ASSIGNMENT – 4
Advanced Programming Lab

Name: Utsav Balar
Roll.no. t24cs003

Objective

Assignments on Seizure Detection using the CHB-MIT Scalp EEG Database. Develop a machine learning model to automatically detect seizure events from EEG data.

Dataset link: https://physionet.org/content/chbmit/1.0.0/

*Analysis:

Epilepsy and Seizures

Epilepsy is the most common neurological disorder in which clusters of nerve cells (neurons) in the brain sometimes signal abnormally and cause seizures. Currently it affects 60 million people world-wide. Normal function of neurons is to generate electrical and chemical signals that act on other neurons, which in turn cause secondary actions leading to a desired behavior. During a seizure, many neurons try to send signals at the same time; this abnormal surge of excessive elec- trical activity causes involuntary movements, and may cause a loss of control, lapse of attention or whole-body convulsion.

The introduction of new anti-epileptic drugs has provided most patients the ability to control their seizures, however, three in ten patients with epilepsy continue to have seizures despite treat- ment. Given the nature of the disease is quite unpredictable, patients often experience high levels of anxiety, mainly due to the impending loss of control and/or awareness during a seizure. Thus, prediction is key to minimizing the anxiety and fear experienced by the patient, and prediction begins with detection. A device able to detect seizures quickly could vastly improve patient care.

A classification problem

Seizure detection is usually done using scalp electroencephalogram (EEG), a non-invasive, multi-channel recording of the brain's electrical activity. Classification between normal activity and a seizure is usually done in two steps: noise filtering and classification. We propose here two approaches to noise filtering: use two transforms, Fourier and Discrete wavelets, to filter the data but also extract the useful information and structure behind the signals. We will test classical models of classification using Python's sklearn libraries to run them.

Data

A study was conducted at the Children's Hospital Boston where EEG was recorded on pediatric subjects with intractable (i.e. epileptic) seizures. There were 22 subjects in the study: 5 male, ages 3-22, and 17 females, ages 1.5-19; subject 21 was evaluated a second time 1.5 years later, thus re- sulting in 23 cases. Each case contains between 9 and 42 continuous .edf fles from a single subject, typically consisting of one hour each. Prior to measuring, subjects were taken of of any anti- seizure medication. The files can be found at https://physionet.org/physiobank/database/chbmit/

EEG Samples

Most of the EEG files have 23 recordings from electrodes placed around the head of the patient. Figures 1 and 2 present two EEG recordings of 40 seconds each. The first one was recorded at 1:43pm, while the patient was awake, and so should represent normal activity of the brain. The second one was recorded 50 minutes later. Within this 40 seconds, the patient experiences one epileptic seizure. As the size of the EEG makes it diffcult to assess how much bigger the amplitudes are when the patient is experiencing a seizure, Figure 3 zooms in on one channel for the non-seizure and seizure activity. Here we begin to see a marked difference between the two.

1. Signal Extraction for training data

Each 8 second signals of 18 channels are extracted, sliding forward by 4 seconds. Each set of signals are labeled with the ratio of seizure in the time window. i.e. a set of signals are labeled 1.0 if it is in the middle of seizure.

1. Preprocessing The original signal frequency is 256 Hz, but to simplify the data, it has been resampled to 128 Hz.

Certainly! Let's walk through this code block-by-block to understand its purpose, including the rationale for using specific callbacks, models, and visualizations:

1. Imports and Setup

The code imports essential libraries:

1. **Pandas, Matplotlib, NumPy**: Used for data manipulation and visualization.
2. **MNE, WFDB**: Specialized libraries for handling EEG data from `.edf` files and annotations, respectively.
3. **Keras & TensorFlow**: For building and training the neural network.

2. Data Setup and Listing Files

1. **Channel Labels**: Defined for EEG data processing. There are 18 labels, representing EEG channels, for analyzing the EEG signals.
2. **Base Path**: Specifies the data directory.
3. **List Files Function**: This function walks through the data directory and lists all file paths, helping in understanding the dataset's structure.

