MCMASTER UNIVERSITY

Modelling of Biological Systems
ELEC ENG 4BC3/6BC3

# Project Report: Epileptic Seizure Detection based on EEG and CNN

Instructor:
Dr. Aleksandar Jeremic

Muhammad Haris, Shiyue Zhang, Sijie Zhang

November 28, 2020

## 1 Introduction

A seizure is a sudden alteration of behavior due to a temporary change in the electrical functioning of the brain. There are different types of seizures, some affect a part of the brain, which is referred to as focal seizures and others affect the whole brain, which is called generalized seizures. Epilepsy is a central nervous system (neurological) disorder in which brain activity becomes abnormal, causing seizures or periods of unusual behavior, sensations, and sometimes loss of awareness. In epilepsy the brain's electrical rhythms have a tendency to become imbalanced, resulting in recurrent seizures. Such seizures can result in situation based accidents if the activity before seizure required atmost focus, like driving a car. Hence, extreme situations like these result in fatal injuries.

There are three phases to seizures: (a) preictal; one hour before a seizure, (b) interictal; between two seizures and (c) ictal; during seizure. If these phases are detected the patient can be alerted about the seizure in advance and hence possibly preventing a fatal injury in the first place. Electroencephalography (EEG) is a method to measure voltage changes between electrodes on a subject's scalp produced by ionic changes in the brain. It provides temporal and spatial information about the subject's brain. Detection of seizures using EEG is possible but mostly requires a direct examination by a physician hence we suggest a automatic computer aided algorithm to detect seizure without any manual intervention whilst diagnosis.

This paper briefly talks about the deep learning model used for detecting the three phases of an epileptic seizure using a large data set. The paper is organised in the following order section 2 will talk about previous attempts at machine learning and deep learning to detect and/or categorise seizures, section 3 talks briefly about the data set used, section 4 talks about methodology used, section 5 lists down the findings, section 6 interprets and analyses the results, and section 8 discusses future directions. All the relevant code can be found in appendix section 9.

## 2 Previous Work

There has been a significant progress in the domain of detecting seizures and/or the three phases of a seizure, using deep learning. There has been a significant development in the field of deep learning especially after we have significant computational power in terms of both speed and memory. Cloud computation has made the situation even more favourable. Prior to this conventional machine learning was used to interpret EEG. However, conventional machine learning was not sufficient for multichannel EEG hence deep learning [1, 2]. Furthermore, conventional machine learning did not perform well on raw EEG signals therefore, the phase of feature extraction was still being done manually. Deep Learning algorithms, like Convolutional Neural Network, extracts and learns features automatically.

CNNs are widely used in computer vision for various image related applications. In [4] a very basic yet elegant approach is employed, inspired from computer vision principles, of using plots as images to detect seizures just like a human would do visually. Apart from image as inputs the actual EEG data can be input to the deep learning framework. In [3] authors use an improved RBF algorithm to dynamically extract features. However, now that deep learning is being used feature extraction does not need to be performed manually. In [1, 2], authors use simple convolutional neural network frameworks for training followed by detecting the three phases of seizures.

Authors in [1] argue that EEG signals in frequency domain provide more insights on the data in order for the CNN to extract features. This paper will compare the performance of architectures on time and frequency domain signals based on performance.

## 3 Data

The database used in this study was an open-source EEG database from CHB-MIT. The recordings were collected from 23 children with epilepsy using scalp electrodes, and EEG data were provided by the Massachusetts Institute of Technology (MIT, USA). The study included 17 females that ranged in age from 1.5 to 19 years and five males that ranged in age from 3 to 22 years. The last patient's name, age and sex is not known in the dataset. All subjects were asked to stop related treatments 1 week before data collection. The sampling frequency for all patients was 256Hz. The seizure start and end times were labeled explicitly based on expert judgments, and the number and durations of seizure events varied for each subject. Except six patients all the others contain readings corresponding to exactly one hour of time. Majority of the files contain 23 channel EEG, we discarded the few exceptions where they were 24 or 26 channeled readings, since our architecture was not flexible in terms of input size. There were a total of 664 files containing the readings and a total of 182 seizures.

## 4 Proposed Methodology

The paper [1] that was followed proposed the architecture shown in figure 1. The architecture is quite minimalist and simple hence does not take up a lot of RAM. The major concern during the implementation is to save as much RAM as possible. In order to do so the entire data was not read, rather python generators were to be used to read and then feed the data in batches to the framework.

