LAMPIRAN

Lampiran 1. *Source Code* FFT-KNN

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

from google.colab import drive
drive.mount('/content/gdrive')
import zipfile
import numpy as np
import librosa
import math
import pyeeg
import joblib
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from scipy import signal
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from scipy.fft import fft
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv1D, Conv2D, MaxPool2D, Flatten, Dense,
InputLayer, BatchNormalization, Dropout, Activation, AveragePooling1D
from keras.optimizers import Adam, SGD
from keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
from scipy import stats
from scipy.stats import zscore
from sklearn.metrics import accuracy_score

# LOAD DATA
# ================================================================
archive = zipfile.ZipFile('gdrive/My Drive/TESIS/dataset.zip', 'r')
file = archive.open('dataset.joblib')

channel=14
signal_length=256

dataset=joblib.load(file)
data=dataset[0]
label=dataset[1]
data=np.array(data)
label=np.array(label)

print("Loaded")
# ================================================================

# BALANCED
# ================================================================
data=data.reshape(int(len(data)*channel), signal_length)
label=np.repeat(label, channel)

# # cutter_num=88718
cutter_num=560
data=data[label>=0]
```

```
label=label[label>=0]

data0=data[label==0]
data1=data[label==1]
data2=data[label==2]
data3=data[label==3]
data4=data[label==4]
data5=data[label==5]
data6=data[label==6]
data7=data[label==7]
data8=data[label==8]
data9=data[label==9]
label0=label[label==0]
label1=label[label==1]
label2=label[label==2]
label3=label[label==3]
label4=label[label==4]
label5=label[label==5]
label6=label[label==6]
label7=label[label==7]
label8=label[label==8]
label9=label[label==9]

data0=data0[0:cutter_num,:]
data1=data1[0:cutter_num,:]
data2=data2[0:cutter_num,:]
data3=data3[0:cutter_num,:]
data4=data4[0:cutter_num,:]
data5=data5[0:cutter_num,:]
data6=data6[0:cutter_num,:]
data7=data7[0:cutter_num,:]
data8=data8[0:cutter_num,:]
data9=data9[0:cutter_num,:]
label0=label0[0:cutter_num]
label1=label1[0:cutter_num]
label2=label2[0:cutter_num]
label3=label3[0:cutter_num]
label4=label4[0:cutter_num]
label5=label5[0:cutter_num]
label6=label6[0:cutter_num]
label7=label7[0:cutter_num]
label8=label8[0:cutter_num]
label9=label9[0:cutter_num]

data=np.concatenate((data0, data1, data2, data3, data4, data5, data6,
data7, data8, data9), axis=0)
label=np.concatenate((label0, label1, label2, label3, label4, label5,
label6, label7, label8, label9), axis=0)
# label=label+1
print("Balanced")
# ==============================================================================

# FFT
# ==============================================================================
N=256
data_fft=[]
for d in data:
  # d=librosa.util.fix_length(d, int(signal_length/2))
  y = fft(d)
  data_fft.append(np.abs(y[1:N//2]))

data=np.array(data_fft)
print("FFT Done")
# ==============================================================================
# print(data.shape)
```

```
# FLATTENING
# ================================================================
data=data.reshape(int(data.shape[0]/channel), data.shape[1]*channel)
label=label[::channel]
print("Flattened")
# ================================================================
k_select=[3,5,7,9]
seeds = [13, 51, 137, 24659, 347]
i=1
for i_s, seed in enumerate(seeds):
  print("train : 70% vs test : 30%")
  sss = StratifiedShuffleSplit(n_splits=2, test_size=0.3,
random_state=seed)
  sss.get_n_splits(data, label)

  # print(k)
  print("============================================")
  print("Iteration : "+str(i))
  i=i+1
  print("============================================")
  for train_index, test_index in sss.split(data, label):
      print("TRAIN:", len(train_index), "TEST:", len(test_index))
      X_train, X_test = data[train_index], data[test_index]
      y_train, y_test = label[train_index], label[test_index]

      # SCALING
      # ============================================================
      scaler = MinMaxScaler(feature_range=(0,1))
      scaler.fit(X_train)
      X_train=scaler.transform(X_train)
      X_test=scaler.transform(X_test)
      # print("Scaled")
      # # ============================================================

      for k in k_select:
        print("K="+str(k))
        # print("============================================")
        neigh = KNeighborsClassifier(n_neighbors=k)
        # neigh=KNeighborsClassifier(metric='wminkowski', p=2,
        #                            metric_params={'w':
np.random.random(X_train.shape[1])})
        neigh.fit(X_train, y_train)

        y_pred = neigh.predict(X_train)
        # print(confusion_matrix(y_train, y_pred))
        # print(classification_report(y_train, y_pred))

        print("Train Acc : "+str(accuracy_score(y_train, y_pred)))

        y_pred = neigh.predict(X_test)

        print(confusion_matrix(y_test, y_pred))
        print(classification_report(y_test, y_pred))
        print("Test Acc : "+str(accuracy_score(y_test, y_pred)))