3. Patient Data Splitting

1. **Collecting Patient IDs**: Extracts patient identifiers from the folder structure.

2. **Train-Test Split**: Splits the patients into training (80%) and test (20%) sets using a random selection. This is crucial for evaluating model performance on unseen data.

4. Signal Extraction

1. **Loading or Extracting Signals**:
2. If preprocessed files exist (`signal_samples.npy` and `is_seizure.npy`), they are loaded to save time.
3. Otherwise, signals are extracted from `.edf` files:
4. **MNE Library**: Used to read EEG signals and metadata.
5. **Sliding Window**: The EEG data is split into 8-second windows, sliding forward by 4 seconds. This temporal segmentation helps in analyzing patterns over time.
6. **Labeling**: Windows are labeled based on the presence of seizures using seizure annotations from the WFDB library.
7. **Balanced Sampling**: The code balances the data by oversampling seizure events and undersampling non-seizure events.

5. Data Preprocessing

1. **Resampling**: EEG signals are downsampled from 256 Hz to 128 Hz to simplify the data.
2. **Visualizing Sample Signals**: Plots EEG signals to visualize a sample of the extracted signals:
3. The **top plot** shows all 18 channels in one figure, with each channel shifted vertically for clarity.
4. The **bottom plot** uses a grayscale colormap to give a heatmap representation of the signals over time.
5. **Checking Seizure Distribution**: The number and ratio of windows with seizures are printed, helping to understand class imbalance.

6. Model Definition

1. A **Sequential CNN Model** is built for seizure detection:
2. **Input Layer**: Specifies the input shape as (18 channels, 1024 time points, 1 channel dimension).
3. **Convolutional Layers**: Multiple Conv2D layers with different filters and kernel sizes to capture spatial and temporal features in the EEG data.
4. **Pooling Layers**: MaxPooling layers reduce the dimensionality of the data while retaining important features.
5. **Global Average Pooling**: Reduces each feature map to a single value, helping to connect the convolutional layers to fully connected layers.
6. **Dense Layers**: Fully connected layers with ReLU activation for classification.
7. **Dropout Layers**: Used for regularization, preventing overfitting by randomly dropping some neurons during training.
8. **Output Layer**: A single neuron with a sigmoid activation function for binary classification (seizure vs. non-seizure).

7. Callbacks for Training

1. **Early Stopping**: Monitors the validation loss and stops training if it does not improve for 20 epochs, restoring the best weights. This prevents overfitting.
2. **ReduceLROnPlateau**: Reduces the learning rate if the validation loss plateaus, allowing the model to fine-tune its performance.

8. Mixed Precision Training

1. **Mixed Precision**: Uses half-precision (float16) arithmetic to speed up training and reduce memory usage, especially helpful when using modern NVIDIA GPUs.

9. Model Training

1. **Model Compilation**: The model is compiled using the Adam optimizer with a learning rate of 0.0001. The loss function is binary cross-entropy, and the metric is accuracy.
2. **Model Fitting**: The model is trained for 50 epochs with a batch size of 128, using the defined callbacks for early stopping and learning rate adjustment.
3. **Saving the Model**: The trained model is saved in the specified file.

10. Visualizing Training Performance

1. **Loss Plot**: Plots the training and validation loss over epochs, with an annotation for the early stopping epoch.
2. **Accuracy Plot**: Plots the training and validation accuracy, with an annotation for the early stopping epoch.
3. These plots help in understanding how well the model is training and when it starts to overfit.

Purpose of Callbacks and Model Design

1. **EarlyStopping**: Avoids overfitting by halting training when performance on the validation set stops improving.
2. **ReduceLROnPlateau**: Helps fine-tune the learning process by lowering the learning rate when progress stagnates.
3. **CNN Architecture**: The convolutional layers are designed to capture both spatial patterns (across channels) and temporal patterns (over time) in the EEG data, which are crucial for detecting seizure events.
4. **Global Average Pooling**: Reduces the model’s complexity by summarizing feature maps, making it efficient for classification.

