COMP 5413 WB - Topics in Deep Learning

Final Project Report: Task 3 - EEG brain signal dataset for emotion prediction

Komal Barge	**(1095709)**
Devanshu Mishra	**(1104093)**

Contents:

Specifications:

Platform: Python using Keras and Tensorflow

Operating System: Google Colab

Runtime Environment: Google Colab GPU

Introduction:

Emotion is a cumulative expression of the consciousness and actions of humans. This essentially mirrors the experience and behavior of humans. In our daily lives emotion often plays a critical role, particularly in human-human interaction. In addition, emotional perception based on the computer system has also become a central part of the advanced brain-machine interaction systems. Therefore, emotion recognition has become a common subject in modern neuroscience, psychology, neural engineering, and computer science as well as its great significance and wide use.

Recognition of emotions can be split into two categories: one based on non-physiological signals and the other based on physiological signals. Some earlier emotion recognition experiments are focused on non-physiological cues, such as the identification of facial expression and voice-dependent emotion. Face expressions and tone of voice can also be intentionally obscured so that the method based on them is clearly not dependable. At the other side, the approach focused at physiological signals, referring to electroencephalography (EEG), electromyogram (EMG), electrocardiogram (ECG), skin resistance (SC), pulse rate and breathing signals, tend to be more efficient and accurate since people are unable to regulate them deliberately. In this project, we have aimed to predict to analyze how different portions of the human brain respond to different stimuli.

Data Description:

EEG dataset has been used for Emotion Recognition. 15 Chinese video clips (positive, neutral and negative emotions) were chosen from the collection of materials as stimuli used in the experiments. The selection criteria for movie clips are as follows:

(a) the length of the whole experiment should not be too long in case it will make subjects fatigue.

(b) the videos should be understood without explanation; and

(c) the videos should elicit a single desired target emotion.

The length of each video clip is about 4 minutes. Each video clip is well edited to create coherent emotion stimulating and maximize emotional meanings. There are totally 15 trials for each experiment. There is a 15s hint before each clips and 10s feedback after each clip. The order of presentation is arranged so that two movie clips targeting the same emotion are not shown consecutively. For the feedback, participants are told to report their emotional reactions to each film clip by completing the questionnaire immediately after watching each clip. A new effective EEG feature named differential entropy is proposed to represent the characteristics associated with emotional states.

In the "data" folder, there are train and test dataset containing down sampled with sampling frequency 200Hz in order to speed up the computation, preprocessed and segmented versions of the EEG differential entropy data.

The dataset containing extracted differential entropy (DE) features of the EEG signals. These data is well-suited to those who want to quickly test a classification method without processing the raw EEG data. The training set contains a total of 84420 data and testing set contains 58128 data. Each piece of data contains 310 values representing the data of 62 electrodes on 5 frequencies. These 310 values can be considered as the feature/attribute values for training a model.

Methods Used:

Data Augmentation:

The term data augmentation refers to methods for constructing iterative optimization or sampling algorithms via the introduction of unobserved data or latent variables. This can reduce overfitting of models that are fed with small datasets, because these do not generalize well from the validation and test set. Data augmentation means increasing the number of data points. In terms of images, it may mean increasing the number of images in the dataset. In terms of traditional row/column format data, it means increasing the number of rows or objects. Data augmentation is a method that artificially creates new data based on the modifications of the existing data. The heuristics underlying these modifications are very dependent on which processes are suitable for the classification task at issue.

1. **Data augmentation using Autoencoder**

   An autoencoder network is a pair of two connected networks, an encoder and a decoder. An encoder network takes in an input, and converts it into a smaller, dense representation, which the decoder network can use to convert it back to the original input. An encoder is a network that takes in an input and produces a much smaller representation (the encoding), that contains enough information for the next part of the network to process it into the desired output format. Typically, the encoder is trained together with the other parts of the network, optimized via back-propagation, to produce encodings specifically useful for the

task at hand. In CNNs, the encodings produced are such that they are specifically useful for classification. Autoencoders take this idea, and slightly flip it on its head, by making the encoder generate encodings specifically useful for reconstructing its own input.
The entire network is usually trained. The loss function is usually either the mean-squared error or cross-entropy between the output and the input, known as the reconstruction loss, which penalizes the network for creating outputs different from the input. As the encoding (which is simply the output of the hidden layer in the middle) has far less units than the input, the encoder must choose to discard information. The encoder learns to preserve as much of the relevant information as possible in the limited encoding, and intelligently discard irrelevant parts. The decoder learns to take the encoding and properly reconstruct it into a full image.

2. **Data augmentation using Standard Scaling**

The image can be scaled outward or inward. While scaling outward, the final image size will be larger than the original image size. Most image frameworks cut out a section from the new image, with size equal to the original image. We'll deal with scaling inward in the next section, as it reduces the image size, forcing us to make assumptions about what lies beyond the boundary.

Feature Extraction:

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is many variables that require a lot of computing resources to process. Feature extraction is the name for methods that select and /or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set.

The process of feature extraction is useful when you need to reduce the number of resources needed for processing without losing important or relevant information. Feature extraction can also reduce the amount of redundant data for a given analysis. Also, the reduction of the data and the machine's efforts in building variable combinations (features) facilitate the speed of learning and generalization steps in the machine learning process.

