Assignment_2_MDR

December 6, 2020

1 Assignment 2 - MDR

1.1 Submitted by:

- Qazi Umer Jamil
- NUST Regn No 317920
- RIME-19

1.2 Assignment:

- Find abnormalities in ECG .
- Use any LDA, SVM.
- Bonus marks for implementation of CNN, LSTM.
- Find the best features when using ML architectures.
- Submission should contain the code, results and achieved accuracies.
- Due date December 6, 2020.

2 Intro to the Dataset

A description of the database is as follows:

- The ECG signals were from 45 patients.
- The ECG signals contained 17 classes: normal sinus rhythm, pacemaker rhythm, and 15 types of cardiac dysfunctions.
- All ECG signals were recorded at a sampling frequency of 360 [Hz] and a gain of 200 [adu / mV].

The 17 classes are as follows:

- 1 Normal sinus rhythm
- 2 Atrial premature beat
- 3 Atrial flutter
- 4 Atrial fibrillation
- 5 Supraventricular tachyarrhythmia
- 6 Pre-excitation (WPW)
- 7 Premature ventricular contraction
- 8 Ventricular bigeminy

- 9 Ventricular trigeminy
- 10 Ventricular tachycardia
- 11 Idioventricular rhythm
- 12 Ventricular flutter
- 13 Fusion of ventricular and normal beat
- 14 Left bundle branch block beat
- 15 Right bundle branch block beat
- 16 Second-degree heart block
- 17 Pacemaker rhythm

Refrence:

- Plawiak, P., 2018. Novel methodology of cardiac health recognition based on ECG signals and evolutionary-neural system. Expert Systems with Applications, 92, pp.334-349.

2.1 Importing Libraries

We will start with importing necessary libraries

[1]:
```
import os
import random
import scipy.io
import seaborn
import numpy as np
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.style
import matplotlib as mpl
mpl.style.use('ggplot')

from numpy import std
from numpy import mean

from sklearn import datasets
from sklearn.svm import SVC
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv1D, LSTM,␣
↪MaxPooling1D, BatchNormalization

device_name = tf.test.gpu_device_name()

if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

2.2 Defining default parameters

[2]:
```
# Defining default parameters

fs = 360 # Sampling frequency
samples = 3600 # no of samples
test_size = 0.2 # to be used in test train split of the data
```

2.3 Loading the dataset

The dataset was upload on Google drive and was processed in Google Colab.

[3]:
```
from google.colab import drive

drive.mount("/content/gdrive")

!ls "/content/gdrive/My Drive/MLII/"
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
'10 VT'   '13 Fusion'  '16 SDHB'  '2 APB'   '5 SVTA'  '8 Bigeminy'
'11 IVR'  '14 LBBBB'   '17 PR'    '3 AFL'   '6 WPW'   '9 Trigeminy'
'12 VFL'  '15 RBBBB'   '1 NSR'    '4 AFIB'  '7 PVC'
```

The dataset is loacted in the 'MLII' directory.

[4]:
```
base_path = "/content/gdrive/My Drive/MLII/"
entries = os.listdir(base_path)

print("Following folders are present in " + base_path + " directory: \n")
for entry in entries:
    if not "." in entry:
        print(entry)
```

```
Following folders are present in /content/gdrive/My Drive/MLII/ directory:

14 LBBBB
4 AFIB
10 VT
7 PVC
17 PR
1 NSR
3 AFL
8 Bigeminy
16 SDHB
5 SVTA
12 VFL
11 IVR
9 Trigeminy
6 WPW
13 Fusion
2 APB
15 RBBBB
```

The above folder names also represent the 17 class labels of the dataset and each folder contain data for the respective class. Next, we will load the data files with class labels so that we have data in following format: (X, y). This form of data is required for supervised learning.

[5]:
```python
# This fucntion load the .mat files from the given directory and stack them in an array and return it

def get_data( directory):
    given_list =[]
    print("Loading files from following directory: ", directory)
    subdirs = os.listdir(base_path+str(directory))
    for subdir in subdirs:
        if not ".DS" in subdir:
            path = base_path + str(directory) + "/"+ str(subdir)
            file = scipy.io.loadmat(path)
            file = file['val'][0]
            given_list.append(file)
            converted_array = np.array(given_list)
    return converted_array
```

[6]:
```python
# To load the data in X, y (Supervised Learning fromat)

def load_data(entry):
    X = get_data(entry)
    y = [entry.split()[1] for x in range(np.shape(X)[0])]
    print("Data loading from", entry, "directory is successfull")
    return X, y
```

[7]:
```python
# For data splitting

def split_data(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
↪test_size=test_size, random_state=4)
    print("Data splited into train and test sets\n")
    return X_train, X_test, y_train, y_test
```

[8]:
```python
# Loadind data from the directories

for entry in entries:
    if not "." in entry:
        if "1 NSR" in entry:
            X_NSR, y_NSR = load_data(entry)
            X_train_NSR, X_test_NSR, y_train_NSR, y_test_NSR =␣
↪split_data(X_NSR, y_NSR)
        if "2 APB" in entry:
            X_APB, y_APB = load_data(entry)
            X_train_APB, X_test_APB, y_train_APB, y_test_APB =␣
↪split_data(X_APB, y_APB)
        if "3 AFL" in entry:
            X_AFL, y_AFL = load_data(entry)
            X_train_AFL, X_test_AFL, y_train_AFL, y_test_AFL =␣
↪split_data(X_AFL, y_AFL)
        if "4 AFIB" in entry:
            X_AFIB, y_AFIB = load_data(entry)
            X_train_AFIB, X_test_AFIB, y_train_AFIB, y_test_AFIB =␣
↪split_data(X_AFIB, y_AFIB)
        if "5 SVTA" in entry:
            X_SVTA, y_SVTA = load_data(entry)
            X_train_SVTA, X_test_SVTA, y_train_SVTA, y_test_SVTA =␣
↪split_data(X_SVTA, y_SVTA)
        if "6 WPW" in entry:
            X_WPW, y_WPW = load_data(entry)
            X_train_WPW, X_test_WPW, y_train_WPW, y_test_WPW =␣
↪split_data(X_WPW, y_WPW)
        if "7 PVC" in entry:
            X_PVC, y_PVC = load_data(entry)
            X_train_PVC, X_test_PVC, y_train_PVC, y_test_PVC =␣
↪split_data(X_PVC, y_PVC)
        if "8 Bigeminy" in entry:
            X_Bigeminy, y_Bigeminy = load_data(entry)
            X_train_Bigeminy, X_test_Bigeminy, y_train_Bigeminy,␣
↪y_test_Bigeminy = split_data(X_Bigeminy, y_Bigeminy)
        if "9 Trigeminy" in entry:
            X_Trigeminy, y_Trigeminy = load_data(entry)
```