Code for training:

```
# import pandas as pd
from keras import layers
from keras.utils import plot_model
from sklearn import model_selection
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import mixed_precision
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping
from typing import List
import gc
import glob
import keras
import logging
import matplotlib.pyplot as plt
import mne
import numpy as np
import os
import random
import re
import tensorflow as tf
import tqdm
import wfdb


# Define channel labels
ch_labels: List[str] = [
    "FP1-F7",
    "F7-T7",
    "T7-P7",
    "P7-O1",
    "FP1-F3",
    "F3-C3",
    "C3-P3",
    "P3-O1",
    "FP2-F4",
    "F4-C4",
    "C4-P4",
    "P4-O2",
    "FP2-F8",
    "F8-T8",
    "T8-P8",
    "P8-O2",
```

```
    "FZ-CZ",
    "CZ-PZ",
]

# Define base path to data
base_path: str =
"/home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit/"


# Walk through the directory and print all filenames
def list_files_in_directory(path: str) -> None:
    for dirname, _, filenames in os.walk(path):
        for filename in filenames:
            print(os.path.join(dirname, filename))


list_files_in_directory(base_path)

# Gather folders and extract patient numbers
folders: List[str] = sorted(glob.glob(f"{base_path}/*/"))
n_patient: List[str] = [folder.rstrip("/").split("/")[-1][-2:] for
folder in folders]

print("Patients:", *n_patient)

# Set seed for reproducibility
random.seed(2024)

# Split patients into train and test sets
# Patients are assigned to training and test groups through a
random selection process.
ratio_train: float = 0.8
train_patient_str: List[str] = sorted(
    random.sample(n_patient, round(ratio_train * len(n_patient)))
)
test_patient_str: List[str] = sorted(
    [patient for patient in n_patient if patient not in
train_patient_str]
)

print("Train PT:", *train_patient_str)
print("Test PT:", *test_patient_str)


# Collect file paths for train and test sets
```

```
def collect_files(patient_str_list: List[str], path: str) ->
List[str]:
    return [
        edf_file
        for patient in patient_str_list
        for edf_file in glob.glob(f"{path}/chb{patient}/*.edf")
    ]


files_train: List[str] = collect_files(train_patient_str,
base_path)
files_test: List[str] = collect_files(test_patient_str, base_path)

print("Number of training files:", len(files_train))
print("Number of test files:", len(files_test))

mne.set_log_level(verbose="ERROR")

logger = logging.getLogger(__name__)
fh = logging.FileHandler("read_files.log")
logger.addHandler(fh)

# Sliding window (The EEG data is split into 8-second windows,
sliding forward by 4 seconds)
time_window: int = 8
time_step: int = 4

if os.path.exists(base_path + "signal_samples.npy") &
os.path.exists(
    base_path + "is_seizure.npy"
):
    print("------------------- File exists
-----------------------------")
    array_signals: np.ndarray = np.load(base_path +
"signal_samples.npy")
    array_is_seizure: np.ndarray = np.load(base_path +
"is_seizure.npy")
else:
    p: float = 0.01
    counter: int = 0
    for temp_f in files_train:
        temp_edf = mne.io.read_raw_edf(temp_f)
        temp_labels: List[str] = temp_edf.ch_names
        if sum(
            [
```

```
                any([0 if re.match(c, l) == None else 1 for l in
temp_edf.ch_names])
                for c in ch_labels
            ]
        ) == len(ch_labels):
            frequency: int = int(1 / (temp_edf.times[1] -
temp_edf.times[0]))
            step_window: int = time_window * frequency
            step: int = time_step * frequency

            temp_is_seizure: np.ndarray =
np.zeros((temp_edf.n_times,))
            if os.path.exists(temp_f + ".seizures"):