1. **Feature extraction using Linear Discriminant Analysis (LDA)**

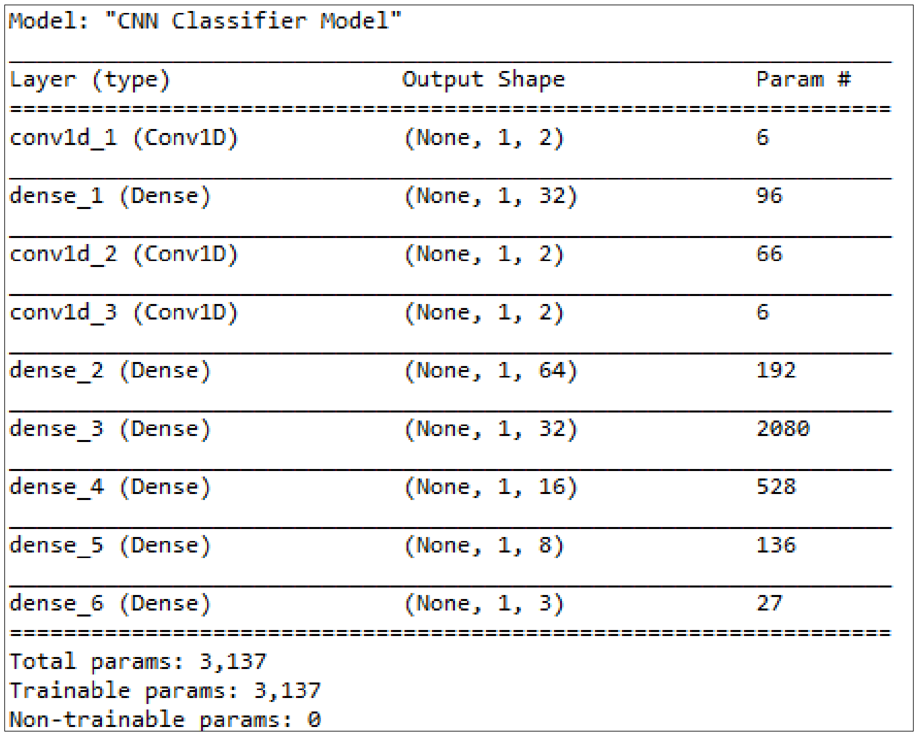
Linear discriminant analysis (LDA) is a traditional statistical technique that reduces dimensionality while preserving as much of the class discriminatory information as possible. The conventional form of the LDA assumes that all the data are available in advance and the LDA feature space is computed by finding the eigen decomposition of an appropriate matrix. However, there are situations where the data are presented in a sequence and the LDA features are required to be updated incrementally by observing the new incoming samples.

2. Feature extraction using Autoencoder

Feature extraction becomes increasingly important as data grows high dimensional. Autoencoder as a neural network based feature extraction method achieves great success in generating abstract features of high dimensional data. However, it fails to consider the relationships of data samples which may affect experimental results of using original and new features.

Proposed Model:

EEG dataset is already split into train and test folders when you download the dataset. But for applying the data augmentation technique on the given data, we have merged the train and test folders and then used **standard scaling** as our final **data augmentation** technique. This standard scaling has been used by centering the data before scaling, and it scaled the data to unit variance i.e. unit standard deviation.
After scaling the dataset, we performed the **feature extraction using LDA** method to reduce the dimensionality of the input by projecting it to the most discriminative directions. The number of components used for dimensionality reduction used is 2. We have split the dataset according to the requirement of the project. The **training set** contains a total of **84420** data and **testing set** contains **58128** data.
After extracting the features through dimension reduction, we forwarded it through the sequential convolutional model that we prepared which can be seen in Figure 1.

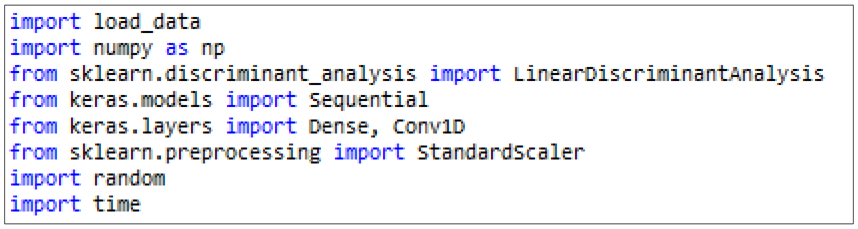


```
Model: "CNN Classifier Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 1, 2)              6
_________________________________________________________________
dense_1 (Dense)              (None, 1, 32)             96
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1, 2)              66
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 1, 2)              6
_________________________________________________________________
dense_2 (Dense)              (None, 1, 64)             192
_________________________________________________________________
dense_3 (Dense)              (None, 1, 32)             2080
_________________________________________________________________
dense_4 (Dense)              (None, 1, 16)             528
_________________________________________________________________
dense_5 (Dense)              (None, 1, 8)              136
_________________________________________________________________
dense_6 (Dense)              (None, 1, 3)              27
=================================================================
Total params: 3,137
Trainable params: 3,137
Non-trainable params: 0
```

