Human Activity Recognition

Subtask 1:

Develop learning-based model(s) to classify the sequential data provided in the dataset. The model(s) can be obtained by using simple RNN, LSTM, bi-direction recurrent network, CNN, and reinforcement learning. The settings of implementing model(s) can be chosen by yourself, such as the number of layers, activation functions, and optimisers, etc. However, different settings may lead to different performance. You should provide a comprehensive explanation on how to implement the model(s) including the steps of importing data, processing data, and building model(s) **(20 marks)**

Data preparation.

The program starting from the necessary libraries import.

```
from collections import defaultdict
from pathlib import Path
import pandas as pd

import matplotlib.pyplot as plt
import numpy as np
```

Arranging the paths to workspace, training, and testing paths.

```
import os
os.chdir("D:/MScAI/Semester1/CS5062MachineLearning/AssessmII/(
```

```
X_train_path = "D:/MScAI/Semester1/CS5062MachineLearning/Asse
X_test_path = "D:/MScAI/Semester1/CS5062MachineLearning/Asses
```

Loading training data from files:

```
#Loading training data from files
X_filelist = os.listdir(X_train_path)
print(X_filelist)

X_train=list()
for name in X_filelist:
    data = pd.read_csv(X_train_path+name, header=None, delim_whitespace=True)
    X_train.append(data)
# stack group so that features are the 3rd dimension
X_train = np.dstack(X_train)
X_train[0:5]
X_train.shape
```

```
['acc_x_train.txt', 'acc_y_train.txt', 'acc_z_train.txt', 'gyro_x_train.txt',
'gyro_y_train.txt', 'gyro_z_train.txt']
```

```
(7352, 128, 6)
```

```
y_train_df = pd.read_csv("D:/MScAI/Semester1/CS5062MachineLearning/AssessmII/
#print(y_train_df)
print(y_train_df.shape)
```

```
(7352, 1)
```

Loading testing data from files:

```
#Loading testing data from files
X_filelist = os.listdir(X_test_path)
print(X_filelist)

X_test=list()
for name in X_filelist:
    data = pd.read_csv(X_test_path+name, header=None, delim_whitespace=True)
    X_test.append(data)
# stack group so that features are the 3rd dimension
X_test = np.dstack(X_test)
X_test[0:5]
X_test.shape
```

```
['acc_x_test.txt', 'acc_y_test.txt', 'acc_z_test.txt', 'gyro_x_test.txt', 'gyro
_y_test.txt', 'gyro_z_test.txt']
```

```
(2947, 128, 6)
```

```
y_test_df = pd.read_csv("D:/MScAI/Semester1/CS5062MachineLearning/AssessmII/CS50
#print(y_test_df)
print(y_test_df.shape)
```

```
(2947, 1)
```

The breakdown analysis shows that the data built exceptionally good and all classes are represented equally in training and testing data. The breakdown analysis:

```
# summarize the balance of classes in an output variable column
def class_breakdown(data):
    # convert the numpy array into a dataframe
    df = pd.DataFrame(data)
    # group data by the class value and calculate the number of rows
    counts = df.groupby(0).size()
    # retrieve raw rows
    counts = counts.values
    # summarize
    for i in range(len(counts)):
        percent = counts[i] / len(df) * 100
        print('Class=%d, total=%d, percentage=%.3f' % (i+1, counts[i], percent))
```

```
# summarize class breakdown
print('Train Dataset')
class_breakdown(y_train_df)
```

```
Train Dataset
Class=1, total=1226, percentage=16.676
Class=2, total=1073, percentage=14.595
Class=3, total=986, percentage=13.411
Class=4, total=1286, percentage=17.492
Class=5, total=1374, percentage=18.689
Class=6, total=1407, percentage=19.138
```

```
# summarize class breakdown
print('Test Dataset')
class_breakdown(y_test_df)
```

```
Test Dataset
Class=1, total=496, percentage=16.831
Class=2, total=471, percentage=15.982
Class=3, total=420, percentage=14.252
Class=4, total=491, percentage=16.661
Class=5, total=532, percentage=18.052
Class=6, total=537, percentage=18.222
```

```
# summarize combined class breakdown
print('Both')
combined = np.vstack((y_train_df, y_test_df))
class_breakdown(combined)
```

```
Both
Class=1, total=1722, percentage=16.720
Class=2, total=1544, percentage=14.992
Class=3, total=1406, percentage=13.652
Class=4, total=1777, percentage=17.254
Class=5, total=1906, percentage=18.507
Class=6, total=1944, percentage=18.876
```