                temp_annotation: wfdb.Annotation =
wfdb.rdann(temp_f, "seizures")
                for i in range(int(temp_annotation.sample.size /
2)):
                    temp_is_seizure[
                        temp_annotation.sample[i * 2] :
temp_annotation.sample[
                            i * 2 + 1
                        ]
                    ] = 1
            temp_len: int = temp_edf.n_times

            temp_is_seizure_ind: np.ndarray = np.array(
                [
                    temp_is_seizure[i * step : i * step +
step_window].sum()
                    / step_window
                    for i in range((temp_len - step_window) //
step)
                ]
            )

            temp_0_sample_size = round(p *
np.where(temp_is_seizure_ind == 0)[0].size)
            temp_1_sample_size = np.where(temp_is_seizure_ind > 0)
[0].size

            counter = counter + temp_0_sample_size +
temp_1_sample_size
        temp_edf.close()

    # pyright: reportPossiblyUnboundVariable=false
    array_signals: np.ndarray = np.zeros(
```

```
        (counter, len(ch_labels), step_window), dtype=np.float32
    )
    array_is_seizure: np.ndarray = np.zeros(counter, dtype=bool)

    counter: int = 0
    for n, temp_f in enumerate(tqdm.tqdm(files_train)):
        to_log: str = "No. {}: Reading. ".format(n)
        temp_edf = mne.io.read_raw_edf(temp_f)
        temp_labels: List[str] = temp_edf.ch_names
        n_label_match: int = sum(
            [
                any([0 if re.match(c, l) == None else 1 for l in
temp_edf.ch_names])
                for c in ch_labels
            ]
        )
        if n_label_match == len(ch_labels):
            ch_mapping = {
                sorted([l for l in temp_edf.ch_names if re.match(c,
l) != None])[0]: c
                for c in ch_labels
            }
            temp_edf.rename_channels(ch_mapping)
            # temp_edf = temp_edf.pick(ch_labels)

            temp_is_seizure: np.ndarray =
np.zeros((temp_edf.n_times,))
            temp_signals: np.ndarray =
temp_edf.get_data(picks=ch_labels) * 1e6

            if os.path.exists(temp_f + ".seizures"):
                to_log: str = to_log + "seizure exists."
                temp_annotation: wfdb.Annotation =
wfdb.rdann(temp_f, "seizures")
                for i in range(int(temp_annotation.sample.size /
2)):
                    temp_is_seizure[
                        temp_annotation.sample[i * 2] :
temp_annotation.sample[
                            i * 2 + 1
                        ]
                    ] = 1
            else:
                to_log = to_log + "No seizure."

            temp_len: int = temp_edf.n_times
```

```
            frequency: int = int(1 / (temp_edf.times[1] -
temp_edf.times[0]))
            step_window: int = time_window * frequency
            step: int = time_step * frequency

            temp_is_seizure_ind: np.ndarray = np.array(
                [
                    temp_is_seizure[i * step : i * step +
step_window].sum()
                    / step_window
                    for i in range((temp_len - step_window) //
step)
                ]
            )
            del temp_is_seizure

            temp_0_sample_size: int = round(
                p * np.where(temp_is_seizure_ind == 0)[0].size
            )
            temp_1_sample_size: int = np.where(temp_is_seizure_ind
> 0)[0].size

            # seizure data
            temp_ind: list = list(np.where(temp_is_seizure_ind > 0)
[0])
            for i in temp_ind:
                array_signals[counter, :, :] = temp_signals[
                    :, i * step : i * step + step_window
                ]
                array_is_seizure[counter] = True