Figure 1. Convolution Neural Network Classifier Model

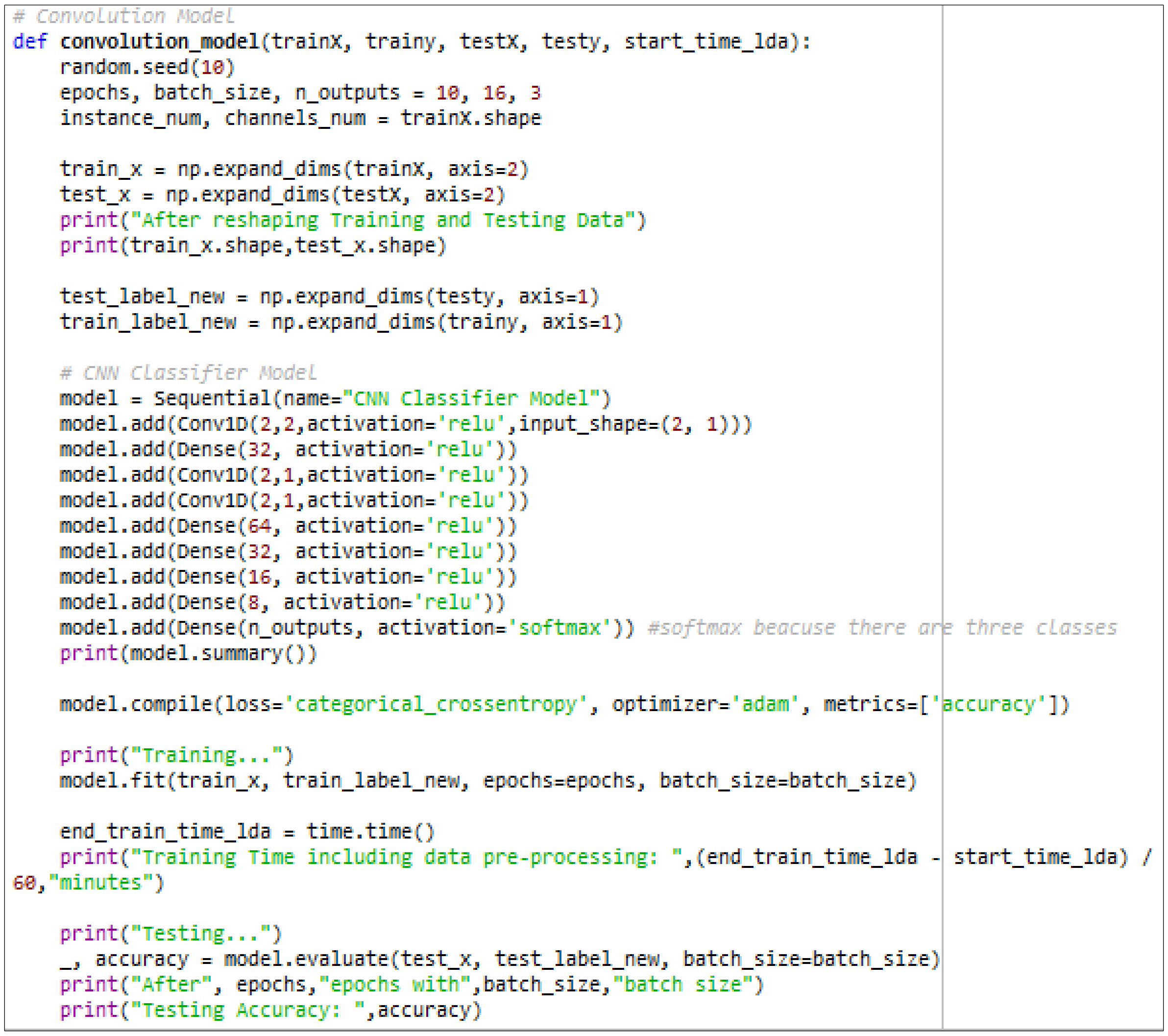
This model consists of 3 Convolutional 1D layers, 5 Dense layers with ReLu activation function. At the end one Dense Layer with Softmax activation layer. We passed the input to the convolutional 1D layer with kernel size 2x2. Then forwarded it to dense layer of hidden nodes 32. Then, applied the 2 convolutional with kernel 2x1 size. And forwarded it to the further 5 dense layers with neurons 64, 32, 16, 8 and 3 (number of outputs). Last layer is having the softmax activation function. We trained the model with '**categorical_crossentropy**' loss function and '**adam**' optimizer. We tried so many combinations for getting the highest accuracy of around **88.2087%**. We will discuss that in detail in discussions section and python code has attached in below section.

Python Code:



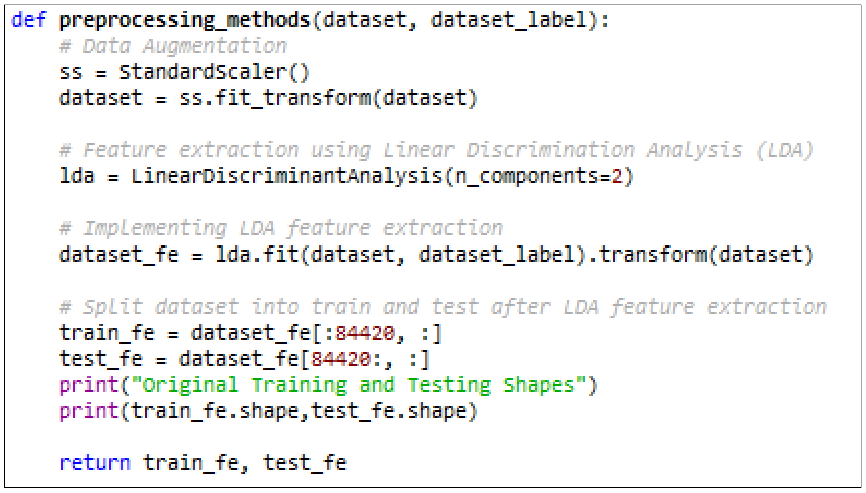
```
import load_data
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from keras.models import Sequential
from keras.layers import Dense, Conv1D
from sklearn.preprocessing import StandardScaler
import random
import time
```