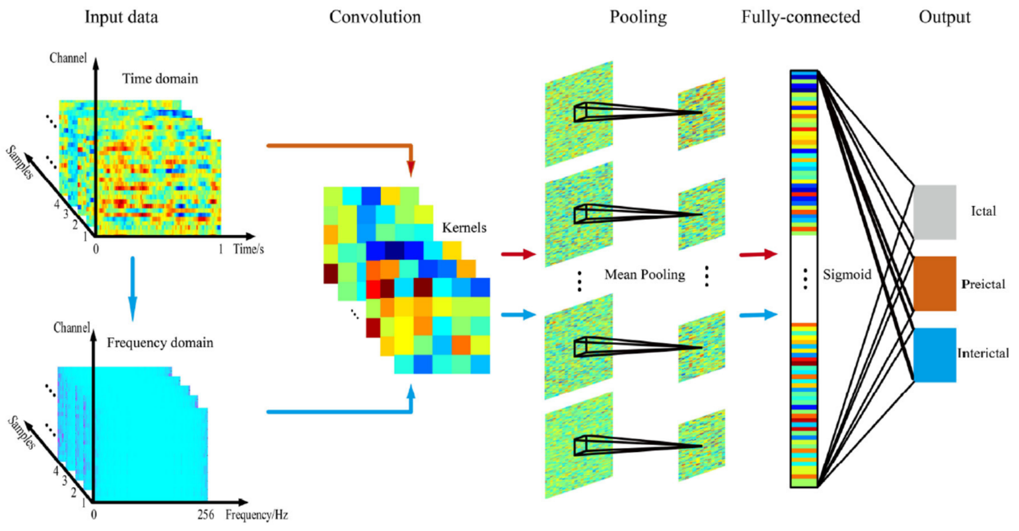

Figure 1: CNN architecture proposed in [1].

The proposed methods takes two main approaches, one for frequency domain and the other for time domain. The input to the architecture is a series of 2D signals, having 23 channels against 256 samples per second. The data is divided into training and validation with a 66%-33% split, and this is further divided into batches such that the RAM does not collapse due to insufficient memory. The batches are then divided such that each instance has 3 $23 \times 256$ 2D signals. The first layer is the convolutional layer where $5 \times 5$ kernels are trained. This layer deals with training the weights of the kernel such that when a signal is passed through this kernel and the subsequent layers the output corresponds to the labels. This kernel works exactly like a filter when convolved with the EEG signal with appropriate stride value (one in this case), which are commonly used to smooth out or extract derievatives (edge information) of a 2D signal. Followed by pooling, for time domain, and mean pooling for frequency domain. Pooling layers generally added to reduce the number of hyper-parameters and hence reducing overfitting. A simple pooling layer combines the outputs of neuron clusters in one layer into a single neuron in the next layer, on the other hand the mean pooling uses the average value from each cluster of neurons in the previous layer. The final layer is a fully connected layer, which connects every neuron in one layer to every neuron in another layer, whose output is passed through a non-linear activation function, sigmoid. We tried other variations where we experimented with interchanging the pooling layers, but the results were not satisfactory enough to be shared. The number of epochs were fixed at 5. The training-validation data split was kept at 66%-33%. Another architecture, shown in figure 2, was tried however, the RAM crashed before it could learn on the entire dataset. Due to limited memory offered by colab we were not able to add additional layers.

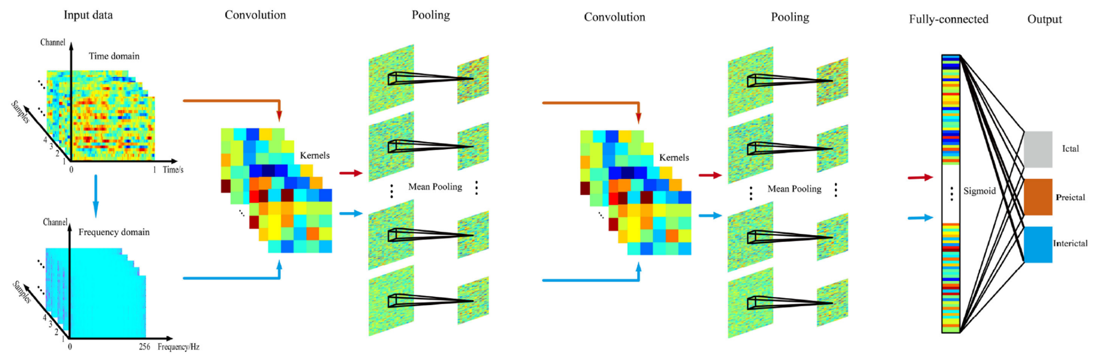

Figure 2: Modified CNN architecture

In order to evaluate performance of this non-binary categorization, accuracy is used as a mesaure. This paper defines accuracy as shown in equation (2), where $f(y_i)$ is an indicator function giving 1 incase of correct prediction and 0 when missclassifed as shown in (1).