```
            X_train_Trigeminy, X_test_Trigeminy, y_train_Trigeminy,␣
↪y_test_Trigeminy = split_data(X_Trigeminy, y_Trigeminy)
        if "10 VT" in entry:
            X_VT, y_VT = load_data(entry)
            X_train_VT, X_test_VT, y_train_VT, y_test_VT = split_data(X_VT,␣
↪y_VT)
        if "11 IVR" in entry:
            X_IVR, y_IVR = load_data(entry)
            X_train_IVR, X_test_IVR, y_train_IVR, y_test_IVR =␣
↪split_data(X_IVR, y_IVR)
        if "12 VFL" in entry:
            X_VFL, y_VFL = load_data(entry)
            X_train_VFL, X_test_VFL, y_train_VFL, y_test_VFL =␣
↪split_data(X_VFL, y_VFL)
        if "13 Fusion" in entry:
            X_Fusion, y_Fusion = load_data(entry)
            X_train_Fusion, X_test_Fusion, y_train_Fusion, y_test_Fusion =␣
↪split_data(X_Fusion, y_Fusion)
        if "14 LBBBB" in entry:
            X_LBBBB, y_LBBBB = load_data(entry)
            X_train_LBBBB, X_test_LBBBB, y_train_LBBBB, y_test_LBBBB =␣
↪split_data(X_LBBBB, y_LBBBB)
        if "15 RBBBB" in entry:
            X_RBBBB, y_RBBBB = load_data(entry)
            X_train_RBBBB, X_test_RBBBB, y_train_RBBBB, y_test_RBBBB =␣
↪split_data(X_RBBBB, y_RBBBB)
        if "16 SDHB" in entry:
            X_SDHB, y_SDHB = load_data(entry)
            X_train_SDHB, X_test_SDHB, y_train_SDHB, y_test_SDHB =␣
↪split_data(X_SDHB, y_SDHB)
        if "17 PR" in entry:
            X_PR, y_PR = load_data(entry)
            X_train_PR, X_test_PR, y_train_PR, y_test_PR = split_data(X_PR,␣
↪y_PR)
```

```
Loading files from following directory:  14 LBBBB
Data loading from 14 LBBBB directory is successfull
Data splited into train and test sets

Loading files from following directory:  4 AFIB
Data loading from 4 AFIB directory is successfull
Data splited into train and test sets

Loading files from following directory:  10 VT
Data loading from 10 VT directory is successfull
Data splited into train and test sets
```

```
Loading files from following directory:  7 PVC
Data loading from 7 PVC directory is successfull
Data splited into train and test sets

Loading files from following directory:  17 PR
Data loading from 17 PR directory is successfull
Data splited into train and test sets

Loading files from following directory:  1 NSR
Data loading from 1 NSR directory is successfull
Data splited into train and test sets

Loading files from following directory:  3 AFL
Data loading from 3 AFL directory is successfull
Data splited into train and test sets

Loading files from following directory:  8 Bigeminy
Data loading from 8 Bigeminy directory is successfull
Data splited into train and test sets

Loading files from following directory:  16 SDHB
Data loading from 16 SDHB directory is successfull
Data splited into train and test sets

Loading files from following directory:  5 SVTA
Data loading from 5 SVTA directory is successfull
Data splited into train and test sets

Loading files from following directory:  12 VFL
Data loading from 12 VFL directory is successfull
Data splited into train and test sets

Loading files from following directory:  11 IVR
Data loading from 11 IVR directory is successfull
Data splited into train and test sets

Loading files from following directory:  9 Trigeminy
Data loading from 9 Trigeminy directory is successfull
Data splited into train and test sets

Loading files from following directory:  6 WPW
Data loading from 6 WPW directory is successfull
Data splited into train and test sets

Loading files from following directory:  13 Fusion
Data loading from 13 Fusion directory is successfull
Data splited into train and test sets
```

```
Loading files from following directory:  2 APB
Data loading from 2 APB directory is successfull
Data splited into train and test sets

Loading files from following directory:  15 RBBBB
Data loading from 15 RBBBB directory is successfull
Data splited into train and test sets
```

In above loaded the data for each class, and also split it into training and testing sets as well.

Next, we will stack/combine out data for all of the classes in (X, y) form for our supervised learning models.

[9]:
```
# Stacking the X_train data
X_train = np.vstack((X_train_NSR,
                     X_train_APB,
                     X_train_AFL,
                     X_train_AFIB,
                     X_train_SVTA,
                     X_train_WPW,
            X_train_PVC,
            X_train_Bigeminy,
            X_train_Trigeminy,
            X_train_VT,
            X_train_IVR,
            X_train_VFL,
            X_train_Fusion,
            X_train_LBBBB,
            X_train_RBBBB,
            X_train_SDHB,
            X_train_PR,
            X_train_AFIB,
            X_train_SVTA,
            X_train_WPW,
            X_train_PVC,
            X_train_Bigeminy,
            X_train_Trigeminy,
            X_train_VT,
            X_train_IVR,
            X_train_VFL,
            X_train_Fusion,
            X_train_LBBBB,
            X_train_RBBBB,
            X_train_SDHB,
            X_train_PR))

# Stacking the X_test data
X_test = np.vstack((X_test_NSR,
```

```
                      X_test_APB,
                      X_test_AFL,
                      X_test_AFIB,
                      X_test_SVTA,
                      X_test_WPW,
             X_test_PVC,
             X_test_Bigeminy,
             X_test_Trigeminy,
             X_test_VT,
             X_test_IVR,
             X_test_VFL,
             X_test_Fusion,
             X_test_LBBBB,
             X_test_RBBBB,
             X_test_SDHB,
             X_test_PR,
             X_test_AFIB,
             X_test_SVTA,
             X_test_WPW,
             X_test_PVC,
             X_test_Bigeminy,
             X_test_Trigeminy,
             X_test_VT,
             X_test_IVR,
             X_test_VFL,
             X_test_Fusion,
             X_test_LBBBB,
             X_test_RBBBB,
             X_test_SDHB,
             X_test_PR))

print("Size of X_train",  np.shape(X_train))
print("Size of X_test",  np.shape(X_test))
```

```
Size of X_train (1296, 3600)
Size of X_test (335, 3600)
```

[10]:
```
# Stacking the y_train data

y_train = np.column_stack(([y_train_NSR],
                      [y_train_APB],
                      [y_train_AFL],
                      [y_train_AFIB],
                      [y_train_SVTA],
                      [y_train_WPW],
             [y_train_PVC],
             [y_train_Bigeminy],
```

```
            [y_train_Trigeminy],
            [y_train_VT],
            [y_train_IVR],
            [y_train_VFL],
            [y_train_Fusion],
            [y_train_LBBBB],
            [y_train_RBBBB],
            [y_train_SDHB],
            [y_train_PR],
            [y_train_AFIB],
            [y_train_SVTA],
            [y_train_WPW],
            [y_train_PVC],
            [y_train_Bigeminy],
            [y_train_Trigeminy],
            [y_train_VT],
            [y_train_IVR],
            [y_train_VFL],
            [y_train_Fusion],
            [y_train_LBBBB],
            [y_train_RBBBB],
            [y_train_SDHB],
            [y_train_PR]))

y_train = y_train[0]

# Stacking the y_test data
y_test = np.column_stack(([y_test_NSR],
                     [y_test_APB],
                     [y_test_AFL],
                     [y_test_AFIB],
                     [y_test_SVTA],
                     [y_test_WPW],
            [y_test_PVC],
            [y_test_Bigeminy],
            [y_test_Trigeminy],
            [y_test_VT],
            [y_test_IVR],
            [y_test_VFL],
            [y_test_Fusion],
            [y_test_LBBBB],
            [y_test_RBBBB],
            [y_test_SDHB],
            [y_test_PR],
            [y_test_AFIB],
            [y_test_SVTA],
            [y_test_WPW],
```