Figure 2. Code Snippet (Libraries Used)



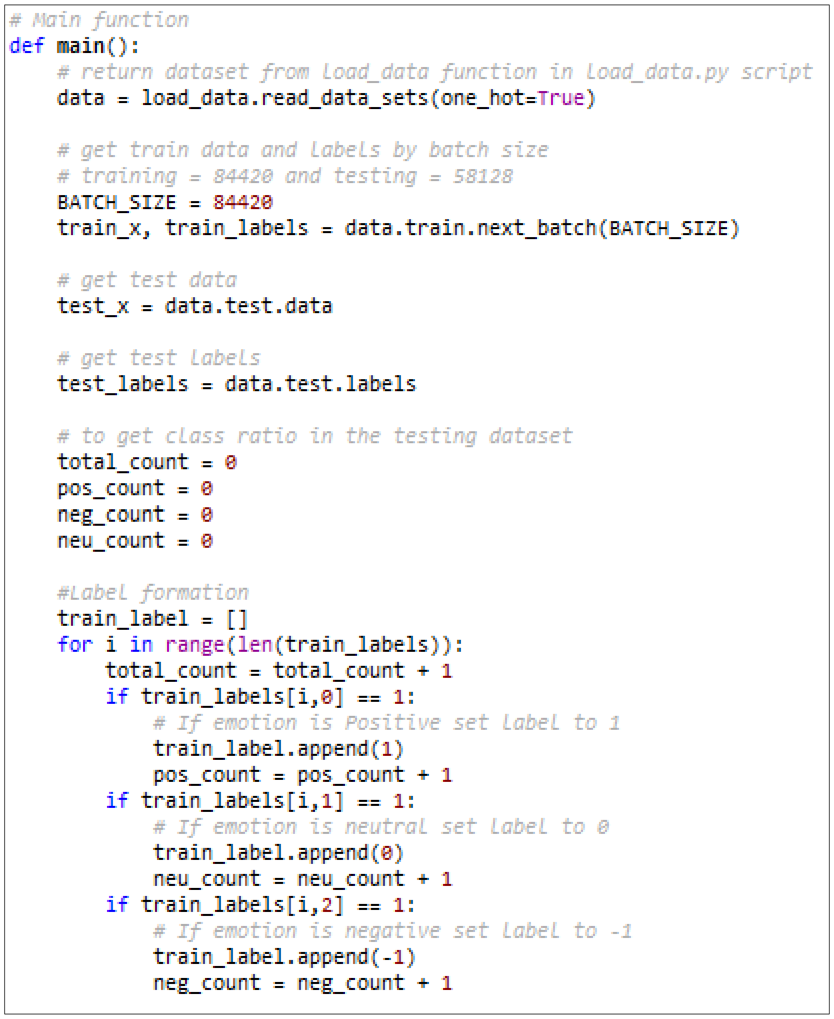
```python
# Convolution Model
def convolution_model(trainX, trainy, testX, testy, start_time_lda):
    random.seed(10)
    epochs, batch_size, n_outputs = 10, 16, 3
    instance_num, channels_num = trainX.shape

    train_x = np.expand_dims(trainX, axis=2)
    test_x = np.expand_dims(testX, axis=2)
    print("After reshaping Training and Testing Data")
    print(train_x.shape,test_x.shape)

    test_label_new = np.expand_dims(testy, axis=1)
    train_label_new = np.expand_dims(trainy, axis=1)

    # CNN Classifier Model
    model = Sequential(name="CNN Classifier Model")
    model.add(Conv1D(2,2,activation='relu',input_shape=(2, 1)))
    model.add(Dense(32, activation='relu'))
    model.add(Conv1D(2,1,activation='relu'))
    model.add(Conv1D(2,1,activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax')) #softmax beacuse there are three classes
    print(model.summary())

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

    print("Training...")
    model.fit(train_x, train_label_new, epochs=epochs, batch_size=batch_size)

    end_train_time_lda = time.time()
    print("Training Time including data pre-processing: ",(end_train_time_lda - start_time_lda) /
60,"minutes")

    print("Testing...")
    _, accuracy = model.evaluate(test_x, test_label_new, batch_size=batch_size)
    print("After", epochs,"epochs with",batch_size,"batch size")
    print("Testing Accuracy: ",accuracy)
```

Figure 3. Code Snippet (Classifier Function)