$$f(y_i) = \begin{cases} 1 & y_i == \hat{y}_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$acc = \frac{1}{n}\sum_{i=1}^{n} f(y_i) \tag{2}$$

The opposite of this metric is the loss metric defined in equation (4).

$$g(y_i) = \begin{cases} 0 & y_i == \hat{y}_i \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

$$loss = \frac{1}{n}\sum_{i=1}^{n} g(y_i) \tag{4}$$

Where $y_i$ is predicted by the model and $\hat{y}_i$ is label on the data, the ground truth.

## 5 Results

Plots in figure 3 show training and validation accuracies and losses for frequency domain architecture. Plots in figure 4 show training and validation accuracies and losses for time domain architecture.

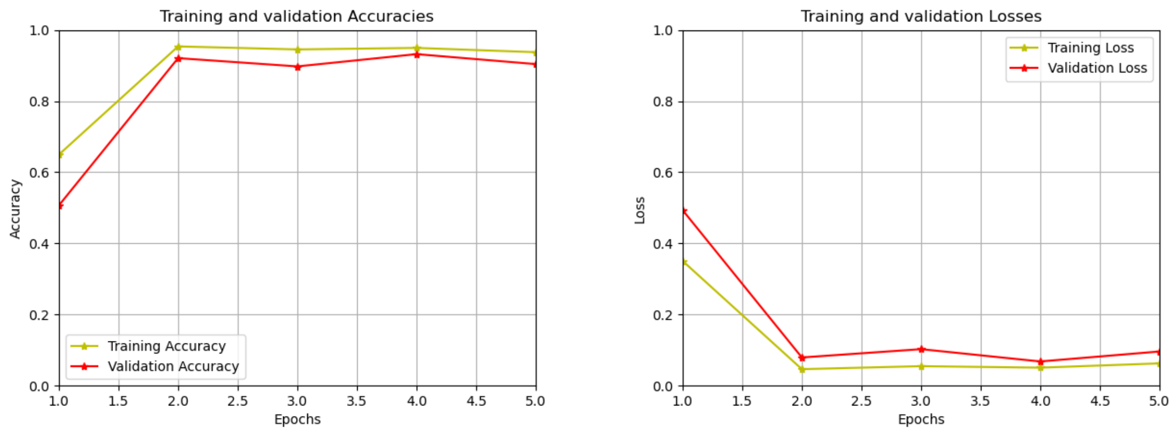

Figure 3: Training and Validation (a) Accuracy (Left), (b) Losses (Right), for frequency domain analysis

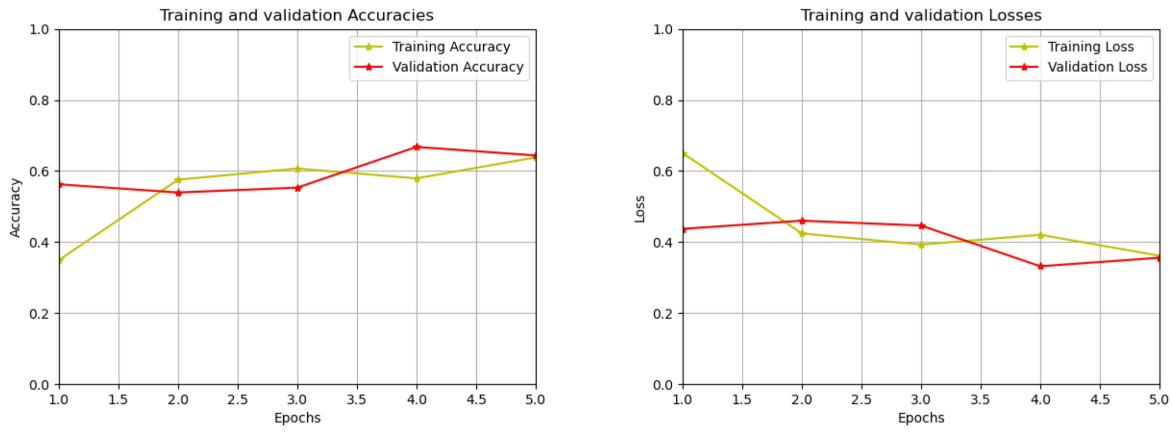

Figure 4: Training and Validation (a) Accuracy (Left), (b) Losses (Right), for time domain analysis