```
              [y_test_PVC],
              [y_test_Bigeminy],
              [y_test_Trigeminy],
              [y_test_VT],
              [y_test_IVR],
              [y_test_VFL],
              [y_test_Fusion],
              [y_test_LBBBB],
              [y_test_RBBBB],
              [y_test_SDHB],
              [y_test_PR]))

y_test = y_test[0]

print("Size of y_train",  np.shape(y_train))
print("Size of y_test",  np.shape(y_test))
```

```
Size of y_train (1296,)
Size of y_test (335,)
```

[11]:
```
# Label Encoding

labelencoder = LabelEncoder()
y_train = labelencoder.fit_transform(y_train)
y_test = labelencoder.fit_transform(y_test)
```

[12]:
```
# Number of classes:

K = len(set(y_train))
print("Total number of classes present in the dataset: ", K)
```

```
Total number of classes present in the dataset:  17
```

[13]:
```
print("Size of y_train",  np.shape(y_train))
print("Size of y_test",  np.shape(y_test))
```

```
Size of y_train (1296,)
Size of y_test (335,)
```

[14]:
```
#Shuffling the datasets

X_train, y_train = shuffle(X_train, y_train)
X_test, y_test = shuffle(X_test, y_test)
```

3 Data Normaliztion

Since, data can be in different ranges, it is helpful for model fitting to normalize the dataset.

[15]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

4 Data Visualiztion/Spectrogram

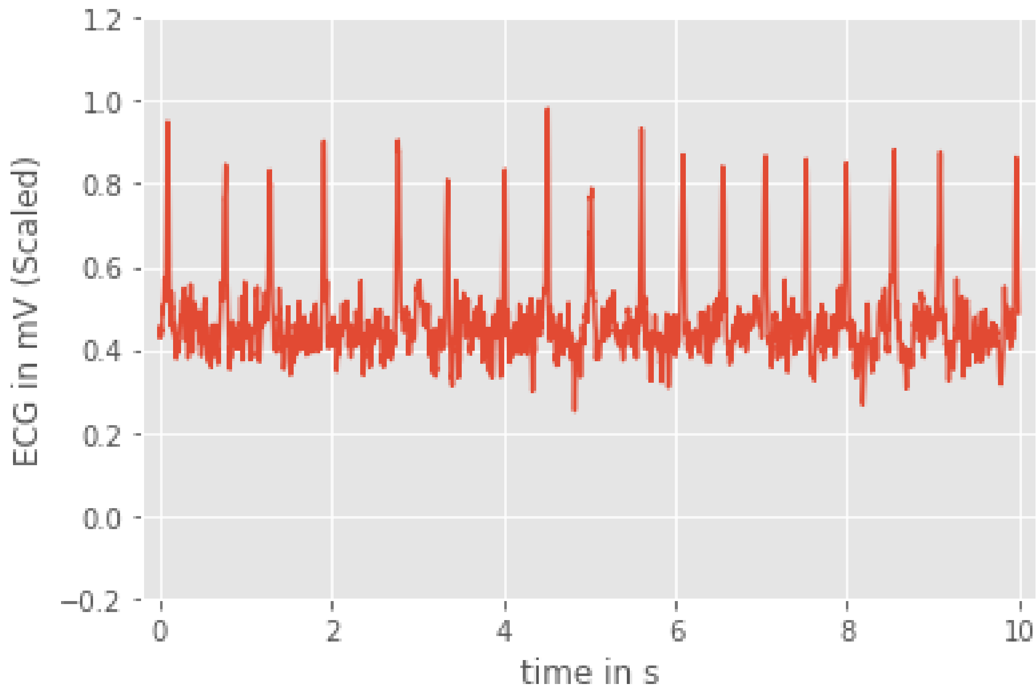
Here, lets visualize some of the signals present in our datasets.

[16]:
```
def visualize_ecg_signal(signal, label):
    label1 = labelencoder.inverse_transform([label])
    print("Label of signal: ", label1[0])
    time = np.arange(signal.size) / fs
    plt.plot(time, signal)
    plt.xlabel("time in s")
    plt.ylabel("ECG in mV (Scaled)")
    plt.xlim(-0.2, 10.2)
    plt.ylim(-0.2, 1.2)
    plt.show()
```

4.1 Plot 1

[17]:
```
visualize_ecg_signal(X_train[2], y_train[2])
```

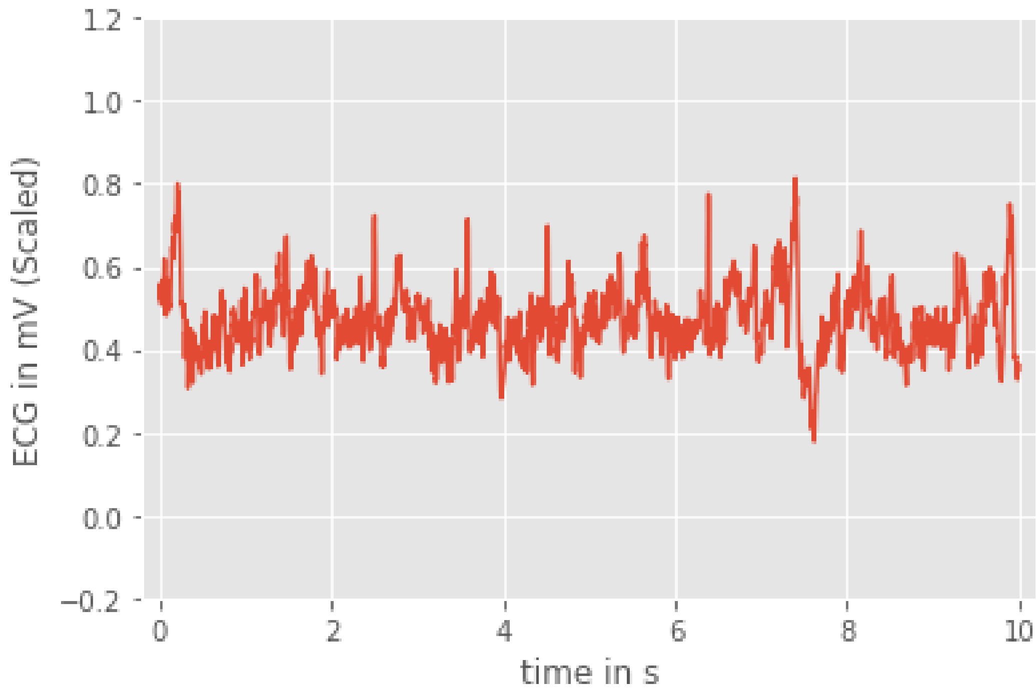
```
Label of signal:  AFIB
```



4.2 Plot 2

[18]:
```
visualize_ecg_signal(X_train[350], y_train[350])
```

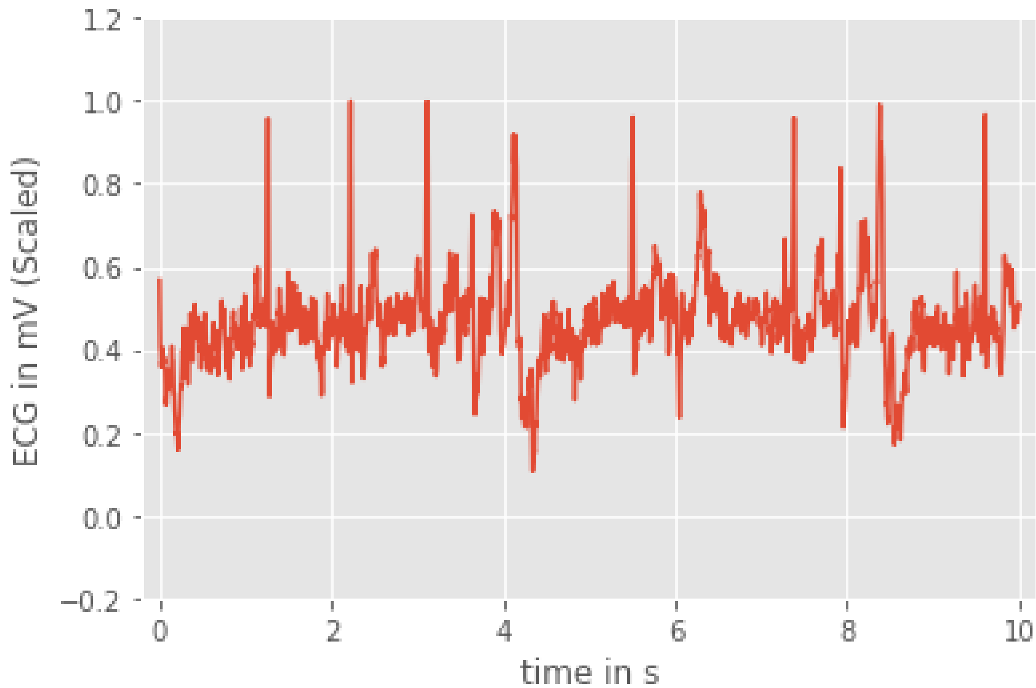
```
Label of signal:  VT
```



4.3 Plot 3

[19]:
```
visualize_ecg_signal(X_train[500], y_train[500])
```