                counter = counter + 1

            # no seizure data
            temp_ind: list = random.sample(
                list(np.where(temp_is_seizure_ind == 0)[0]),
temp_0_sample_size
            )
            for i in temp_ind:
                array_signals[counter, :, :] = temp_signals[
                    :, i * step : i * step + step_window
                ]
                array_is_seizure[counter] = False
                counter = counter + 1
```

```
            to_log += "{} signals added: {} w/o seizure, {} w/
seizure.".format(
                temp_0_sample_size + temp_1_sample_size,
                temp_0_sample_size,
                temp_1_sample_size,
            )

        else:
            to_log += "Not appropriate channel labels. Reading
skipped.".format(n)

        logger.info(to_log)
        temp_edf.close()

        if n % 10 == 0:
            gc.collect()
    gc.collect()

    np.save("signal_samples", array_signals)
    np.save("is_seizure", array_is_seizure)


# Preprocessing
#
# The original signal frequency is 256 Hz, but to simplify the
data,
# it has been resampled to 128 Hz.
array_signals = array_signals[:, :, ::2]

# show a sample of extracted signals (the last one)
vertical_width: int = 300
signals: np.ndarray = array_signals[-1, :, :]
fs: int = 128

fig, ax = plt.subplots()
for i in range(signals.shape[0]):
    ax.plot(
        np.arange(signals.shape[-1]) / frequency,
        signals[i, :] + i * vertical_width,
        linewidth=0.5,
        color="tab:blue",
    )
    ax.annotate(ch_labels[i], xy=(0, i * vertical_width))
ax.invert_yaxis()
plt.show()
```

```
# Checking how much of signals have seizure inside.
array_n: np.ndarray = np.where(array_is_seizure > 0.0)[0]
print("Number of all the extracted signals:
{}".format(array_is_seizure.size))
print("Number of signals with seizures: {}".format(array_n.size))
print(
    "Ratio of signals with seizures: {:.3f}".format(
        array_n.size / array_is_seizure.size
    )
)

# Let's see samples with seizures.
for n in random.sample(list(array_n), 10):
    vertical_width: int = 300
    temp_signals: np.ndarray = array_signals[n, :, :]
    frequency: int = 128

    fig, ax = plt.subplots(2, 1, figsize=(10, 6),
gridspec_kw={"height_ratios": [3, 1]})
    for i in range(temp_signals.shape[0]):
        ax[0].plot(
            np.arange(temp_signals.shape[-1]) / frequency,
            temp_signals[i, :] + i * vertical_width,
            linewidth=0.5,
            color="tab:blue",
        )
        ax[0].annotate(ch_labels[i], xy=(0, i * vertical_width))
    ax[0].invert_yaxis()
    ax[0].set_xlim(0, 8)
    ax[0].set_title("sample no. {}".format(n))

    ax[1].pcolormesh(
        np.arange(temp_signals.shape[-1]) / frequency,
        np.arange(len(ch_labels)),
        temp_signals[:, :],
        cmap="gray",
    )
    ax[1].invert_yaxis()

    plt.show()

# Add a Channel dimension
array_signals: np.ndarray = array_signals[:, :, :, np.newaxis]

# splitting training data into training & validation data.
```

```
X_train, X_val, y_train, y_val = model_selection.train_test_split(
    array_signals, array_is_seizure, test_size=0.3,
stratify=(array_is_seizure > 0)
)

del array_signals, array_is_seizure

# Initialize the deep learning model
model = keras.models.Sequential()

# Add the first convolutional layer with input shape
# Ensure input_shape matches the expected dimensions (channels,
height, width)
model.add(
    layers.Input(shape=(18, 1024, 1)),
)

model.add(
    layers.Conv2D(filters=64, kernel_size=(2, 4), padding="same",
activation="relu")
)

model.add(
    layers.Conv2D(
        # filters=64,
        filters=32,