        # print("============================================================")
```

Lampiran 2. *Source Code* CNN

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Activation, Permute, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
AveragePooling2D
from tensorflow.keras.layers import SeparableConv2D, DepthwiseConv2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import SpatialDropout2D
from tensorflow.keras.regularizers import l1_l2
from tensorflow.keras.layers import Input, Flatten
from tensorflow.keras.constraints import max_norm
from tensorflow.keras import backend as K
from keras.models import load_model
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

def DeepConvNet(nb_classes, Chans = 14, Samples = 256,
                dropoutRate = 0.5):
    """ Keras implementation of the Deep Convolutional Network as
described in
    Schirrmeister et. al. (2017), Human Brain Mapping.

    This implementation assumes the input is a 2-second EEG signal
sampled at
    128Hz, as opposed to signals sampled at 250Hz as described in the
original
    paper. We also perform temporal convolutions of length (1, 5) as
opposed
    to (1, 10) due to this sampling rate difference.

    Note that we use the max_norm constraint on all convolutional
layers, as
    well as the classification layer. We also change the defaults for
the
    BatchNormalization layer. We used this based on a personal
communication
    with the original authors.

                      ours        original paper
    pool_size        1, 2        1, 3
    strides          1, 2        1, 3
    conv filters     1, 5        1, 10

    Note that this implementation has not been verified by the original
    authors.

    """

    # start the model
    input_main   = Input((Chans, Samples, 1))
    block1       = Conv2D(25, (1, 5),
                                 input_shape=(Chans, Samples, 1),
                                 kernel_constraint = max_norm(2.,
axis=(0,1,2)))(input_main)
    block1       = Conv2D(25, (Chans, 1),
                                 kernel_constraint = max_norm(2.,
axis=(0,1,2)))(block1)
    block1       = BatchNormalization(epsilon=1e-05,
momentum=0.1)(block1)
    block1       = Activation('elu')(block1)
```

```
    block1       = MaxPooling2D(pool_size=(1, 2), strides=(1,
2))(block1)
    block1       = Dropout(dropoutRate)(block1)

    block2       = Conv2D(50, (1, 5),
                                  kernel_constraint = max_norm2.,
axis=(0,1,2)))(block1)
    block2       = BatchNormalization(epsilon=1e-05,
momentum=0.1)(block2)
    block2       = Activation('elu')(block2)
    block2       = MaxPooling2D(pool_size=(1, 2), strides=(1,
2))(block2)
    block2       = Dropout(dropoutRate)(block2)

    block3       = Conv2D(100, (1, 5),
                                  kernel_constraint = max_norm2.,
axis=(0,1,2)))(block2)
    block3       = BatchNormalization(epsilon=1e-05,
momentum=0.1)(block3)
    block3       = Activation('elu')(block3)
    block3       = MaxPooling2D(pool_size=(1, 2), strides=(1,
2))(block3)
    block3       = Dropout(dropoutRate)(block3)

    block4       = Conv2D(200, (1, 5),
                                  kernel_constraint = max_norm2.,
axis=(0,1,2)))(block3)
    block4       = BatchNormalization(epsilon=1e-05,
momentum=0.1)(block4)
    block4       = Activation('elu')(block4)
    block4       = MaxPooling2D(pool_size=(1, 2), strides=(1,
2))(block4)
    block4       = Dropout(dropoutRate)(block4)

    flatten      = Flatten()(block4)

    dense        = Dense(nb_classes, kernel_constraint =
max_norm(0.5))(flatten)
    softmax      = Activation('softmax')(dense)

    return Model(inputs=input_main, outputs=softmax)

!pip install joblib
!pip install keras_hist_graph

%tensorflow_version 2.x
from google.colab import drive
drive.mount('/content/gdrive')
import zipfile
import numpy as np
import librosa
import math
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import StratifiedShuffleSplit
import joblib
from keras.models import load_model
from keras.callbacks import ModelCheckpoint, EarlyStopping
import os.path
from os import path
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

print(tf.__version__)
print(tf.test.gpu_device_name())
```

```
print("Num GPUs Available: ",
len(tf.config.experimental.list_physical_devices('GPU')))

# LOAD DATA
# ============================================================================
archive = zipfile.ZipFile('gdrive/My Drive/TESIS/dataset.zip', 'r')
file = archive.open('dataset.joblib')

channel=14
signal_length=256

dataset=joblib.load(file)
data=dataset[0]
label=dataset[1]
data=np.array(data)
label=np.array(label)
print("Loaded")