```
Label of signal:  PVC
```



[20]:
```
# For plotting confusion matrices

def plot_confusion_matrix(data, labels):
    seaborn.set(color_codes=True)
    plt.figure(1, figsize=(60, 60))

    plt.title("Confusion Matrix")

    seaborn.set(font_scale=5)
    ax = seaborn.heatmap(data, annot=True, fmt='d', cmap="Blues",␣
 ↪cbar_kws={'label': 'Scale'})

    ax.set_xticklabels(labels, fontsize=40)
    ax.set_yticklabels(labels, fontsize=40)

    ax.set(ylabel="True Label", xlabel="Predicted Label")
```

5 Model 01: Linear discriminant analysis (LDA)

LDA is used to find a linear combination of features that characterizes or separates two or more classes of objects or events.

Lets define the model now.

[21]:
```python
# Defining the LDA model
model_LDA = LinearDiscriminantAnalysis()

# defining the model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# evaluating model
scores_training_LDA = cross_val_score(model_LDA, X_train, y_train,␣
 ↪scoring='accuracy', cv=cv, n_jobs=-1, verbose=1)

# fit model
model_LDA.fit(X_train, y_train)

# make a prediction
yhat = model_LDA.predict(X_test)
scores_testing_LDA = accuracy_score(y_test, yhat)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  1.1min finished
```

5.1 LDA Model Accuracy

[22]:
```python
# Printing the training and testing accuracy

print('Mean Training Accuracy: %.3f' % (mean(scores_training_LDA)*100))
print('Mean Testing Accuracy: %.3f' % (mean(scores_testing_LDA)*100))
```

```
Mean Training Accuracy: 71.450
Mean Testing Accuracy: 6.866
```

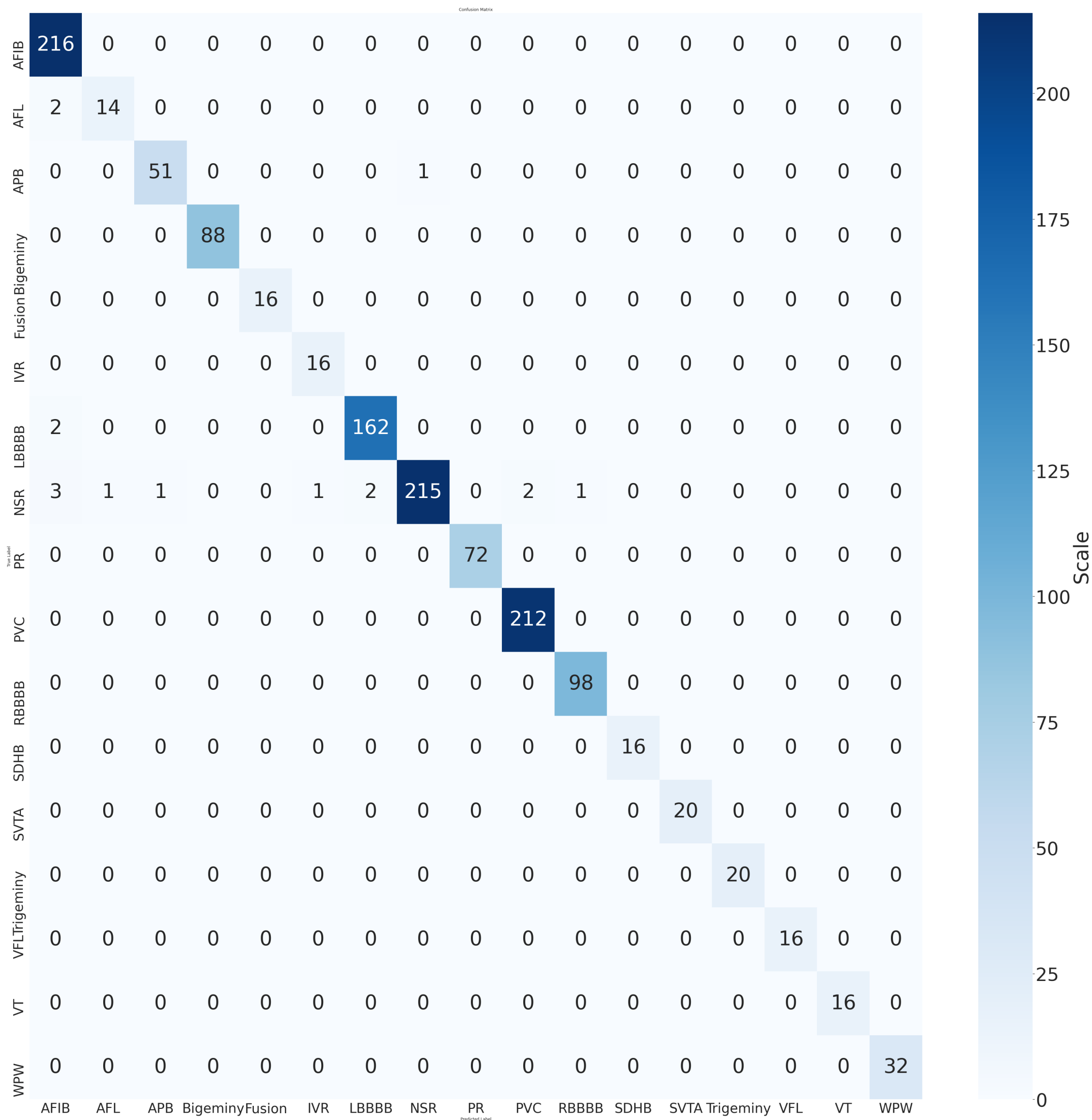
5.2 Confusion Matrix of Training data

[23]:
```python
yhat1 = model_LDA.predict(X_train)

cm = confusion_matrix(y_train, yhat1)

labels = [0, 1, 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
labels= labelencoder.inverse_transform(labels)

plot_confusion_matrix(cm, labels)
```



6 Model 02: Support Vector Machine

SVM can also be used to find features for classfication

Lets define the model now.

[24]:
```
# Defining the SVM model
model_SVC = SVC(kernel='poly', degree=3, C = 1)