        kernel_size=(2, 4),
        padding="same",
        activation="relu",
        input_shape=(18, 1024, 1),
    )
)  # Modify input shape as needed

# Add more convolutional layers and pooling layers
model.add(
    layers.Conv2D(
        filters=32,
        kernel_size=(2, 4),
        strides=(1, 2),
        padding="same",
        activation="relu",
    )
)
model.add(layers.MaxPooling2D(pool_size=(1, 2)))

model.add(
```

```
    layers.Conv2D(filters=64, kernel_size=(2, 4), padding="same",
activation="relu")
)
model.add(
    layers.Conv2D(
        filters=64,
        kernel_size=(2, 4),
        strides=(1, 2),
        padding="same",
        activation="relu",
    )
)
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(
    layers.Conv2D(filters=128, kernel_size=(4, 4), padding="same",
activation="relu")
)
model.add(
    layers.Conv2D(
        filters=128,
        kernel_size=(4, 4),
        strides=(1, 2),
        padding="same",
        activation="relu",
    )
)
model.add(layers.MaxPooling2D(pool_size=(1, 2)))

# Add a global average pooling layer
model.add(layers.GlobalAveragePooling2D())

# Add fully connected layers
model.add(layers.Dense(256, activation="relu"))
model.add(layers.Dropout(0.25))  # Dropout for regularization
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dropout(0.25))  # Another dropout layer
model.add(
    layers.Dense(1, activation="sigmoid")
)  # Output layer for binary classification
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])

model.summary()
```

```
plot_model(model, show_shapes=True, to_file="model.png")
plot_model(model, show_shapes=True, dpi=70)

LEARNING_RATE: float = 1e-4
OPTIMIZER: tf.keras.optimizers =
tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)

model.compile(optimizer=OPTIMIZER, loss="binary_crossentropy",
metrics=["accuracy"])
# callbacks
VERBOSE: int = 1

# Define the early stopping callback
es = EarlyStopping(
    monitor="val_loss",
    patience=20,
    verbose=VERBOSE,
    mode="auto",
    restore_best_weights=True,
)


# Define the learning rate reduction callback
lr = ReduceLROnPlateau(
    monitor="val_loss", factor=0.75, patience=5, verbose=1,
min_lr=1e-8
)

callbacks: List = [es, lr]

# Nvidia thing, allows using float16 instead of float32
mixed_precision.set_global_policy("mixed_float16")

# Clear any previous session
tf.keras.backend.clear_session()

# Train the model
hist = model.fit(
    x=X_train,
    y=y_train,
    validation_data=(X_val, y_val),
    # epochs=200, # lets not use this
    epochs=50,
    # batch_size=256, # does not work
    batch_size=128,
    callbacks=callbacks,
```

```
)
# Save the model
keras.saving.save_model(model, "CHB_MIT_sz_detec_demo.keras")

fig, ax = plt.subplots(1, 2, figsize=(12, 4))

ax[0].plot(hist.history["loss"], label="loss")
ax[0].plot(hist.history["val_loss"], label="val_loss")
ax[0].set_xlabel("epoch")
ax[0].set_ylabel("loss")
ax[0].axvline(x=es.best_epoch, label="early stopping",
color="tab:red", alpha=0.5)
r = 0.2
temp_y = r * min(hist.history["loss"]) + (1 - r) *
max(hist.history["loss"])
ax[0].annotate(" early stopping:\n best epoch", xy=(es.best_epoch,
temp_y))
ax[0].set_title("Loss")
ax[0].legend()

ax[1].plot(hist.history["accuracy"], label="accuracy")
ax[1].plot(hist.history["val_accuracy"], label="val_accuracy")
ax[1].set_xlabel("epoch")
ax[1].set_ylabel("accuracy")
r = 0.8
temp_y = r * min(hist.history["accuracy"]) + (1 - r) *
max(hist.history["accuracy"])
ax[1].axvline(x=es.best_epoch, label="early stopping",
color="tab:red", alpha=0.5)
ax[1].annotate(" early stopping:\n best epoch", xy=(es.best_epoch,
temp_y))
ax[1].set_title("Accuracy")
ax[1].legend()

plt.show()
```