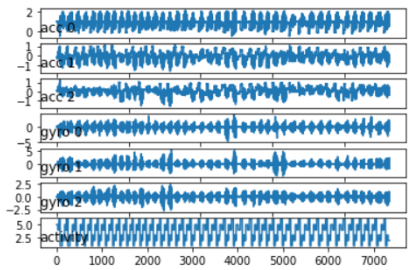
The data is comprised of windows with overlap. We can write a function to remove this overlap and squash the windows down for a given variable into one long sequence that can be plotted directly as a line plot. The next I am creating a function to convert a series of windows to a 1D list, and creating numpy arrays for results data:

```
# convert a series of windows to a 1D list
def to_series(windows):
    series = list()
    for window in windows:
        # remove the overlap from the window
        half = int(len(window) / 2) - 1
        for value in window[-half:]:
            series.append(value)
    return series
```

```
y_train = np.array(y_train_df)
y_test = np.array(y_test_df)
```

The next I am plotting the data, from each data file in a series of plots. The plots are arranged vertically at the same timeline, so we can see the data and compare the data from the different data files.

```
from matplotlib import pyplot

pyplot.figure()
# determine the total number of plots
n, off = X_train.shape[2] + 1, 0
# plot acc
for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X_train[:, :, off]))
    pyplot.title('acc '+str(i), y=0, loc='left')
    off += 1
# plot gyro
for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X_train[:, :, off]))
    pyplot.title('gyro '+str(i), y=0, loc='left')
    off += 1
pyplot.subplot(n, 1, n)
pyplot.plot(y_train_df)
pyplot.title('activity', y=0, loc='left')
pyplot.show()
```

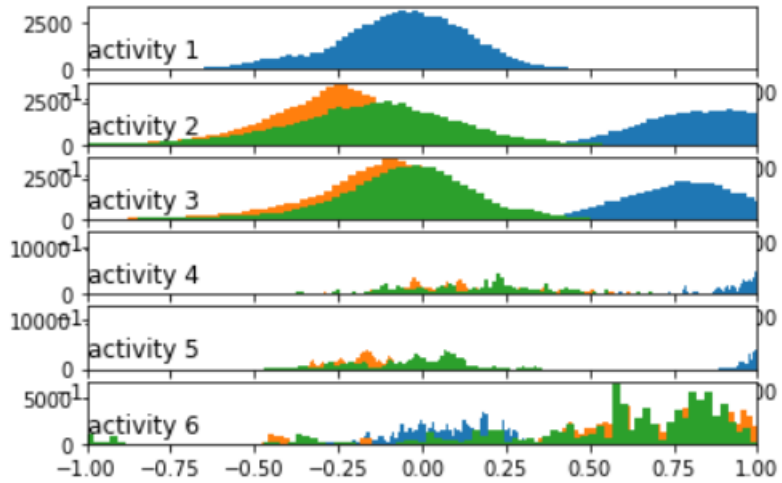


From plots we can see that the amplitude is growing when the activity is changing this growth corresponds to the active exercises such as walking, walking upstairs and downstairs, but it is only a guess at this stage. We can only assume that each parameter variates according to the activity change which means that our data loaded correctly.