# defining the model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
```

```
# evaluating model
scores_training_SVM = cross_val_score(model_SVC, X_train, y_train,␣
↪scoring='accuracy', cv=cv, n_jobs=-1, verbose=1)

# fit model
model_SVC.fit(X_train, y_train)

# make a prediction
yhat = model_SVC.predict(X_test)
scores_testing_SVM = accuracy_score(y_test, yhat)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  3.1min finished
```

6.1 SVM Model Accuracy

[25]:
```
# Printing the training and testing accuracy

print('Mean Training Accuracy: %.3f' % (mean(scores_training_SVM)*100))
print('Mean Testing Accuracy: %.3f' % (mean(scores_testing_SVM)*100))
```

```
Mean Training Accuracy: 81.378
Mean Testing Accuracy: 19.104
```

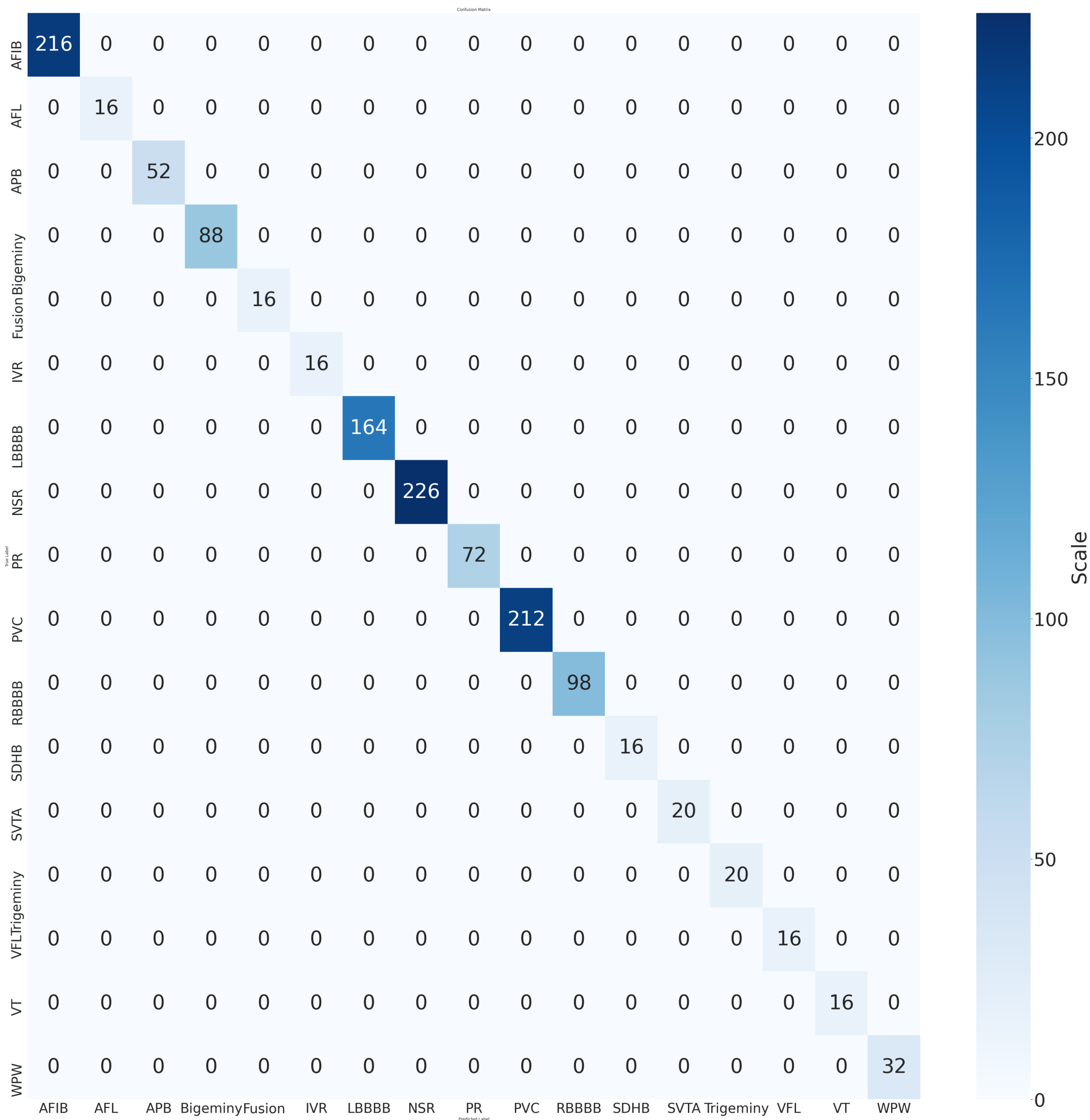
6.2 Confusion Matrix of Training data

[26]:
```
yhat1 = model_SVC.predict(X_train)

cm = confusion_matrix(y_train, yhat1)

labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
labels= labelencoder.inverse_transform(labels)

plot_confusion_matrix(cm, labels)
```



7 Model 3: Convolutional Neural Network (CNN)

Here, we will use Convolutional Neural Network (CNN) for classification.

First we have to reshape our input in following form:

- [batch_size, no_of_timestamps, no_of_channels]

Tensorflow can already infer batch size. No of time stamps in our case is 3600 and no of channel is 1. So our input shape will be [batch_size, 3600, 1]. We will also one hot encode our label vectors. This can be implemented as follows:

[27]:
```
print("Size of Previous X_train",  np.shape(X_train))
print("Size of Previous X_test",  np.shape(X_test))

print("\n")

X_train_n = np.reshape(X_train, [X_train.shape[0], X_train.shape[1], 1]).
 ↪astype(np.float64)
X_test_n = np.reshape(X_test, [X_test.shape[0], X_test.shape[1], 1]).astype(np.
 ↪float64)

y_train_n = tf.keras.utils.to_categorical(y_train, num_classes=K)
y_test_n = tf.keras.utils.to_categorical(y_test, num_classes=K)

print("Size of New X_train",  np.shape(X_train_n))
print("Size of New X_test",  np.shape(X_test_n))

print("\n")

print("Size of y_train",  np.shape(y_train_n))
print("Size of y_test",  np.shape(y_test_n))
```

```
Size of Previous X_train (1296, 3600)
Size of Previous X_test (335, 3600)


Size of New X_train (1296, 3600, 1)
Size of New X_test (335, 3600, 1)