# ============================================================================

# BALANCED
# ============================================================================
data=data.reshape(int(len(data)*channel), signal_length)
label=np.repeat(label, channel)

# cutter_num=88718
cutter_num=560
data=data[label>=0]
label=label[label>=0]

# data_1=data[label==-1]
data0=data[label==0]
data1=data[label==1]
data2=data[label==2]
data3=data[label==3]
data4=data[label==4]
data5=data[label==5]
data6=data[label==6]
data7=data[label==7]
data8=data[label==8]
data9=data[label==9]
# label_1=label[label==-1]
label0=label[label==0]
label1=label[label==1]
label2=label[label==2]
label3=label[label==3]
label4=label[label==4]
label5=label[label==5]
label6=label[label==6]
label7=label[label==7]
label8=label[label==8]
label9=label[label==9]

# data_1=data_1[0:cutter_num,:]
data0=data0[0:cutter_num,:]
data1=data1[0:cutter_num,:]
data2=data2[0:cutter_num,:]
data3=data3[0:cutter_num,:]
data4=data4[0:cutter_num,:]
data5=data5[0:cutter_num,:]
data6=data6[0:cutter_num,:]
data7=data7[0:cutter_num,:]
data8=data8[0:cutter_num,:]
data9=data9[0:cutter_num,:]
# label_1=label_1[0:cutter_num]
label0=label0[0:cutter_num]
```

```
label1=label1[0:cutter_num]
label2=label2[0:cutter_num]
label3=label3[0:cutter_num]
label4=label4[0:cutter_num]
label5=label5[0:cutter_num]
label6=label6[0:cutter_num]
label7=label7[0:cutter_num]
label8=label8[0:cutter_num]
label9=label9[0:cutter_num]

data=np.concatenate((data0, data1, data2, data3, data4, data5, data6,
data7, data8, data9), axis=0)
label=np.concatenate((label0, label1, label2, label3, label4, label5,
label6, label7, label8, label9), axis=0)
# label=label+1
print("Balanced")
# ================================================================

# FLATTENING
# ================================================================
data=data.reshape(int(len(data)/channel), signal_length*channel)
label=label[::channel]
print("Flattened")
# ================================================================

# # SCALING
#================================================================
# scaler = MinMaxScaler(feature_range=(0,1))
# scaler.fit(data)
# data=scaler.transform(data)
# print("Scaled")
#================================================================

# define the checkpoint
filepath = F"/content/gdrive/My Drive/TESIS/cnn_test.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=0,
save_best_only=True, mode='min')
# es = EarlyStopping(monitor='val_acc', mode='min', verbose=0,
patience=20)
callbacks_list = [checkpoint]

model=DeepConvNet(10)
model.summary()

k_select=[3,5,7,9]
seeds = [13, 51, 137, 24659, 347]
i=1
for i_s, seed in enumerate(seeds):
  print("train : 70% vs test : 30%")
  sss = StratifiedShuffleSplit(n_splits=2, test_size=0.3,
random_state=seed)
  sss.get_n_splits(data, label)

  # print(k)
  print("============================================")
  print("Iteration : "+str(i))
  i=i+1
  print("============================================")
  for train_index, test_index in sss.split(data, label):
    X_train, X_test = data[train_index], data[test_index]
    y_train, y_test = label[train_index], label[test_index]

    # SCALING
    # ================================================================
    scaler = MinMaxScaler(feature_range=(0,1))
```

```
    scaler.fit(X_train)
    X_train=scaler.transform(X_train)
    X_test=scaler.transform(X_test)
    # print("Scaled")
    #================================================================

    X_train=X_train.reshape(len(X_train), 14, 256,1)
    X_test=X_test.reshape(len(X_test), 14, 256, 1)

    model=DeepConvNet(10)
    model.compile(
        optimizer=tf.keras.optimizers.Adam(0.001),
        loss=tf.keras.losses.sparse_categorical_crossentropy,
        metrics=['acc']
    )

    history=model.fit(X_train, y_train, batch_size=32, epochs=1000,
validation_split=0.3, verbose=0, callbacks=callbacks_list)

    model=load_model(filepath)

    y_pred=model.predict(X_train)

    y_pred=np.argmax(y_pred,axis=1)

    print("Train Acc : "+str(accuracy_score(y_train, y_pred)))

    y_pred=model.predict(X_test)

    y_pred=np.argmax(y_pred,axis=1)

    print("Test Acc : "+str(accuracy_score(y_test, y_pred)))

    print(classification_report(y_test, y_pred))

    plt.figure(figsize=(10, 8))

    # summarize history for accuracy

    plt.subplot(211)
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Training', 'Validation'], loc='lower right')

    # summarize history for loss

    plt.subplot(212)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Training', 'Validation'], loc='upper right')

    plt.tight_layout()

    plt.show()
```