The next plot shows the histograms for each activity. The histogram based on the assumption that the data scaled between -1 and 1, presumably per subject, which means that the amplitude of the detected movements will be similar. The plots and code:

```
# group data by activity
def data_by_activity(X, y, activities):
    # group windows by activity
    return {a:X[y[:,0]==a, :, :] for a in activities}
```

```
# plot histograms for each activity for a subject

# get a list of unique activities for the subject
activity_ids = np.unique(y_train[:,0])
# group windows by activity
grouped = data_by_activity(X_train, y_train, activity_ids)
# plot per activity, histograms for each axis
pyplot.figure()
xaxis = None
for k in range(len(activity_ids)):
    act_id = activity_ids[k]
    # total acceleration
    for i in range(3):
        ax = pyplot.subplot(len(activity_ids), 1, k+1, sharex=xaxis)
        ax.set_xlim(-1,1)
        if k == 0:
            xaxis = ax
        pyplot.hist(to_series(grouped[act_id][:,:,i]), bins=100)
        pyplot.title('activity '+str(act_id), y=0, loc='left')
pyplot.show()
```



We can see from the histograms that distribution is gaussian and the data is in the separate big groups. The most important for us that the data is consistent and can be analysed to make predictions for the new data.

Model building and training.

The model building needs to import necessary libraries. And we need to do some more data preparation to parse into the model. Here we are changing our results array from array to the categorical matrix representation. Arranging the verbose=1 to see the progress information (slows down a process can be 0 to do not show), the epochs = 25 to pass 25 epochs through the training data

with different chunks of the data sequence to improve training, the batch size 64 means the model will be considering the chunks of the data size 64 each suggested to choose this data in the power of 2 to improve speed on GPU. The last we are reshaping our data into the subsequence's (samples, time steps, rows, cols, channels).

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.layers import TimeDistributed
from keras.layers import ConvLSTM2D
from keras.utils import to_categorical
```

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

verbose, epochs, batch_size = 1, 25, 64
n_timesteps, n_features, n_outputs = X_train.shape[1], X_train.shape[2], y_train.shape[1]
# reshape into subsequences (samples, time steps, rows, cols, channels)
n_steps, n_length = 4, 32
X_train = X_train.reshape((X_train.shape[0], n_steps, 1, n_length, n_features))
X_test = X_test.reshape((X_test.shape[0], n_steps, 1, n_length, n_features))
```

The next step we are going to define and fit our CNN LSTM model. The CNN LSTM model is built the way where we perform the convolutions of the CNN as part of the LSTM. In our model the CNN reads the input data, and LSTM used on the input data to support sequence prediction.

"CNN LSTMs were developed for visual time series prediction problems and the application of generating textual descriptions from sequences of images (e.g. videos). Specifically, the problems of:

- **Activity Recognition**: Generating a textual description of an activity demonstrated in a sequence of images.
- **Image Description**: Generating a textual description of a single image.
- **Video Description**: Generating a textual description of a sequence of images."

(https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/)

```
# define model
model = Sequential()
model.add(ConvLSTM2D(filters=64, kernel_size=(1,3), activation='relu', input_shape=(n_steps, 1, n_length, n_features)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit network
history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test), verbose=verbose)
```

```
y: 0.8504
Epoch 20/25
7352/7352 [==============================] - 7s 1ms/step - loss: 0.0951 - accuracy: 0.9573 - val_loss: 0.8069 - val_accuracy:
0.8744
Epoch 21/25
7352/7352 [==============================] - 7s 995us/step - loss: 0.0976 - accuracy: 0.9585 - val_loss: 1.1043 - val_accurac
y: 0.8334
Epoch 22/25
7352/7352 [==============================] - 7s 1ms/step - loss: 0.1049 - accuracy: 0.9543 - val_loss: 0.7502 - val_accuracy:
0.8880
Epoch 23/25
7352/7352 [==============================] - 7s 1ms/step - loss: 0.0873 - accuracy: 0.9601 - val_loss: 0.9462 - val_accuracy:
0.8558
Epoch 24/25
7352/7352 [==============================] - 7s 1ms/step - loss: 0.0859 - accuracy: 0.9588 - val_loss: 1.0165 - val_accuracy:
0.8663
Epoch 25/25
7352/7352 [==============================] - 8s 1ms/step - loss: 0.0849 - accuracy: 0.9623 - val_loss: 1.0092 - val_accuracy:
0.8724
```