Size of y_train (1296, 17)
Size of y_test (335, 17)
```

Now, we will define our CNN model architecture.

[47]:
```
verbose, epochs, batch_size = 1, 15, 32

n_timesteps, n_channels, n_outputs = X_train_n.shape[1], X_train_n.shape[2],␣
 ↪y_train_n.shape[1]

def CNN_model_arch(n_timesteps, n_channels, n_outputs):
    model = Sequential()
    model.add(Conv1D(filters=128, kernel_size=3, activation='relu',␣
 ↪input_shape=(n_timesteps,n_channels)))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
    model.add(Dropout(0.4))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2))
```

```
    model.add(Flatten())
    model.add(Dense(120, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    return model

CNN_model = CNN_model_arch(n_timesteps, n_channels, n_outputs)
CNN_model.summary()
CNN_model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_2 (Conv1D)            (None, 3598, 128)         512
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 3596, 64)          24640
_________________________________________________________________
dropout_2 (Dropout)          (None, 3596, 64)          0
_________________________________________________________________
batch_normalization_2 (Batch (None, 3596, 64)          256
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 1798, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 115072)            0
_________________________________________________________________
dense_4 (Dense)              (None, 120)               13808760
_________________________________________________________________
dense_5 (Dense)              (None, 17)                2057
=================================================================
Total params: 13,836,225
Trainable params: 13,836,097
Non-trainable params: 128
_________________________________________________________________
```

Lets train our model now

[48]:
```
history = CNN_model.fit(X_train_n, y_train_n, epochs=epochs,␣
 ↪validation_data=(X_test_n, y_test_n), batch_size=batch_size, verbose=verbose)

# Save model weights
CNN_model.save_weights("CNN_model_weights.hdf5")
```

```
Epoch 1/15
41/41 [==============================] - 1s 31ms/step - loss: 4.0607 - accuracy:
0.2323 - val_loss: 5.7561 - val_accuracy: 0.0776
Epoch 2/15
41/41 [==============================] - 1s 27ms/step - loss: 1.9126 - accuracy:
```

```
0.3750 - val_loss: 4.8706 - val_accuracy: 0.0776
Epoch 3/15
41/41 [==============================] - 1s 27ms/step - loss: 1.6706 - accuracy:
0.4568 - val_loss: 3.7003 - val_accuracy: 0.0776
Epoch 4/15
41/41 [==============================] - 1s 27ms/step - loss: 1.4579 - accuracy:
0.5185 - val_loss: 3.7862 - val_accuracy: 0.0478
Epoch 5/15
41/41 [==============================] - 1s 27ms/step - loss: 1.3245 - accuracy:
0.5725 - val_loss: 3.7848 - val_accuracy: 0.1433
Epoch 6/15
41/41 [==============================] - 1s 28ms/step - loss: 1.0900 - accuracy:
0.6605 - val_loss: 2.7263 - val_accuracy: 0.2090
Epoch 7/15
41/41 [==============================] - 1s 28ms/step - loss: 0.8808 - accuracy:
0.7230 - val_loss: 3.3610 - val_accuracy: 0.1075
Epoch 8/15
41/41 [==============================] - 1s 27ms/step - loss: 0.6596 - accuracy:
0.8009 - val_loss: 3.6593 - val_accuracy: 0.1343
Epoch 9/15
41/41 [==============================] - 1s 28ms/step - loss: 0.3741 - accuracy:
0.8974 - val_loss: 4.1894 - val_accuracy: 0.1582
Epoch 10/15
41/41 [==============================] - 1s 28ms/step - loss: 0.3000 - accuracy:
0.9236 - val_loss: 4.8176 - val_accuracy: 0.0955
Epoch 11/15
41/41 [==============================] - 1s 27ms/step - loss: 0.1521 - accuracy:
0.9730 - val_loss: 6.1038 - val_accuracy: 0.1522
Epoch 12/15
41/41 [==============================] - 1s 27ms/step - loss: 0.0941 - accuracy:
0.9877 - val_loss: 6.1856 - val_accuracy: 0.1254
Epoch 13/15
41/41 [==============================] - 1s 28ms/step - loss: 0.0640 - accuracy:
0.9892 - val_loss: 5.0515 - val_accuracy: 0.1284
Epoch 14/15
41/41 [==============================] - 1s 28ms/step - loss: 0.0428 - accuracy:
0.9954 - val_loss: 7.4674 - val_accuracy: 0.1403
Epoch 15/15
41/41 [==============================] - 1s 27ms/step - loss: 0.0635 - accuracy:
0.9884 - val_loss: 6.5152 - val_accuracy: 0.1642
```

Lets evaluate the trained model

[49]:
```
# Evaluating model
scores_training_CNN = CNN_model.evaluate(X_train_n, y_train_n, verbose=0)
scores_testing_CNN = CNN_model.evaluate(X_test_n, y_test_n, verbose=0)

print('Mean Training Accuracy: %.3f' % (mean(scores_training_CNN[1])*100))
```

```python
print('Mean Testing Accuracy: %.3f' % (mean(scores_testing_CNN[1])*100))
```