Now for the model performance on individual patient for this model, the prediction function is used and accuracy is computed as demonstrated in equation (2). The plot for accuracy against each patient ID is shown in figure 5 and 6 for original architecture proposed in [1] as shown in figure 1 and modified architecture respectively as shown in figure 2.

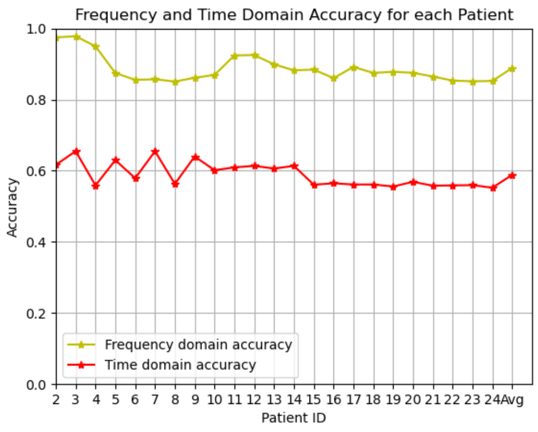

Figure 5: Accuracy against each patient for original architecture shown in figure 1

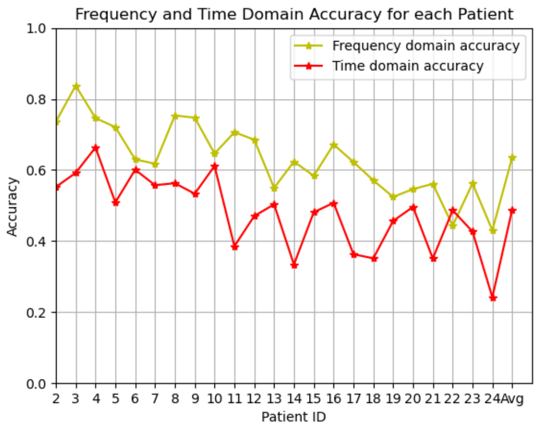

Figure 6: Accuracy against each patient for architecture shown in figure 2

## 6 Discussion

The results highlight that the original works quite well as compared to the modified one overall. Figures 4 and 3 show that our accuracies start from some minimum value rising to a steady state value where they saturate and the opposite trend can be seen for loss where the losses are maximum and they drop to steady state. The most interesting observation can be seen in figures 3 through 6, regarding how accuracy in the frequency domain is higher than that of accuracy in the time domain.

The statistical measures for figures 5 and 6, mean and variance, are summarized in table 1. We can see that there are some minor fluctuations, in figure 5, for the original architecture, hence the model performs better for some patients than the others. But for modified architecture, in figure 6, the fluctuations are quite large and since the training got interrupted the model performance decreases as the patient ID increases. Nevertheless, the model for frequency domain outperforms time domain model in both the architectures. The magnitude of fluctuations can be observed in the variances for respective architectures. It can be seen that the simpler model produces a higher accuracy and a smaller variance in accuracies over the patients, but this cannot be concretely said because the training did not take place on the entire 24 patients in the extended architecture.

| Architecture | mean ($\mu$) | variance ($\sigma^2$) |
|---|---|---|
| Original for Frequency domain | 0.887 | 0.00142 |
| Original for Time domain | 0.589 | 0.00104 |
| Modified for Frequency domain | 0.635 | 0.00966 |
| Modified for Time domain | 0.488 | 0.0112 |

Table 1: Summary of statistics for Architectures

### 6.1 Comparison With Other Methods

The original paper shows a 90% accuracy for frequency domain and nearly 60% for time domain on average for 24 patients. In our study the results are nearly the same for both time and frequency for the original architecture however for the modified architecture the results are significantly poor when compared to the results in original paper [1]. The trends over the patients are surprisingly quite similar for the original architecture for frequency domain. However, as opposed to the original implementation ours has far less fluctuations in the time domain and does not have any significant drops and even though the average accuracy value for time domain is close our implementation gives a consistent less fluctuating accuracy over each patient.