The other model is Bidirectional LSTM model. The LSTM (long short term memory) model is a type of recurrent network which is learning order dependence in sequence prediction. As the model before CNN-LSTM, this model takes input data with Bidirectional layer and inside of this layer performs LSTM analysis to perform sequence prediction. I tried to keep the model structure the same as in previous case to make comparison fairer. The difference is that I could not apply the flatten layer to this layer because of the data specific.

Short additional preparation:

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore', sparse=False)

enc = enc.fit(y_train)

y_train = enc.transform(y_train)
y_test = enc.transform(y_test)
```

Model:

```
from keras.models import Sequential
from keras.layers import Dense
#from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
#from keras.layers import TimeDistributed
from keras.layers import Bidirectional
#from keras.utils import to_categorical
```

```
model = Sequential()
model.add(Bidirectional(LSTM(
          units=128,
          input_shape=[X_train.shape[1], X_train.shape[2]])))
model.add(Dropout(rate=0.5))

model.add(Dense(units=128, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))

model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['acc'])
```

Training model and validating with the test set. It is recommended to separate the training data on training and validation set, but because of the time given I decided to use testing data as validation.

```
history = model.fit(
    X_train, y_train,
    epochs=25,
    batch_size=64,
    validation_data=(X_test, y_test))
```

```
7352/7352 [==============================] - 22s 3ms/step - loss: 0.16
04 - acc: 0.9381 - val_loss: 0.4612 - val_acc: 0.8887
Epoch 20/25
7352/7352 [==============================] - 22s 3ms/step - loss: 0.12
99 - acc: 0.9474 - val_loss: 0.4854 - val_acc: 0.8965
Epoch 21/25
7352/7352 [==============================] - 22s 3ms/step - loss: 0.13
31 - acc: 0.9464 - val_loss: 0.5347 - val_acc: 0.8836
Epoch 22/25
7352/7352 [==============================] - 22s 3ms/step - loss: 0.11
90 - acc: 0.9499 - val_loss: 0.5405 - val_acc: 0.8941
Epoch 23/25
7352/7352 [==============================] - 22s 3ms/step - loss: 0.12
40 - acc: 0.9478 - val_loss: 0.6402 - val_acc: 0.8877
Epoch 24/25
7352/7352 [==============================] - 22s 3ms/step - loss: 0.14
90 - acc: 0.9445 - val_loss: 0.3378 - val_acc: 0.8985
Epoch 25/25
7352/7352 [==============================] - 23s 3ms/step - loss: 0.13
17 - acc: 0.9482 - val_loss: 0.4096 - val_acc: 0.8945
```

Subtask 2:

To inspect the intermediate results of the training process, you need to train loss and training accuracy. The confusion matrix of each model, based on the predictions of your developed model(s) and corresponding labels in the provided dataset, must be plotted as well. Use the following three metrics to report the model's performance, i.e. Precision/Recall, Accuracy and Area under the curve (AUROC). When reporting performance, please only use the test set to evaluate the performance (15 marks)

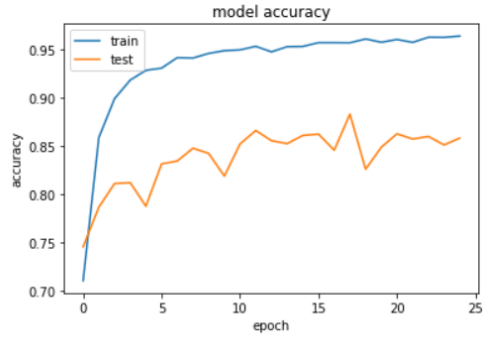
Accuracy for CNN-LSTM:

