracy times epochs plot, where the blue curve represents the training, and the orange curve represents the testing [4].

It can be seen that at the first epoch, the training start at 58% and kept increasing gradually as long as the epochs increased. When the last epoch was reached, which is the 10th epoch, the model successfully reached an accuracy of 92.3%. When the model was presented to a new set of data, better known as the testing data, the model started in the first epoch at 72%, what shows that the model training was successful is because the testing also increased gradually until the 10th epoch reaching 85.3%.

Looking at the plot on the left, it shows the model loss against the epochs, where again, the blue curve represents the training, and the orange curve represents the testing. It can be seen that the training loss decreased gradually when starting at the first epoch with 95% and reaching the 10th epoch with only 28% which shows that the loss decreased fast confirming the success of the model. Where it can also be seen on the testing data which decreased gradually, where it started at 84% and gradually decreased reaching 81% in the last epoch.

After understanding, analysing and plotting the results, as we have printed the confusion matrix in figure 11, a heatmap is plotted explaining the confusion matrix and the correlation between the predicted values and the real values [5].

```
[70] cm = confusion_matrix(Y_test, model.predict(X_test))

     plt.imshow(cm, cmap=plt.cm.Blues)
     plt.xlabel("Predicted labels")
     plt.ylabel("True labels")
     plt.xticks([], [])
     plt.yticks([], [])
     plt.title('Confusion matrix ')
     plt.colorbar()
     plt.show()
```

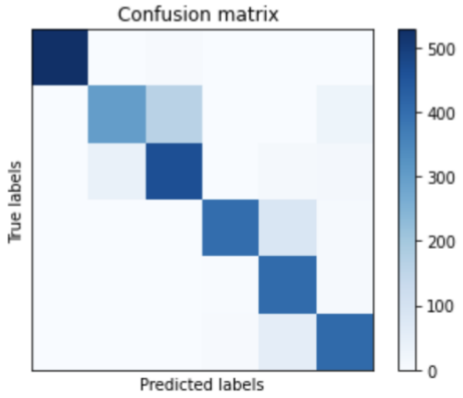

Figure 13: Plotting the confusion matrix

Figure 13 above, plots the confusion matrix that was printed in figure 11. It is a heatmap showing the correlation and relationship between the real labels / output values and the predicted ones. When a box is darker, it means it has a great effect on other aspects and is very close to the predicted value, whereas, when a box is light, it has a low relationship with the predicted value and can be ignored. The confusion matrix reads values between 0 and 500 analysing the relationships and prediction accuracy [5].

5. Conclusion:

To conclude, this task uses LSTM to solve a human activity recognition problem. Many different issues were faced when solving the task such as importing the dataset which includes multiple files and therefore it was needed to import the dataset using the subset and signals functions at once and then loading the dataset to new variables such as X_train, Y_train, X_test, and Y_test.

Tensorflow Keras was mainly used to build the LSTM model, train it, and test it. After building the model and importing the data into the model with the correct dimensions, the model showed very good accuracy rate of 92.3% increasing with each epoch and showed a very low rate of training loss and validation loss which decreased gradually by epochs.

When the model is trained more, it has a higher accuracy by changing the batch size and the number of classes. Finally, plots and heatmaps represented all the outputs which helped with better understanding of the problem and the solutions.

6. References:

[1] V.Valkov. 2014. “Time series classification form human activity recognition with LSTMs using TensorFlow 2 and Keras”. Available at: https://towardsdatascience.com/time-series-classification-for-human-activity-recognition-with-lstms-using-tensorflow-2-and-keras-b816431afdff [Online]. Accessed: 25 November 2020.

[2] A. Pavai. 2018. “Sensor-based human activity recognition using bidirectional LSTM for closely related activities”. Available at: https://core.ac.uk/download/pdf/212814302.pdf [Online]. Accessed 25 November 2020.

[3] R.Verma. 2017. “Human Activity Recognition using LSTM” Available at: https://www.kaggle.com/machinoai/human-activity-recognition-using-lstm [Online]. Accessed 23 November 2020.

[4] S.Amin. 2020. “Human Activity Recognition – Using Deep Learning Model”. Available at: https://www.geeksforgeeks.org/human-activity-recognition-using-deep-learning-model/ [Online]. Accessed 28 November 2020.

[5] R.Wainwright, A.Shenfield. “Human Activity Recognition Making Use of Long Short -Term Memory technique”. Available at: https://www.athensjournals.gr/sciences/2019-6-1-2-Wainwright.pdf [Online]. Accessed 29 November 2020.