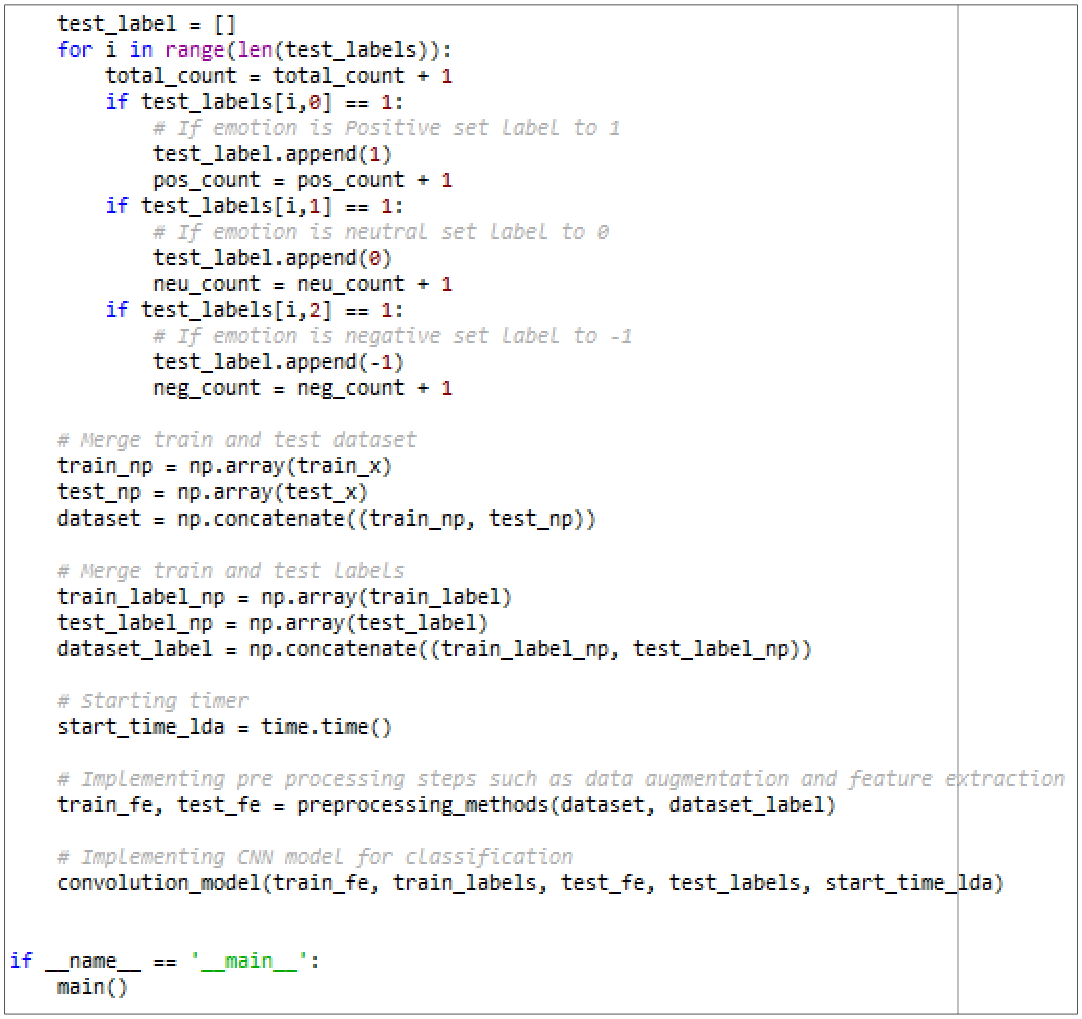
```python
def preprocessing_methods(dataset, dataset_label):
    # Data Augmentation
    ss = StandardScaler()
    dataset = ss.fit_transform(dataset)

    # Feature extraction using Linear Discrimination Analysis (LDA)
    lda = LinearDiscriminantAnalysis(n_components=2)

    # Implementing LDA feature extraction
    dataset_fe = lda.fit(dataset, dataset_label).transform(dataset)

    # Split dataset into train and test after LDA feature extraction
    train_fe = dataset_fe[:84420, :]
    test_fe = dataset_fe[84420:, :]
    print("Original Training and Testing Shapes")
    print(train_fe.shape,test_fe.shape)

    return train_fe, test_fe
```

Figure 4. Code Snippet (Pre-processing Function)



```python
# Main function
def main():
    # return dataset from load_data function in load_data.py script
    data = load_data.read_data_sets(one_hot=True)

    # get train data and labels by batch size
    # training = 84420 and testing = 58128
    BATCH_SIZE = 84420
    train_x, train_labels = data.train.next_batch(BATCH_SIZE)

    # get test data
    test_x = data.test.data

    # get test labels
    test_labels = data.test.labels

    # to get class ratio in the testing dataset
    total_count = 0
    pos_count = 0
    neg_count = 0
    neu_count = 0

    #Label formation
    train_label = []
    for i in range(len(train_labels)):
        total_count = total_count + 1
        if train_labels[i,0] == 1:
            # If emotion is Positive set label to 1
            train_label.append(1)
            pos_count = pos_count + 1
        if train_labels[i,1] == 1:
            # If emotion is neutral set label to 0
            train_label.append(0)
            neu_count = neu_count + 1
        if train_labels[i,2] == 1:
            # If emotion is negative set label to -1
            train_label.append(-1)
            neg_count = neg_count + 1
```