```
model.evaluate(X_test, y_test)
```

```
2947/2947 [==============================] - 1s 191us/step
```

```
[1.0091601451691583, 0.8724126219749451]
```

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Accuracy for Bidirectional-LSTM:

```
model.evaluate(X_test, y_test)
```

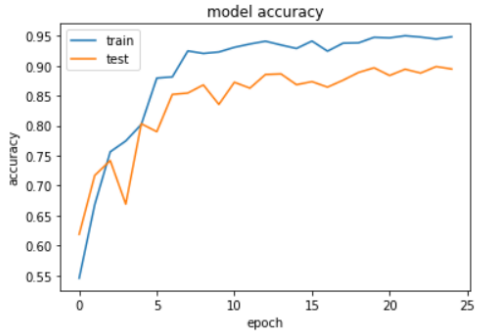
```
2947/2947 [==============================] - 3s 882us/step
```

```
[0.40955591633660554, 0.8944689631462097]
```

```
 #list all data in history
print(history.history.keys())
```

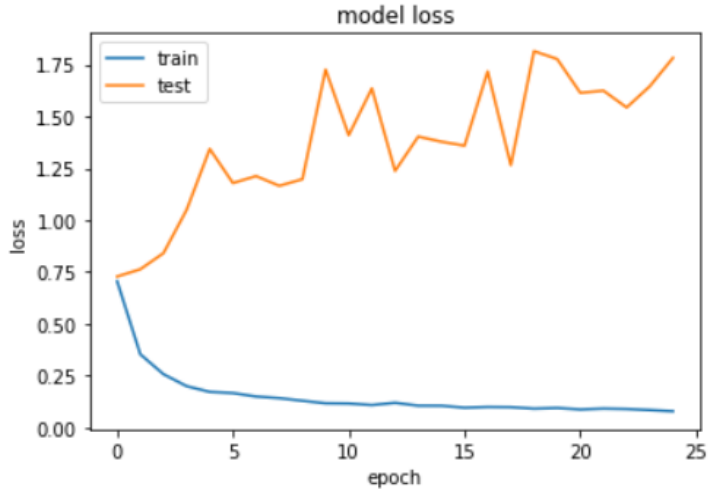
```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

```
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



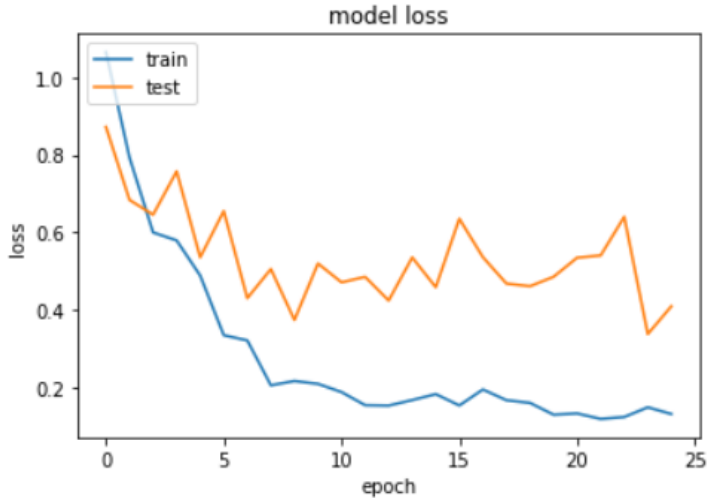
Loss for CNN-LSTM:

```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Loss for Bidirectional-LSTM:

```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Confusion matrix code is the same for both models:

```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

```
Y_pred = model.predict(X_test)
y_pred = np.argmax(Y_pred, axis=1)
y_test_arr = np.argmax(y_test, axis=-1)
print(y_pred[0:5], y_test_arr[0:5])
```

```
[4 4 4 4 4] [4 4 4 4 4]
```

