Thesis Code (Python)

Installed pyeeg

```
pip install git+https://github.com/forrestbao/pyeeg.git
```

Mounted with google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Import necessary library packages

```
import numpy as np
import pyeeg as pe
import pickle as pickle
import pandas as pd
import math

from sklearn import svm
from sklearn.preprocessing import normalize

import os
#import tensorflow as tf
import time
```

Path set and define variable

```
channel = [1,2,3,4,6,11,13,17,19,20,21,25,29,31] #14 Channels chosen to fit Emotiv Epoch+
band = [4,8,12,16,25,45] #5 bands
window_size = 256 #Averaging band power of 2 sec
step_size = 16 #Each 0.125 sec update once
sample_rate = 128 #Sampling rate of 128 Hz

subjectList=[]
for i in range(1,33):
  if i<10:
     subjectList.append(f"{i:02d}")
  else:
    subjectList.append(f"{i:2d}")

print(subjectList)
```

```
#List of subjects
path_to_dataset_2 = '/content/drive/MyDrive/Thesis/DEAP-
dataset/Datasets/dat_File_Folder/'
path_to_dataset = '/content/drive/MyDrive/Thesis/DEAP-dataset/Datasets/model_5_Khosru/'
```

Plot raw EEG signal

```
dat_file_path = path_to_dataset_2 + "s01.dat"
s01_np = np.fromfile(dat_file_path, dtype='byte')

from matplotlib import pyplot as plt
plt.style.use('bmh')

t = np.arange(0, 100, 1)
EEG_s01 = s01_np[:100]

plt.figure()
plt.plot(t, EEG_s01, label="Raw EEG")
plt.xlabel("time")
plt.ylabel("amplitude")
plt.legend(loc="center left")
```

Feature extraction function define

```
def FFT_Processing (sub, channel, band, window_size, step_size, sample_rate):
    '''
    arguments:  string subject
                list channel indice
                list band
                int window size for FFT
                int step size for FFT
                int sample rate for FFT
    return:     void
    '''
    meta = []
    with open(path_to_dataset_2+'s' + sub + '.dat', 'rb') as file:

        subject = pickle.load(file, encoding='latin1') #resolve the python 2 data problem by enc
oding : latin1

        for i in range (0,40):
            # loop over 0-39 trails
            data = subject["data"][i]
            labels = subject["labels"][i]
            start = 0;
            while start + window_size < data.shape[1]:
                meta_array = []
                meta_data = [] #meta vector for analysis
                for j in channel:
                    X = data[j][start : start + window_size] #Slice raw data over 2 sec, at interval of
0.125 sec
                    Y = pe.bin_power(X, band, sample_rate) #FFT over 2 sec of channel j, in seq of
     theta, alpha, low beta, high beta, gamma
                    meta_data = meta_data + list(Y[0])

                meta_array.append(np.array(meta_data))
                meta_array.append(labels)

                meta.append(np.array(meta_array))
                start = start + step_size

        meta = np.array(meta)
        #np.save('C:/Users/faizan/Downloads/data_preprocessed_python/data_preprocessed_py
thon/s' + sub, meta, allow_pickle=True, fix_imports=True)
        np.save(path_to_dataset+'s' + sub, meta, allow_pickle=True, fix_imports=True)
```

Feature extraction function call

```
for subjects in subjectList:
    FFT_Processing (subjects, channel, band, window_size, step_size, sample_rate)
```

Modify the default parameters of np.load

```
import numpy as np
# save np.load
np_load_old = np.load

np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)
```

Splitting data into test set and training set

```
data_training = []
label_training = []
data_testing = []
label_testing = []

for subjects in subjectList:
 with open(path_to_dataset + 's' + subjects + '.npy', 'rb') as file:
    sub = np.load(file)
    for i in range (0,sub.shape[0]):
       if i % 8 == 0:
          data_testing.append(sub[i][0])
          label_testing.append(sub[i][1])
       else:
          data_training.append(sub[i][0])
          label_training.append(sub[i][1])
```