There was a study [5] on the same dataset but with conventional machine learning method, SVM. They extracted spectral and spatial features and then combined non-EEG features to form a feature vector; an SVM was then used for classification. Their approach detected 96% accuracy.

## 7 Conclusion

Epilepsy remains uncurable and there are measure taken to protect the patient from brain damage or physical injury in case of seizures. Seizure detection is now largely being done via computer aided algorithms. With the introduction of deep learning and powerful hardware manual feature extraction before feeding in data to train is not required. The objective of this study was to use EEG signals to detect seizures using a CNN architecture and compare performance for time and frequency domain input. The objective of training a CNN to detect various phases of an elliptic seizure using EEG data was achieved quite successfully. The results demonstrate that a simpler architecture is easier to train when the resources are limited. A modified architecture is not necessary when a simple architecture gives satisfactory results. Furthermore, the architecture involving frequency domain signals outperforms the ones involving time domain.

## 8 Future Work

There are several direction in which work can be done in future. One direction is to use more resources such that the architecture can be extended to see if the simplest one yields the best results. Moreover, the paper [1] argues the importance of having diverse data to achieve robust training, hence the use of data from multiple sources. Furthermore, the algorithm can be modified in a way that it works online and alerts in real time whether the person is approaching a seizure hence saving the patient from a potential fatal injury and alerting nearest medical facility at the earliest.