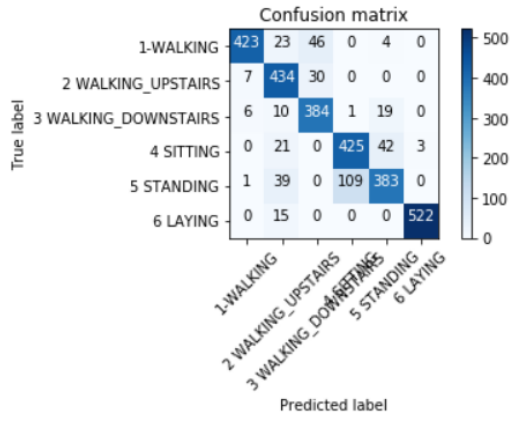
```
cm = confusion_matrix(y_test_arr, y_pred)
```

```
def plot_conf_matr(cm, classes, normalize = False, cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title('Confusion matrix')
    plt.colorbar()
    tick_marks=np.arange(len(classes))
    plt.xticks(tick_marks,classes,rotation=45)
    plt.yticks(tick_marks,classes)
    if normalize:
        cm=cm.astype('float')/cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print("Confusion matrix, without normalization")

    print(cm)

    thresh = cm.max()/2
    for i,j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                horizontalalignment="center",
                color="white" if cm[i,j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Confusion matrix for the CNN-LSTM model:

```
cm_plot_labels = ['1-WALKING', '2 WALKING_UPSTAIRS','3 WALKING_DOWNSTAIRS'
                  '4 SITTING', '5 STANDING', '6 LAYING']
plot_conf_matr(cm, cm_plot_labels)
```

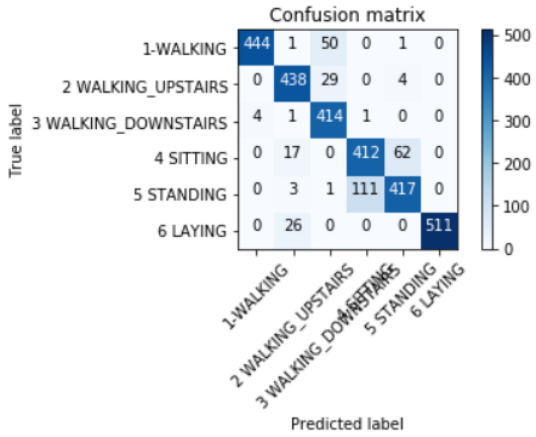
```
Confusion matrix, without normalization
[[423  23  46   0   4   0]
 [  7 434  30   0   0   0]
 [  6  10 384   1  19   0]
 [  0  21   0 425  42   3]
 [  1  39   0 109 383   0]
 [  0  15   0   0   0 522]]
```



Confusion matrix for the Bidirectional-LSTM model:

```
cm_plot_labels = ['1-WALKING', '2 WALKING_UPSTAIRS','3 WALKING_DOWNSTAIRS'
                  '4 SITTING', '5 STANDING', '6 LAYING']
plot_conf_matr(cm, cm_plot_labels)
```

```
Confusion matrix, without normalization
[[444   1  50   0   1   0]
 [  0 438  29   0   4   0]
 [  4   1 414   1   0   0]
 [  0  17   0 412  62   0]
 [  0   3   1 111 417   0]
 [  0  26   0   0   0 511]]
```



We can calculate recall and precision with the data from confusion matrix. The true positive is the sum of the main diagonal values. The false positive is the sum of the values above. The false negative is the sum of the values below. Recall = TP / (TP + FN). Precision = TP / (TP + FP). The problem here that precision/recall is used for the binary classification and we have 6 different classes, so we need to binarize this data.

For CNN-LSTM model:

```
• The x-axis showing recall
    (= sensitivity = TP / (TP + FN))
• The y-axis showing precision
    (= positive predictive value = TP / (TP + FP))
```

```
1 tp = sum(cm[i][i] for i in range(len(cm)))
2 fp = np.triu(cm).sum()-np.trace(cm)
3 fn = np.tril(cm).sum()-np.trace(cm)
4
5 recall=tp/(tp+fn)
6 precision=tp/(tp+fp)
7 print("recall    ", recall)
8 print("precision ", precision)
```