```
Mean Training Accuracy: 89.738
Mean Testing Accuracy: 16.418
```

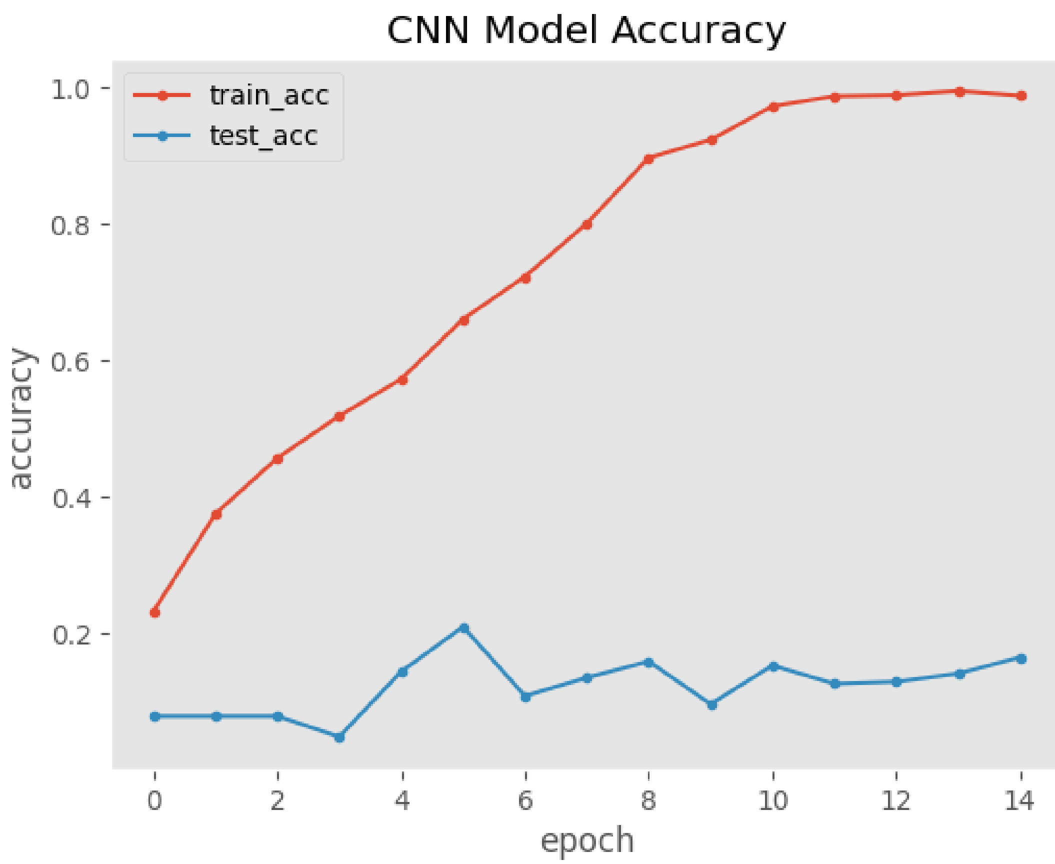
Now, lets plot the model training and validation accuracy plots

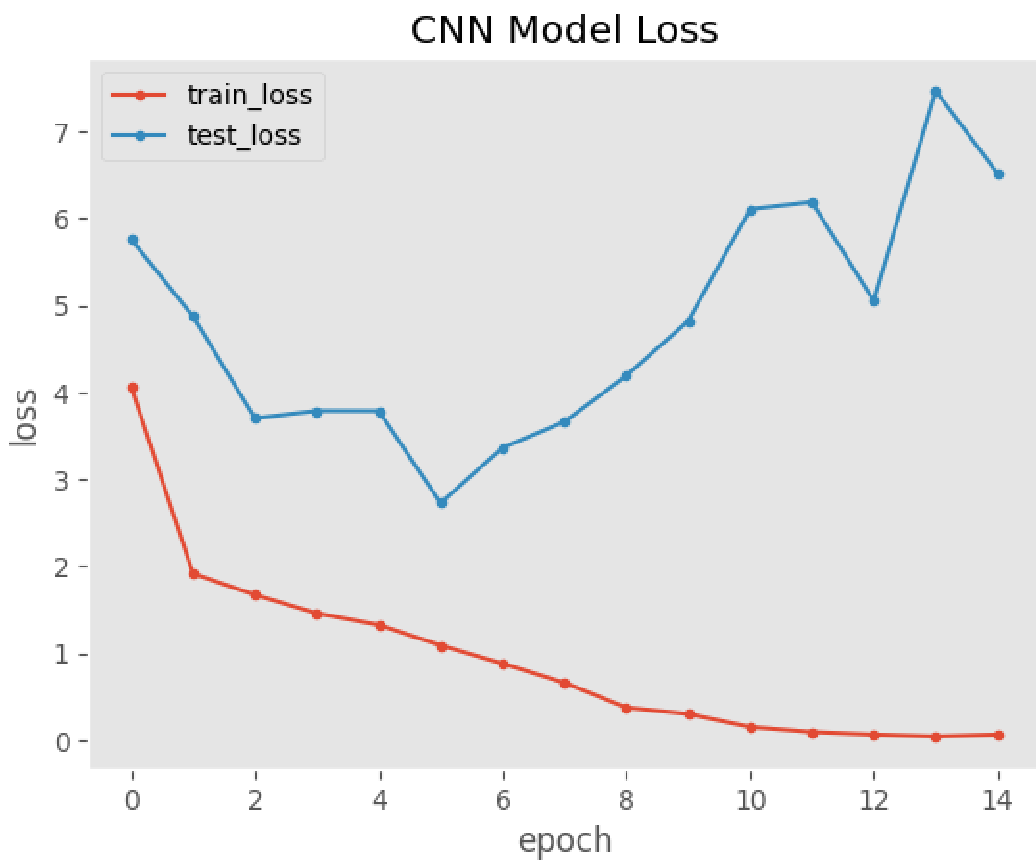
[50]:
```python
import matplotlib.style
import matplotlib as mpl
mpl.rcParams.update(mpl.rcParamsDefault)
mpl.style.use('ggplot')

# Plotting model accuracy
plt.plot(history.history['accuracy'], marker='.', label='train_acc')
plt.plot(history.history['val_accuracy'], marker='.', label='test_acc')
plt.title('CNN Model Accuracy')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(loc='best')
plt.show()

# Plotting model validation accuracy
plt.plot(history.history['loss'], marker='.', label='train_loss')
plt.plot(history.history['val_loss'], marker='.', label='test_loss')
plt.title('CNN Model Loss')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(loc='best')

plt.show()
```


CNN Model Accuracy
train_acc
test_acc
1.0
0.8
0.6
0.4
0.2
accuracy
0
2
4
6
8
10
12
14
epoch



8 Model 4: Long short-term memory cells (LSTMs)

Here, we will use LSTM for classification.

The data is already shaped into our desired format (as it was done for CNN model previously).

Now, we will define our LSTM model architecture.

[51]:
```
verbose, epochs, batch_size = 1, 15, 32

n_timesteps, n_channels, n_outputs = X_train_n.shape[1], X_train_n.shape[2],␣
↪y_train_n.shape[1]

def LSTM_model_arch(n_timesteps, n_channels, n_outputs):
    model = Sequential()
    model.add(LSTM(100, input_shape=(n_timesteps,n_channels)))
    model.add(Dropout(0.4))
    model.add(BatchNormalization())
    model.add(Dense(120, activation='relu'))
```

```
    model.add(Dense(n_outputs, activation='softmax'))
    return model

LSTM_model = LSTM_model_arch(n_timesteps, n_channels, n_outputs)
LSTM_model.summary()
LSTM_model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               40800
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 100)               400
_________________________________________________________________
dense_6 (Dense)              (None, 120)               12120
_________________________________________________________________
dense_7 (Dense)              (None, 17)                2057
=================================================================
Total params: 55,377
Trainable params: 55,177
Non-trainable params: 200
_________________________________________________________________
```

Lets train our model now

[52]:
```
history = LSTM_model.fit(X_train_n, y_train_n, epochs=epochs,␣
 ↪validation_data=(X_test_n, y_test_n), batch_size=batch_size, verbose=verbose)

# Save model weights
LSTM_model.save_weights("LSTM_model_weights.hdf5")
```

```
Epoch 1/15
41/41 [==============================] - 6s 138ms/step - loss: 2.6117 -
accuracy: 0.1528 - val_loss: 2.7220 - val_accuracy: 0.1313
Epoch 2/15
41/41 [==============================] - 5s 123ms/step - loss: 2.4095 -
accuracy: 0.1983 - val_loss: 2.6735 - val_accuracy: 0.1642
Epoch 3/15
41/41 [==============================] - 5s 123ms/step - loss: 2.3559 -
accuracy: 0.2130 - val_loss: 2.6021 - val_accuracy: 0.1791
Epoch 4/15
41/41 [==============================] - 5s 123ms/step - loss: 2.4369 -
accuracy: 0.1998 - val_loss: 3.4043 - val_accuracy: 0.1701
Epoch 5/15
```