Figure 5. Code Snippet (Main Function - 1)



```
    test_label = []
    for i in range(len(test_labels)):
        total_count = total_count + 1
        if test_labels[i,0] == 1:
            # If emotion is Positive set label to 1
            test_label.append(1)
            pos_count = pos_count + 1
        if test_labels[i,1] == 1:
            # If emotion is neutral set label to 0
            test_label.append(0)
            neu_count = neu_count + 1
        if test_labels[i,2] == 1:
            # If emotion is negative set label to -1
            test_label.append(-1)
            neg_count = neg_count + 1

    # Merge train and test dataset
    train_np = np.array(train_x)
    test_np = np.array(test_x)
    dataset = np.concatenate((train_np, test_np))

    # Merge train and test labels
    train_label_np = np.array(train_label)
    test_label_np = np.array(test_label)
    dataset_label = np.concatenate((train_label_np, test_label_np))

    # Starting timer
    start_time_lda = time.time()

    # Implementing pre processing steps such as data augmentation and feature extraction
    train_fe, test_fe = preprocessing_methods(dataset, dataset_label)

    # Implementing CNN model for classification
    convolution_model(train_fe, train_labels, test_fe, test_labels, start_time_lda)


if __name__ == '__main__':
    main()
```

Figure 6. Code Snippet (Main Function - 2)

Results and Discussion:

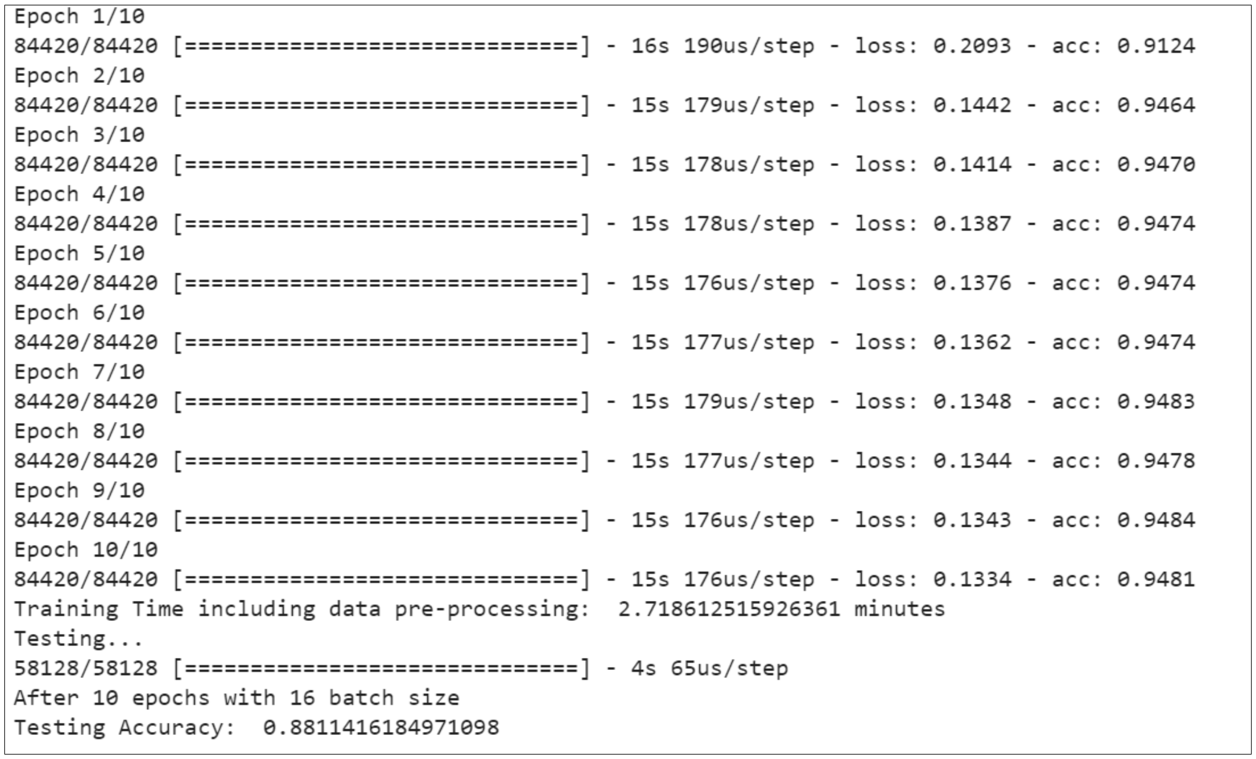
While performing this experiment on EEG dataset, we tried so many other models. We attempted data augmentation with autoencoder, feature extraction using Minimum Redundancy Maximum relevance (MRMR) method and classifiers like SVM and Random forest. You can see the combinations table in Table 1.

Data Augmentation	Feature Extraction	Classifier	Accuracy (After 3 runs)
Standard Scalar	MRMR	SVM	80.56%
Standard Scalar	LDA	Random Forest	84.88%
Autoencoder	Autoencoder	DCNN	86.31%
Standard Scalar	LDA	DCNN	88.21%