```
recall     0.9251529327096077
precision  0.9386637458926615
```

For Bidirectional-LSTM model:

```
tp = sum(cm[i][i] for i in range(len(cm)))
fp = np.triu(cm).sum()-np.trace(cm)
fn = np.tril(cm).sum()-np.trace(cm)

recall=tp/(tp+fn)
precision=tp/(tp+fp)
print("recall    ", recall)
print("precision ", precision)
```

```
recall     0.9417649160414434
precision  0.9468390804597702
```

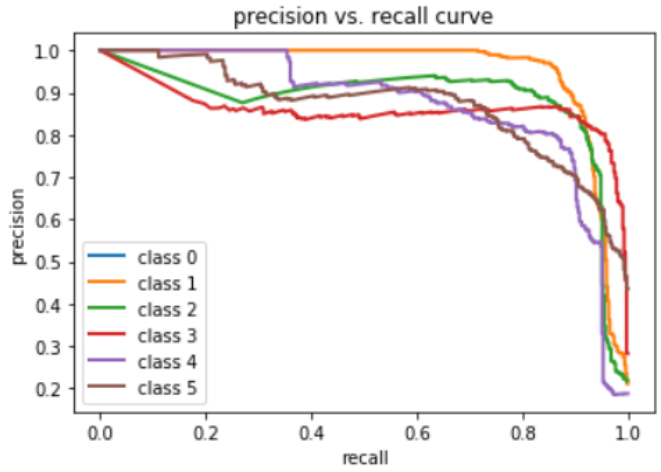
Graphs for each class related to the rest of the data CNN-LSTM:

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
```

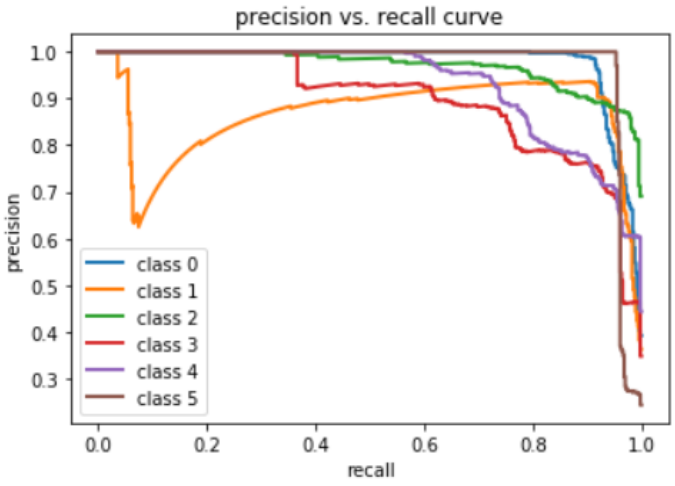
```
# precision recall curve
n_classes = 6
precision = dict()
recall = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test[:, i],
                                                        Y_pred[:, i])
    plt.plot(recall[i], precision[i], lw=2, label='class {}'.format(i))

plt.xlabel("recall")
plt.ylabel("precision")
plt.legend(loc="best")
plt.title("precision vs. recall curve")
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_ranking.py:6
81: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
```

Python gives error in some cases some of the values negative. “average_precision_score does not return correct AP when all negative ground truth labels”. This is the explanation I found online: https://github.com/scikit-learn/scikit-learn/issues/8245 I am still not sure about this error, as I do not get it for Bidirectional-LSTM model.