Restore np.load for future normal usage

```
np.load = np_load_old
```

Training and test data save in google drive

```
np.save(path_to_dataset + 'data_training', np.array(data_training), allow_pickle=True, fix_i
mports=True)
np.save(path_to_dataset + 'label_training', np.array(label_training), allow_pickle=True, fix_i
mports=True)
print("training dataset:", np.array(data_training).shape, np.array(label_training).shape)

np.save(path_to_dataset + 'data_testing', np.array(data_testing), allow_pickle=True, fix_imp
orts=True)
np.save(path_to_dataset + 'label_testing', np.array(label_testing), allow_pickle=True, fix_im
ports=True)
print("testing dataset:", np.array(data_testing).shape, np.array(label_testing).shape)
```

Training data and label load (arousal)

```
with open(path_to_dataset + 'data_training.npy', 'rb') as fileTrain:
X  = np.load(fileTrain)

with open(path_to_dataset + 'label_training.npy', 'rb') as fileTrainL:
Y  = np.load(fileTrainL)

X = normalize(X)
Z = np.ravel(Y[:, [0]])


Arousal_Train = np.ravel(Y[:, [0]])
Valence_Train = np.ravel(Y[:, [1]])
```

```
Domain_Train = np.ravel(Y[:, [2]])

Like_Train = np.ravel(Y[:, [3]])

for i in range(len(Z)):
if Z[i] == 9:
Z[i] = 8.99
```

Training data and label load (valence)

```
with open(path_to_dataset + 'data_training.npy', 'rb') as fileTrain:
X  = np.load(fileTrain)

with open(path_to_dataset + 'label_training.npy', 'rb') as fileTrainL:
Y  = np.load(fileTrainL)

X = normalize(X)
Z = np.ravel(Y[:, [1]])

Arousal_Train = np.ravel(Y[:, [0]])
Valence_Train = np.ravel(Y[:, [1]])
Domain_Train = np.ravel(Y[:, [2]])
Like_Train = np.ravel(Y[:, [3]])

for i in range(len(Z)):
if Z[i] == 9:
Z[i] = 8.99
```

Label value segmentation for binary classification (traning)

```
count_0 = 0
count_1 = 0
for i in range(len(Z)):
  if Z[i] >= 1 and Z[i]<=4.99:
    Z [i] = 0
    count_0 = count_0 + 1
  else:
    Z [i] = 1
    count_1 = count_1 + 1
print(count_0,count_1)
```

Import necessary some another libraries

```
import pandas as pd
import keras.backend as K
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.models import Sequential
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
#from keras.utils import to_categorical
from keras.utils.np_utils import to_categorical
from keras.layers import Flatten
from keras.layers import Dense
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras import backend as K
from keras.models import Model
import timeit
from keras.models import Sequential
from keras.layers.core import Flatten, Dense, Dropout
from keras.layers.convolutional import Convolution1D, MaxPooling1D, ZeroPadding1D
from keras.optimizers import SGD
#import cv2, numpy as np
import warnings
warnings.filterwarnings('ignore')
```

Label data categorized (training)

```
from keras.utils.np_utils import to_categorical
y_train = to_categorical(Z)
y_train = y_train[:,1:]
y_train[:10]
```

Train data convert into numpy array format

```
x_train = np.array(X[:])
```

Tasting data and label load (arousal)

```
with open(path_to_dataset + 'data_testing.npy', 'rb') as fileTrain:
    M  = np.load(fileTrain)
```

```
with open(path_to_dataset + 'label_testing.npy', 'rb') as fileTrainL:
    N  = np.load(fileTrainL)

M = normalize(M)
L = np.ravel(N[:, [0]])

Arousal_Test = np.ravel(N[:, [0]])
Valence_Test = np.ravel(N[:, [1]])
Domain_Test = np.ravel(N[:, [2]])
Like_Test = np.ravel(N[:, [3]])

for i in range(len(L)):
  if L[i] == 9:
    L[i] = 8.99
```

Tasting data and label load (valence)

```
with open(path_to_dataset + 'data_testing.npy', 'rb') as fileTrain:
    M  = np.load(fileTrain)

with open(path_to_dataset + 'label_testing.npy', 'rb') as fileTrainL:
    N  = np.load(fileTrainL)

M = normalize(M)
L = np.ravel(N[:, [1]])

Arousal_Test = np.ravel(N[:, [0]])
Valence_Test = np.ravel(N[:, [1]])
Domain_Test = np.ravel(N[:, [2]])
Like_Test = np.ravel(N[:, [3]])

for i in range(len(L)):
  if L[i] == 9:
    L[i] = 8.99
```