## 9 Appendix

```
"""
## Setting up Tensorflow
Restart runtime, set it to TPU and then run this chunk. The default is tf2.x which has slightly different ...
    functions. I prefer working with tf1.x
"""

# Commented out IPython magic to ensure Python compatibility.
# %tensorflow_version 1.x
import tensorflow as tf
print(tf.__version__)

"""## Loading Dataset
Loading the EEG dataset from physionet databases [1]. Physionet allows us to load dataset directly from cloud.
**Stop as soon as 24th folder has been loaded, try not to go below 20 GB**
"""

# !wget -r -N -c -np https://physionet.org/files/chbmit/1.0.0/
```

```
!gsutil -m cp -r gs://chbmit-1.0.0.physionet.org DESTINATION

"""Checking some files content."""

#Just to see if files got loaded.
file = '/content/DESTINATION/chbmit-1.0.0.physionet.org/chb03/chb03-summary.txt'
f = open(file, 'r')
file_contents = f.read()
print(file_contents)

# only run this for the first time.
!pip install pyedflib

# Just to check if things are syncing in.
from pyedflib import highlevel
import numpy as np
import matplotlib.pyplot as plt

# read an edf file
file2 = '/content/DESTINATION/chbmit-1.0.0.physionet.org/chb02/chb02_16+.edf'
signals, signal_headers, header = highlevel.read_edf(file2)

dft = np.fft.fft(signals, axis=1)

plt.subplot(1,2,1)
plt.plot(np.transpose(signals[0:3,:1000]))
plt.title('Raw signals')
plt.subplot(1,2,2)
plt.plot(np.transpose(dft[0:3,:1000]))
plt.title('Fourier transform')
plt.show()

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

"""## Preproccessing
Some basic data preprocessing includes obtaining signals in frequency domain usinf fft and shaping the data to ...
    arrange as labels and training+test data. Run all of this. Very important chunk.

### First we will read all the ".edf" and ".txt" files in the directory and stack them.
"""

import os

path = '/content/DESTINATION'

edfFiles = []
txtFiles = []
# r=root, d=directories, f = files
for r, d, f in os.walk(path):
    for file in f:
        if file[-4:] == '.edf':
            edfFiles.append(os.path.join(r, file))
        elif file[-4:] == '.txt':
            txtFiles.append(os.path.join(r, file))

edfFiles = sorted(edfFiles)
txtFiles = sorted(txtFiles)

for f in edfFiles:
    print(f)

for f in txtFiles:
    print(f)

"""### Reading EDF & TXT files and stacking them in batches."""

from pyedflib import highlevel
import numpy as np
import matplotlib.pyplot as plt
import re
import matplotlib.pyplot as plt

def generateLabels(edfFileName):
  sub = edfFileName[54:59]
  filePath = '/content/DESTINATION/chbmit-1.0.0.physionet.org/' + sub + '/' + sub + '-summary.txt'
  f = open(filePath, 'r')
  file_contents = f.read()

  file_list = file_contents.split('\n')
  sub = edfFileName[54:-4]
  sub = 'File Name: ' + sub + '.edf'
  ind = file_list.index(sub)

  seizures = list(map(int, re.findall(r'\d+', file_list[ind+3]) ))[0]
  start = []
  end   = []
  for i in range(seizures):
    start.append(list(map(int, re.findall(r'\d+', file_list[ind+2*i+4])))[0])
    end.append(list(map(int, re.findall(r'\d+', file_list[ind+2*i+5])))[0])
    # print(start, end)

  if seizures == 0:
    labels = np.zeros((3600))
  else:
    labels = np.ones((3600))
    labels[end[-1]:] *= 0
    for i in range(len(start)):
      labels[start[i]:end[i]] *= 2

  return labels

"""Shuffling and partitioning list."""

import random

totalData       = len(edfFiles)
random.shuffle(edfFiles)
partition       = int(len(edfFiles) * 2/3)
edfFilesVal     = edfFiles[partition:]
edfFilesTrain   = edfFiles[:partition]
trainData       = len(edfFilesTrain)
valData         = len(edfFilesVal)

print(totalData, trainData, valData)

"""Frequency Domain"""

from keras.utils import to_categorical

def stackDFTTrain(nbatch = 2):
  count = 0

  stackedDFT = np.zeros((1, 23, 256, 3))
  stackedLabels = np.zeros((1))
  rejected   = []

  while True:
    for f in edfFilesTrain:
      if stackedDFT.shape[0] ≥ nbatch*3600//3 + 1:
        print(stackedLabels.shape)
        if stackedDFT[1:nbatch*3600//3 + 1,:,:,:].shape == (3600*nbatch//3, 23, 256, 3) and ...
    to_categorical(stackedLabels[1:nbatch*3600//3 + 1], num_classes=3).shape == (3600*nbatch//3, 3):
          yield (stackedDFT[1:nbatch*3600//3 + 1,:,:,:],
                 to_categorical(stackedLabels[1:nbatch*3600//3 + 1], num_classes=3))
        stackedDFT = stackedDFT[nbatch*3600//3:,:,:,:]
        stackedLabels = stackedLabels[nbatch*3600:]
        print('extra', stackedDFT.shape, stackedLabels.shape)

      signals, signal_headers, header = highlevel.read_edf(f)
      if signals.shape[-1] % 3600 != 0 or signals.shape[0] != 23:
        rejected.append(f[54:59])