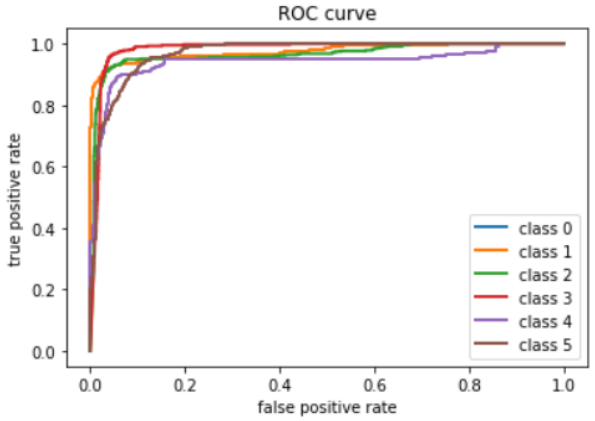
Graphs for each class related to the rest of the data Bidirectional-LSTM:



ROC curve CNN-LSTM model:

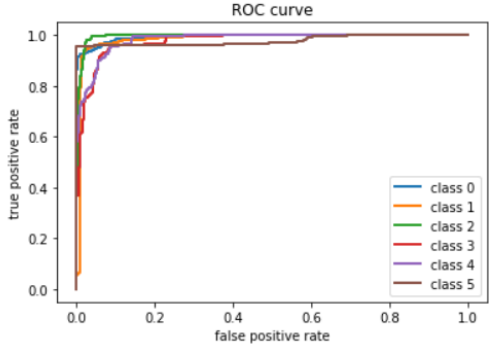
```
# roc curve
fpr = dict()
tpr = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i],
                                  Y_pred[:, i])
    plt.plot(fpr[i], tpr[i], lw=2, label='class {}'.format(i))

plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.legend(loc="best")
plt.title("ROC curve")
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_ranking.py:8
13: UndefinedMetricWarning: No positive samples in y_true, true positive
value should be meaningless
  UndefinedMetricWarning)
```



ROC curve Bidirectional-LSTM model (no error for this model):

```
# roc curve
fpr = dict()
tpr = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i],
                                  Y_pred[:, i])
    plt.plot(fpr[i], tpr[i], lw=2, label='class {}'.format(i))

plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.legend(loc="best")
plt.title("ROC curve")
plt.show()
```



Subtask 3:

Justification and evaluation: you are required to analyse, comment, and elaborate on your findings of the experimental results. Ideally, you can provide some deep explanation on why the experimental results are obtained. For example, the different parameter settings of networks (e.g. layers, activation functions, optimisers) may lead to different results. You can have a comprehensive discussion about it. This is only an example and you can find other points to have a deep exploration. (**15 marks**)

This subtask is more like conversational, not discretional. I was trying to describe every parameter I chose in this report, and yes, some parameters would be worth trying to experiment with.
One parameter is the most obvious the number of epochs with increase of the epochs number we are improving our training model, but on other side the number of epochs can do not change the overall model performance or in a worst case overfit the model and make it even worse.
The number of layers again the increase of the neuron layers in most cases improves the prediction results especially when data is big, but sometimes this will not give us big advantage, but increase the calculation cost.
The activation function is attached to each neuron in a layer, and determines when the neuron should be activated, also activation functions normalise the output in a range between 0 and 1 or -1and 1. It is very important when we do backpropagation, as more computational workload placed on activation function. These days a lot of research conducting to find an optimal activation function automatically, but this parameter is still manually tuned.
The optimizers such as gradient descent can significantly improve the model and loss function, but again we have a choice between performance and computational cost. For example, the gradient descent with small learning rate will cost us computational time and with big learning rate can miss the local minima. The regularization is the optimizer that helps to avoid overfitting when the model predicts well on training data and performs bad with the new real-world data.

All these parameters and many more can be considered to experiment with, based on knowledge, experience, and intuition we have. This subject is very interesting and I am sorry that I did not have more time to experiment with more models and parameters for these models before the assessment.

The models I used in this assessment gave particularly good result and showed high accuracy on the testing set 87% for CNN-LSTM and 89% for Bidirectional-LSTM. This is related to the data given is very consistent as we seen from the data analysis on the preparation stage. Need to say that LSTM by itself performs good with the sequence analysis, as it mentioned before this what it was developed for.