```
41/41 [==============================] - 5s 123ms/step - loss: 2.5011 -
accuracy: 0.1605 - val_loss: 2.6495 - val_accuracy: 0.1433
Epoch 6/15
41/41 [==============================] - 5s 123ms/step - loss: 2.3642 -
accuracy: 0.2222 - val_loss: 2.5106 - val_accuracy: 0.1493
Epoch 7/15
41/41 [==============================] - 5s 124ms/step - loss: 2.3540 -
accuracy: 0.1983 - val_loss: 2.5306 - val_accuracy: 0.1254
Epoch 8/15
41/41 [==============================] - 5s 124ms/step - loss: 2.3447 -
accuracy: 0.2184 - val_loss: 2.5029 - val_accuracy: 0.1313
Epoch 9/15
41/41 [==============================] - 5s 124ms/step - loss: 2.3077 -
accuracy: 0.2191 - val_loss: 2.4828 - val_accuracy: 0.1761
Epoch 10/15
41/41 [==============================] - 5s 123ms/step - loss: 2.3201 -
accuracy: 0.2238 - val_loss: 2.5028 - val_accuracy: 0.1403
Epoch 11/15
41/41 [==============================] - 5s 123ms/step - loss: 2.3085 -
accuracy: 0.2122 - val_loss: 2.5001 - val_accuracy: 0.2000
Epoch 12/15
41/41 [==============================] - 5s 124ms/step - loss: 2.3075 -
accuracy: 0.2207 - val_loss: 2.4289 - val_accuracy: 0.2149
Epoch 13/15
41/41 [==============================] - 5s 124ms/step - loss: 2.2815 -
accuracy: 0.2153 - val_loss: 2.6155 - val_accuracy: 0.1403
Epoch 14/15
41/41 [==============================] - 5s 124ms/step - loss: 2.2853 -
accuracy: 0.2307 - val_loss: 2.3927 - val_accuracy: 0.2179
Epoch 15/15
41/41 [==============================] - 5s 123ms/step - loss: 2.2842 -
accuracy: 0.2384 - val_loss: 3.2655 - val_accuracy: 0.0925
```

Lets evaluate the trained model

[53]:
```
# Evaluating model
scores_training_LSTM = LSTM_model.evaluate(X_train_n, y_train_n, verbose=0)
scores_testing_LSTM = LSTM_model.evaluate(X_test_n, y_test_n, verbose=0)

print('Mean Training Accuracy: %.3f' % (mean(scores_training_LSTM[1])*100))
print('Mean Testing Accuracy: %.3f' % (mean(scores_testing_LSTM[1])*100))
```

```
Mean Training Accuracy: 9.105
Mean Testing Accuracy: 9.254
```

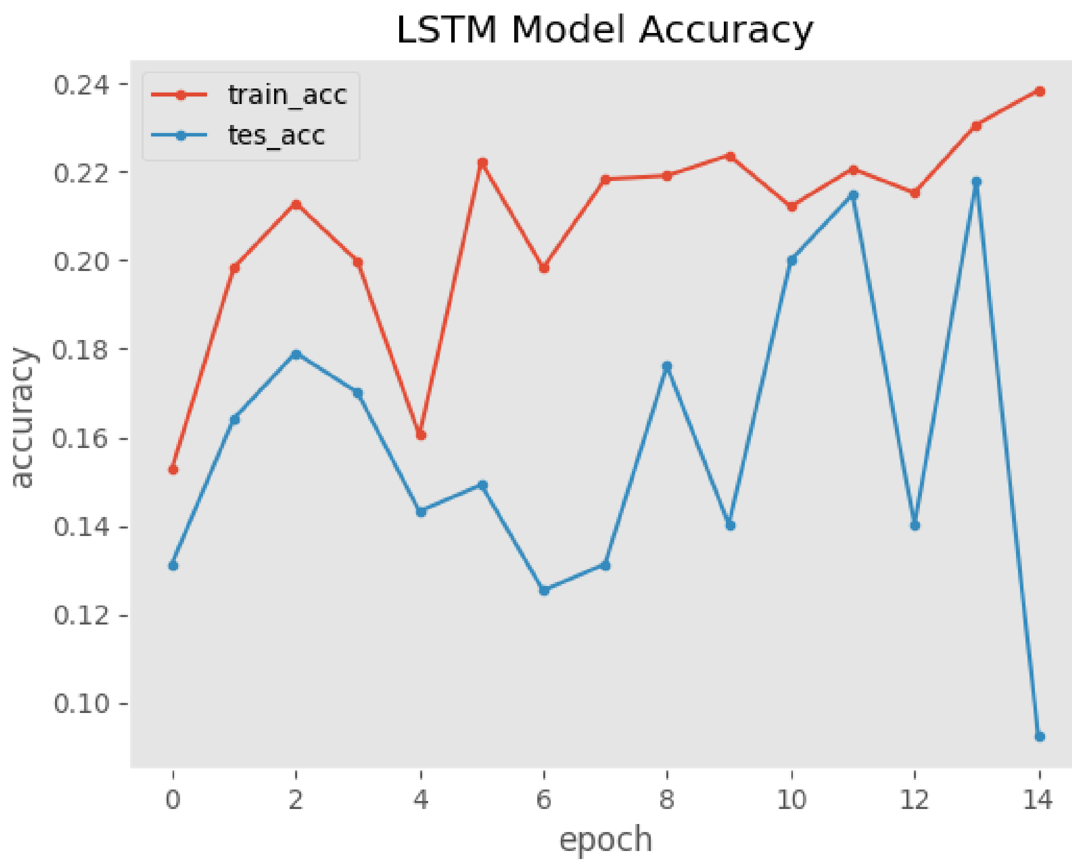
Now, lets plot the model training and validation accuracy plots

[54]:
```
# Plotting model accuracy
plt.plot(history.history['accuracy'], marker='.', label='train_acc')
```

```
plt.plot(history.history['val_accuracy'], marker='.', label='tes_acc')
plt.title('LSTM Model Accuracy')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(loc='best')
plt.show()

# Plotting model validation accuracy
plt.plot(history.history['loss'], marker='.', label='train_loss')
plt.plot(history.history['val_loss'], marker='.', label='test_loss')
plt.title('LSTM Model Loss')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(loc='best')

plt.show()
```



9 Conclusion and Summarized Results

In total, 4 supervised machine learning/deep learning models were trained and tested (LDA, SVM, CNN, LSTM). Genreally, the training accuracies were high on LDA, SVM, CNN and low on LSTM. I believe the deep learning models further has to be hyper tuned for achieving high accuracies on this dataset.

The results/accuracies are as follows:

[55]:
```
print("Summarized Results:\n")

print("1. LDA:")
print('Mean Training Accuracy: %.3f' % (mean(scores_training_LDA)*100))
print('Mean Testing Accuracy: %.3f' % (mean(scores_testing_LDA)*100))

print("\n")
print("2. SVM:")
print('Mean Training Accuracy: %.3f' % (mean(scores_training_SVM)*100))
```

```
print('Mean Testing Accuracy: %.3f' % (mean(scores_testing_SVM)*100))

print("\n")
print("3. CNN:")
print('Mean Training Accuracy: %.3f' % (mean(scores_training_CNN[1])*100))
print('Mean Testing Accuracy: %.3f' % (mean(scores_testing_CNN[1])*100))

print("\n")
print("4. LSTM:")
print('Mean Training Accuracy: %.3f' % (mean(scores_training_LSTM[1])*100))
print('Mean Testing Accuracy: %.3f' % (mean(scores_testing_LSTM[1])*100))
```

```
Summarized Results:

1. LDA:
Mean Training Accuracy: 71.450
Mean Testing Accuracy: 6.866


2. SVM:
Mean Training Accuracy: 81.378
Mean Testing Accuracy: 19.104


3. CNN:
Mean Training Accuracy: 89.738
Mean Testing Accuracy: 16.418


4. LSTM:
Mean Training Accuracy: 9.105
Mean Testing Accuracy: 9.254
```