        continue

      count += 1
      print(f, signals.shape)
      s = int(signals.shape[1]/256)
      signals = np.reshape(signals, (23,256,3,s//3))
      signals = signals.transpose(3,0,1,2)
      stackedDFT = np.concatenate((stackedDFT, np.fft.fft(signals, axis=1)), axis=0)
      genLabels = generateLabels(f)
      stackedLabels = np.concatenate((stackedLabels, genLabels), axis=-1)



def stackDFTVal(nbatch = 1):
  count = 0

  stackedDFT = np.zeros((1, 23, 256, 3))
  stackedLabels = np.zeros((1))
  rejected   = []

  while True:

    for f in edfFilesVal:
      # print(f[54:-4])
      if stackedDFT.shape[0] ≥ nbatch*3600//3 + 1:
        if stackedDFT[1:nbatch*3600//3 + 1,:,:,:].shape == (3600*nbatch//3, 23, 256, 3) and ...
    to_categorical(stackedLabels[1:nbatch*3600//3 + 1], num_classes=3).shape == (3600*nbatch//3, 3):
          yield (stackedDFT[1:nbatch*3600//3 + 1,:,:,:],
                 to_categorical(stackedLabels[1:nbatch*3600//3 + 1], num_classes=3))
        stackedDFT = stackedDFT[nbatch*3600//3:,:,:,:]
        stackedLabels = stackedLabels[nbatch*3600:]

      signals, signal_headers, header = highlevel.read_edf(f)
      if signals.shape[-1] % 3600 != 0 or signals.shape[0] != 23:
        rejected.append(f[54:59])
        continue

      count += 1
      print(f, signals.shape)
      s = int(signals.shape[1]/256)
      signals = np.reshape(signals, (23,256,3,s//3))
      signals = signals.transpose(3,0,1,2)
      stackedDFT = np.concatenate((stackedDFT, np.fft.fft(signals, axis=1)), axis=0)
      genLabels = generateLabels(f)
      stackedLabels = np.concatenate((stackedLabels, genLabels), axis=-1)


"""Time domain

## Callback Class
In order to stop the training at a given threshold.
"""

from keras.utils import to_categorical

def stackTimeTrain(nbatch = 2):
  count = 0

  stackedDFT = np.zeros((1, 23, 256, 3))
  stackedLabels = np.zeros((1))
  rejected   = []

  while True:
    for f in edfFilesTrain:
      if stackedDFT.shape[0] ≥ nbatch*3600//3 + 1:
        print(stackedLabels.shape)
        if stackedDFT[1:nbatch*3600//3 + 1,:,:,:].shape == (3600*nbatch//3, 23, 256, 3) and ...
    to_categorical(stackedLabels[1:nbatch*3600//3 + 1], num_classes=3).shape == (3600*nbatch//3, 3):
          yield (stackedDFT[1:nbatch*3600//3 + 1,:,:,:],
                 to_categorical(stackedLabels[1:nbatch*3600//3 + 1], num_classes=3))
        stackedDFT = stackedDFT[nbatch*3600//3:,:,:,:]
        stackedLabels = stackedLabels[nbatch*3600:]
        print('extra', stackedDFT.shape, stackedLabels.shape)

      signals, signal_headers, header = highlevel.read_edf(f)
      if signals.shape[-1] % 3600 != 0 or signals.shape[0] != 23:
        rejected.append(f[54:59])
        continue

      count += 1
      print(f, signals.shape)
      s = int(signals.shape[1]/256)
      signals = np.reshape(signals, (23,256,3,s//3))
      signals = signals.transpose(3,0,1,2)
      stackedDFT = np.concatenate((stackedDFT, signals), axis=0)
      genLabels = generateLabels(f)
      stackedLabels = np.concatenate((stackedLabels, genLabels), axis=-1)



def stackTimeVal(nbatch = 1):
  count = 0

  stackedDFT = np.zeros((1, 23, 256, 3))
  stackedLabels = np.zeros((1))
  rejected   = []

  while True:
    for f in edfFilesTrain:
      if stackedDFT.shape[0] ≥ nbatch*3600//3 + 1:
        print(stackedLabels.shape)
        if stackedDFT[1:nbatch*3600//3 + 1,:,:,:].shape == (3600*nbatch//3, 23, 256, 3) and ...
    to_categorical(stackedLabels[1:nbatch*3600//3 + 1], num_classes=3).shape == (3600*nbatch//3, 3):
          yield (stackedDFT[1:nbatch*3600//3 + 1,:,:,:],
                 to_categorical(stackedLabels[1:nbatch*3600//3 + 1], num_classes=3))
        stackedDFT = stackedDFT[nbatch*3600//3:,:,:,:]
        stackedLabels = stackedLabels[nbatch*3600:]
        print('extra', stackedDFT.shape, stackedLabels.shape)

      signals, signal_headers, header = highlevel.read_edf(f)
      if signals.shape[-1] % 3600 != 0 or signals.shape[0] != 23:
        rejected.append(f[54:59])
        continue

      count += 1
      print(f, signals.shape)
      s = int(signals.shape[1]/256)
      signals = np.reshape(signals, (23,256,3,s//3))
      signals = signals.transpose(3,0,1,2)
      stackedDFT = np.concatenate((stackedDFT, signals), axis=0)
      genLabels = generateLabels(f)
      stackedLabels = np.concatenate((stackedLabels, genLabels), axis=-1)

"""## Building a CNN for Frequency domain

Here we just import some libraries and use them to buid an architecture.
"""

from keras.models import Model
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, Flatten, Dropout, AveragePooling2D
from keras.utils import to_categorical