Label value segmentation for binary classification (testing)

```
count_0 = 0
count_1 = 0
for i in range(len(L)):
  if L[i] >= 1 and L[i]<=4.99:
    L [i] = 0
    count_0 = count_0 + 1
  else:
    L [i] = 1
    count_1 = count_1 + 1
print(count_0,count_1)
```

Test data convert into numpy array format

```
x_test = np.array(M[:])
x_test
```

Label data categorized (testing)

```
from keras.utils.np_utils import to_categorical
y_test = to_categorical(L)
y_test = y_test[:,1:]
y_test[:10]
```

Fit the train and test data with StandarScaler

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
```

Reshape train and test data

```
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1], 1)
```

Model parameter define

```
batch_size = 256
num_classes = 8
epochs = 120
input_shape=(x_train.shape[1], 1)
```

Import another library packages

```
from keras.layers import Convolution1D, ZeroPadding1D, MaxPooling1D, BatchNormalizat
ion, Activation, Dropout, Flatten, Dense
from keras.regularizers import l2
```

Model architecture-1

```
model = Sequential()
intput_shape=(x_train.shape[1], 1)
model.add(Conv1D(128, kernel_size=3,padding = 'same',activation='relu', input_shape=inpu
t_shape))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
model.add(Conv1D(128,kernel_size=3,padding = 'same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
#model.add(Conv1D(64,kernel_size=3,padding = 'same', activation='relu'))
#model.add(MaxPooling1D(pool_size=(2)))
model.add(Flatten())
model.add(Dense(64, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(32, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Model architecture-2

```
model = Sequential()
intput_shape=(x_train.shape[1], 1)
model.add(Conv1D(256, kernel_size=3,padding = 'same',activation='relu', input_shape=inpu
t_shape))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
model.add(Conv1D(128,kernel_size=3,padding = 'same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
model.add(Conv1D(64,kernel_size=3,padding = 'same', activation='relu'))
model.add(MaxPooling1D(pool_size=(2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Model architecture-3

```
model = Sequential()
```

```
intput_shape=(x_train.shape[1], 1)
model.add(Conv1D(128, kernel_size=3,padding = 'same',activation='relu', input_shape=inpu
t_shape))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
model.add(Conv1D(64,kernel_size=3,padding = 'same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
model.add(Conv1D(32,kernel_size=3,padding = 'same', activation='relu'))
model.add(MaxPooling1D(pool_size=(2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Model architecture-4

```
model = Sequential()
intput_shape=(x_train.shape[1], 1)
model.add(Conv1D(512, kernel_size=9,padding = 'same',activation='relu', input_shape=inpu
t_shape))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
model.add(Conv1D(128,kernel_size=6,padding = 'same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
model.add(Conv1D(32,kernel_size=3,padding = 'same', activation='relu'))
model.add(MaxPooling1D(pool_size=(2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Model architecture-5

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(num_classes, activation='hard_sigmoid'))

initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=100000,
    decay_rate=0.96,
    staircase=True)
```

Model architecture-6

```
model = Sequential()

intput_shape=(x_train.shape[1], 1)

model.add(Conv1D(1024, kernel_size=9,padding = 'same',activation='relu', input_shape=inp
ut_shape))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))

model.add(Conv1D(512,kernel_size=6,padding = 'same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))

model.add(Conv1D(256,kernel_size=6,padding = 'same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))

model.add(Conv1D(128,kernel_size=6,padding = 'same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))

model.add(Flatten())

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Compile the model

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

Plot model architecture specification

```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

Save checkpoint details

```
# check points and early stopping
from keras.callbacks import ModelCheckpoint,EarlyStopping
model_name = ""
filepath="/content/drive/MyDrive/Thesis/DEAP-dataset/Saved_checkpoints_2/Copy_Valance_Check_point_2/"  + model_name + "weights-improvement-{epoch:02d}-{accuracy:.4f}.hdf5"
print(filepath)
checkpoint = ModelCheckpoint(filepath, monitor='accuracy', verbose=1, save_best_only=True, mode='max')
es = EarlyStopping(monitor='accuracy', mode='max', verbose=1, patience=15)
callbacks_list = [es, checkpoint]
```