Table 1. Experiments around methods

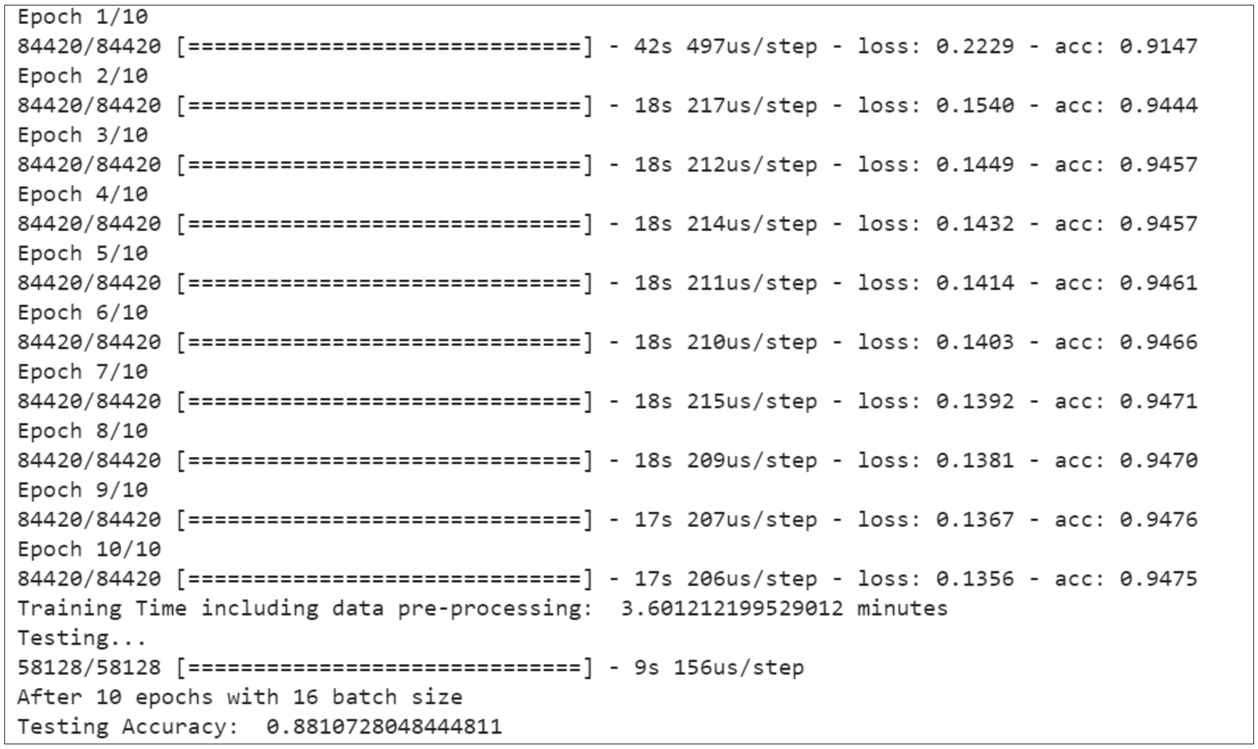
As from Table 1, DCNN gave us the high accuracy, we tried so many other combinations of the convolutional layers and dense layers which at the end with the proposed model gave use around **88.2087% accuracy**. The accuracy obtain is from **10 epochs** and data **batch size of 16**. Below are the output screenshots for 3 runs of DCNN. **Training time** for executing 10 epochs is **3.1 minutes** at an average.

Output Screenshots:



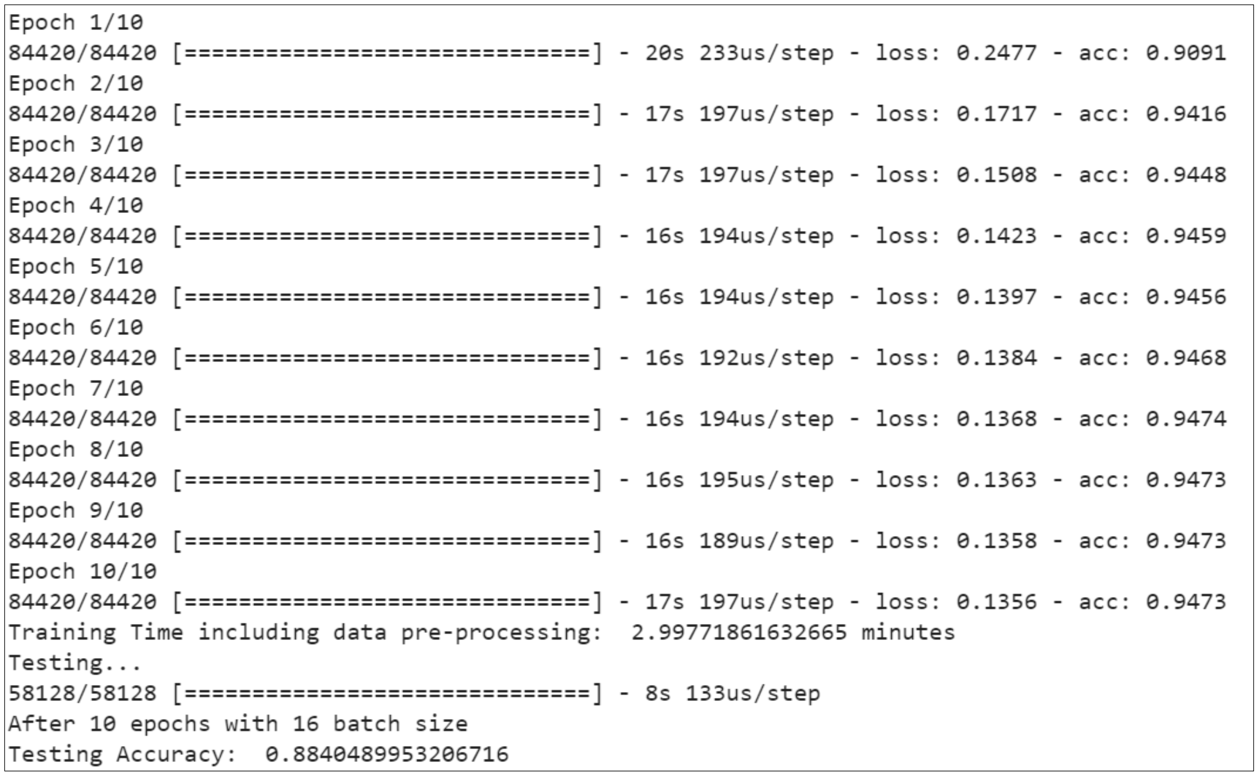
```
Epoch 1/10
84420/84420 [==============================] - 16s 190us/step - loss: 0.2093 - acc: 0.9124
Epoch 2/10
84420/84420 [==============================] - 15s 179us/step - loss: 0.1442 - acc: 0.9464
Epoch 3/10
84420/84420 [==============================] - 15s 178us/step - loss: 0.1414 - acc: 0.9470
Epoch 4/10
84420/84420 [==============================] - 15s 178us/step - loss: 0.1387 - acc: 0.9474
Epoch 5/10
84420/84420 [==============================] - 15s 176us/step - loss: 0.1376 - acc: 0.9474
Epoch 6/10
84420/84420 [==============================] - 15s 177us/step - loss: 0.1362 - acc: 0.9474
Epoch 7/10
84420/84420 [==============================] - 15s 179us/step - loss: 0.1348 - acc: 0.9483
Epoch 8/10
84420/84420 [==============================] - 15s 177us/step - loss: 0.1344 - acc: 0.9478
Epoch 9/10
84420/84420 [==============================] - 15s 176us/step - loss: 0.1343 - acc: 0.9484
Epoch 10/10
84420/84420 [==============================] - 15s 176us/step - loss: 0.1334 - acc: 0.9481
Training Time including data pre-processing:  2.718612515926361 minutes
Testing...
58128/58128 [==============================] - 4s 65us/step
After 10 epochs with 16 batch size
Testing Accuracy:  0.8811416184971098
```