nBatch = 2

in1 = Input(shape=(23, 256, 3))
c1 = Conv2D(16, (5,5), activation='relu')(in1)
m1 = AveragePooling2D()(c1)
fl = Flatten()(m1)
o = Dense(3, activation='sigmoid')(fl)

model = Model(inputs=in1, outputs=o)
print(model.summary())

"""Now after we have constructed our model let's train it."""

model.compile(optimizer = 'sgd', loss = 'categorical_crossentropy', metrics=['acc'])
testSteps = int(trainData/10)
valSteps = int(valData/5)
history_cnn = model.fit_generator(generator = stackDFTTrain(),
                                  steps_per_epoch = testSteps,
                                  epochs = 5,
                                  validation_data = stackDFTVal(),
                                  validation_steps = valSteps)
```

```
from keras.callbacks import Callback

# when accuracy reaches ACCURACY_THRESHOLD
ACCURACY_THRESHOLD = 0.95

class myCallback(Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('acc') > ACCURACY_THRESHOLD):
            print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD*100))
            self.model.stop_training = True

# Instantiate a callback object
callbacks = myCallback()

"""### Results"""

import matplotlib.pyplot as pyplot

  loss = history_cnn.history['loss']
  val_loss = history_cnn.history['val_loss']
  epochs = range(1, len(loss) + 1)
  pyplot.grid()
  pyplot.plot(epochs, loss, '*y-', label='Training loss')
  pyplot.plot(epochs, val_loss, '*r-', label='Validation loss')
  pyplot.title('Training and validation loss')
  pyplot.xlabel('Epochs')
  pyplot.ylabel('Loss')
  pyplot.legend()
  pyplot.show()

  pyplot.grid()
  acc = history_cnn.history['acc']
  val_acc = history_cnn.history['val_acc']
  epochs = range(1, len(loss) + 1)
  pyplot.plot(epochs, acc, '*y-', label='Training Accuracy')
  pyplot.plot(epochs, val_acc, '*r-', label='Validation Accuracy')
  pyplot.title('Training and validation Accuracies')
  pyplot.xlabel('Epochs')
  pyplot.ylabel('Accuracy')
  pyplot.legend()
  pyplot.show()

history_cnn.history

"""## CNN for Time domain

Repeating the process for Time domain, with the same parameters and architecture.
"""

from keras.models import Model
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, Flatten, Dropout, AveragePooling2D
from keras.utils import to_categorical

nBatch = 2

in1 = Input(shape=(23, 256, 3))
c1 = Conv2D(16, (5,5), activation='relu')(in1)
m1 = MaxPooling2D()(c1)
fl = Flatten()(m1)
o = Dense(3, activation='sigmoid')(fl)

model2 = Model(inputs=in1, outputs=o)
print(model2.summary())

model2.compile(optimizer = 'sgd', loss = 'categorical_crossentropy', metrics=['acc'])
testSteps = int(trainData/10)
valSteps = int(valData/5)
history_cnn2 = model2.fit_generator(generator = stackTimeTrain(),
                                    steps_per_epoch = testSteps,
                                    epochs = 5,
                                    validation_data = stackTimeVal(),
                                    validation_steps = valSteps)

"""### Results"""
```

```python
import matplotlib.pyplot as pyplot

loss2 = history_cnn2.history['loss']
val_loss2 = history_cnn2.history['val_loss']
epochs2 = range(1, len(loss2) + 1)
pyplot.grid()
pyplot.plot(epochs2, loss2, '*y-', label='Training loss')
pyplot.plot(epochs2, val_loss2, '*r-', label='Validation loss')
pyplot.title('Training and validation loss')
pyplot.xlabel('Epochs')
pyplot.ylabel('Loss')
pyplot.legend()
pyplot.show()

pyplot.grid()
acc2 = history_cnn2.history['acc']
val_acc2 = history_cnn2.history['val_acc']
epochs2 = range(1, len(loss2) + 1)
pyplot.plot(epochs2, acc2, '*y-', label='Training Accuracy')
pyplot.plot(epochs2, val_acc2, '*r-', label='Validation Accuracy')
pyplot.title('Training and validation Accuracies')
pyplot.xlabel('Epochs')
pyplot.ylabel('Accuracy')
pyplot.legend()
pyplot.show()


"""
# Bibliography
[1] CHB-MIT Scalp EEG Database, Retrieved from: https://physionet.org/content/chbmit/1.0.0/
"""
```

Listing 1: Python code

## References

[1] Mengni Zhou1, et al, *Epileptic Seizure Detection Based on EEG Signals and CNN*, Front. Neuroinform, 2018.

[2] Xiaoyan Wei, et al, *Automatic seizure detection using three-dimensional CNN based on multi-channel EEG*, BMC Med Inform Decis Mak, 2018.

[3] Dongmei Zhou and Xuemei Li, *Epilepsy EEG Signal Classification Algorithm Based on Improved RBF*, Front. Neuroscience, 2020.

[4] Ali Emami, et al, *Seizure detection by convolutional neural network-based analysis of scalp electroencephalography plot images*, Elsevier Inc., 2019.

[5] Shoeb, A. H., et al, *Application of machine learning to epileptic seizure detection*, International Conference on Machine Learning (Haifa), 975–982, 2010