Fit the model

```
H = model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        callbacks= callbacks_list)
```

Find the accuracy after completing the model train

```
score = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Manual save

```
base_path_model = "/content/drive/My Drive/Google_Colab/Autoencoder/saved_models/"
accuracy = "_89"
path_model = base_path_model + model_name + accuracy + ".h5"
model.save(path_model)
```

Find the accuracy from a save point

```
from keras.models import load_model
model_loaded = load_model('/content/drive/MyDrive/Thesis/DEAP-
dataset/Saved_checkpoints_2/Copy_Valance_Check_point_2/weights-improvement-120-
0.9857.hdf5')
score = model_loaded.evaluate(x_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Training loss graph

```
N = num_classes
EPOCS = 120
# construct a plot that plots and saves the training history
import matplotlib.pyplot as plt
N = np.arange(0, EPOCS)
plt.style.use("ggplot")
plt.figure()
plt.plot(N, H.history["loss"], label="train_loss")
#plt.plot(N, H.history["val_loss"], label="val_loss")
plt.title("Training Loss Graph")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="upper left")
plt.show()
```

Training accuracy graph

```
N = num_classes
EPOCS = 120
# construct a plot that plots and saves the training history
import matplotlib.pyplot as plt
N = np.arange(0, EPOCS)
plt.style.use("bmh")
plt.figure()
plt.plot(N, H.history["accuracy"], label="Train_Accuracy")
#plt.plot(N, H.history["val_loss"], label="val_loss")
plt.title("Training Accuracy Graph")
plt.xlabel("Epoch #")
plt.ylabel("Accuracy")
plt.legend(loc="lower left")
plt.show()
```

Training Loss & Training Accuracy graph

```
# summarize history for loss
plt.plot(H.history['loss'])
plt.plot(H.history['accuracy'])
plt.title('Training Loss & Training Accuracy graph')
plt.ylabel('loss/accuracy')
plt.xlabel('epoch')
plt.legend(['loss', 'accuracy'], loc='upper right')
plt.show()
```

Confusion matrix plotting function

```
from sklearn.utils.multiclass import unique_labels
from matplotlib import pyplot as plt
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax

np.set_printoptions(precision=2)
```

Confusion matrix plotting function call

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (30,10)
plt.rcParams.update({'font.size': 27})

from sklearn.metrics import confusion_matrix

y_pred = model_loaded.predict(x_test)
y_test_argmax = y_test.argmax(axis=1)
y_pred_argmax = y_pred.argmax(axis=1)
```

```
class_names = np.array([0,1, 2, 3, 4, 5, 6, 7])

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test_argmax, y_pred_argmax, classes=class_names,
                      title='Confusion matrix, without normalization')
# Plot normalized confusion matrix
plot_confusion_matrix(y_test_argmax, y_pred_argmax, classes=class_names, normalize=Tr
ue,
                      title='Valence')
#fig = plt.figure(figsize=(20 ,20), dpi=300)
plt.xlabel('Predicted label', fontsize=25)
plt.show()
#plt.figure(figsize=(40, 40))
```

Classification report

```
y_true = np.array(y_test)

y_pred = np.squeeze(model.predict(x_test))
y_pred = np.array(y_pred >= 0.5, dtype=np.int)

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Load model from a save point

```
from keras.models import load_model
model_loaded = load_model('/content/drive/MyDrive/Thesis/DEAP-
dataset/Saved_checkpoints_2/Copy_Valance_Check_point_2/weights-improvement-120-
0.9857.hdf5')

H = model_loaded.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_split = 0.2,
          callbacks= callbacks_list)
```

Test accuracy from a saving point

```
from keras.models import load_model
```

```
model_loaded = load_model('/content/drive/MyDrive/Thesis/DEAP-
dataset/Saved_checkpoints_2/Copy_Valance_Check_point_2/weights-improvement-115-
0.9854.hdf5')
score = model_loaded.evaluate(x_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```