Figure 7. 1st Run



```
Epoch 1/10
84420/84420 [==============================] - 42s 497us/step - loss: 0.2229 - acc: 0.9147
Epoch 2/10
84420/84420 [==============================] - 18s 217us/step - loss: 0.1540 - acc: 0.9444
Epoch 3/10
84420/84420 [==============================] - 18s 212us/step - loss: 0.1449 - acc: 0.9457
Epoch 4/10
84420/84420 [==============================] - 18s 214us/step - loss: 0.1432 - acc: 0.9457
Epoch 5/10
84420/84420 [==============================] - 18s 211us/step - loss: 0.1414 - acc: 0.9461
Epoch 6/10
84420/84420 [==============================] - 18s 210us/step - loss: 0.1403 - acc: 0.9466
Epoch 7/10
84420/84420 [==============================] - 18s 215us/step - loss: 0.1392 - acc: 0.9471
Epoch 8/10
84420/84420 [==============================] - 18s 209us/step - loss: 0.1381 - acc: 0.9470
Epoch 9/10
84420/84420 [==============================] - 17s 207us/step - loss: 0.1367 - acc: 0.9476
Epoch 10/10
84420/84420 [==============================] - 17s 206us/step - loss: 0.1356 - acc: 0.9475
Training Time including data pre-processing:  3.601212199529012 minutes
Testing...
58128/58128 [==============================] - 9s 156us/step
After 10 epochs with 16 batch size
Testing Accuracy:  0.8810728048444811
```

Figure 8. 2nd Run



```
Epoch 1/10
84420/84420 [==============================] - 20s 233us/step - loss: 0.2477 - acc: 0.9091
Epoch 2/10
84420/84420 [==============================] - 17s 197us/step - loss: 0.1717 - acc: 0.9416
Epoch 3/10
84420/84420 [==============================] - 17s 197us/step - loss: 0.1508 - acc: 0.9448
Epoch 4/10
84420/84420 [==============================] - 16s 194us/step - loss: 0.1423 - acc: 0.9459
Epoch 5/10
84420/84420 [==============================] - 16s 194us/step - loss: 0.1397 - acc: 0.9456
Epoch 6/10
84420/84420 [==============================] - 16s 192us/step - loss: 0.1384 - acc: 0.9468
Epoch 7/10
84420/84420 [==============================] - 16s 194us/step - loss: 0.1368 - acc: 0.9474
Epoch 8/10
84420/84420 [==============================] - 16s 195us/step - loss: 0.1363 - acc: 0.9473
Epoch 9/10
84420/84420 [==============================] - 16s 189us/step - loss: 0.1358 - acc: 0.9473
Epoch 10/10
84420/84420 [==============================] - 17s 197us/step - loss: 0.1356 - acc: 0.9473
Training Time including data pre-processing:  2.99771861632665 minutes
Testing...
58128/58128 [==============================] - 8s 133us/step
After 10 epochs with 16 batch size
Testing Accuracy:  0.8840489953206716
```

Figure 9. 3rd Run

Final Accuracy:

(88.1141 + 87.1072 + 88.4048) / 3 = 88.2087 %

Final Training Time:

(2.7186 + 3.6012 + 2.9977) / 3 = 3.1058 Minutes