Results:

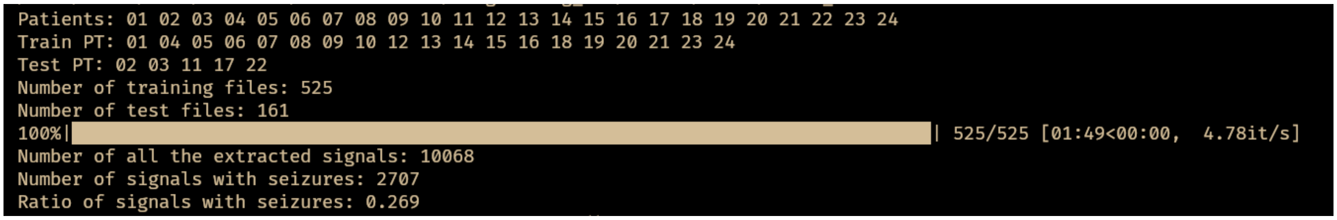


fig1: database information after processing and splitting into test and train

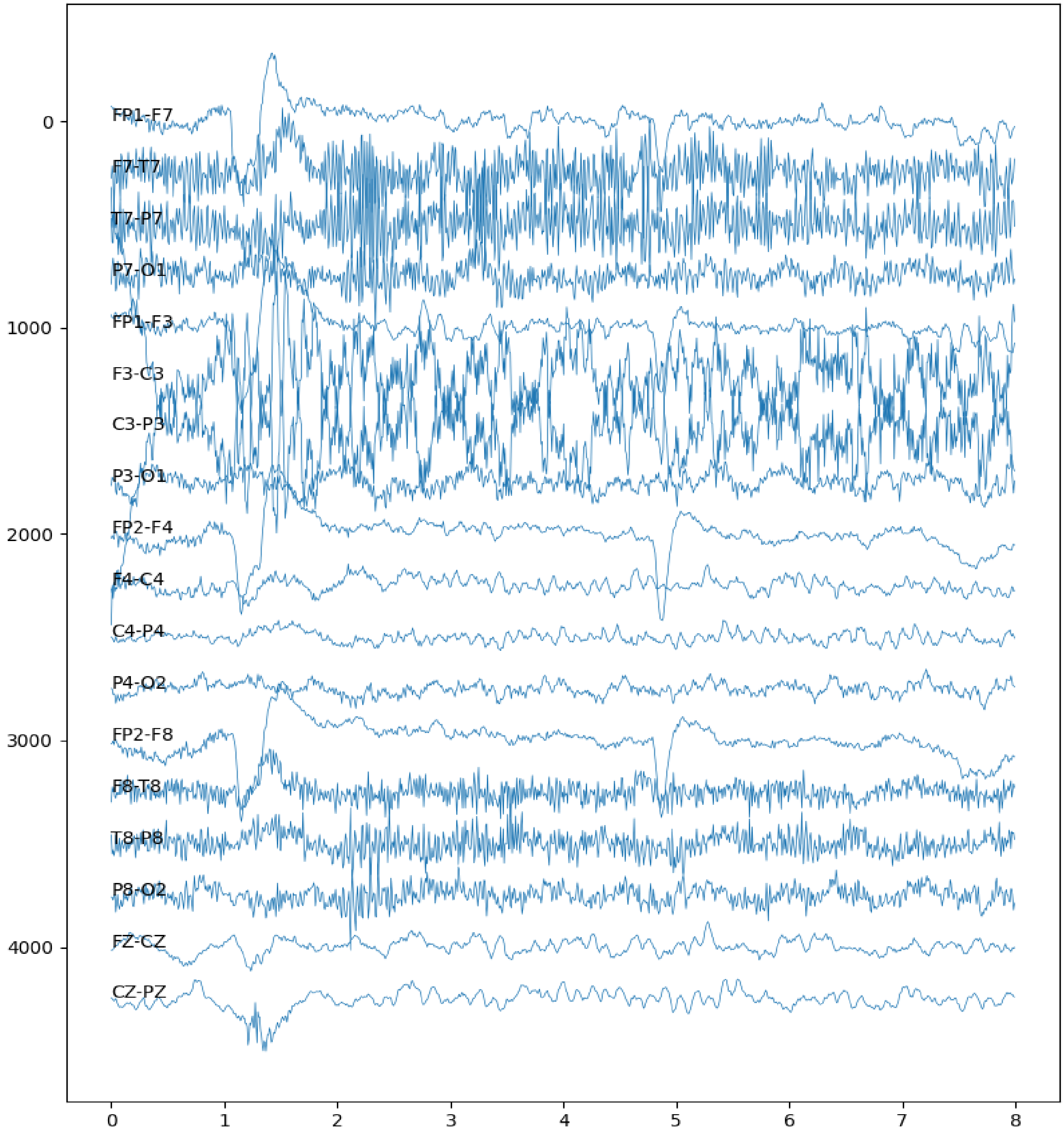
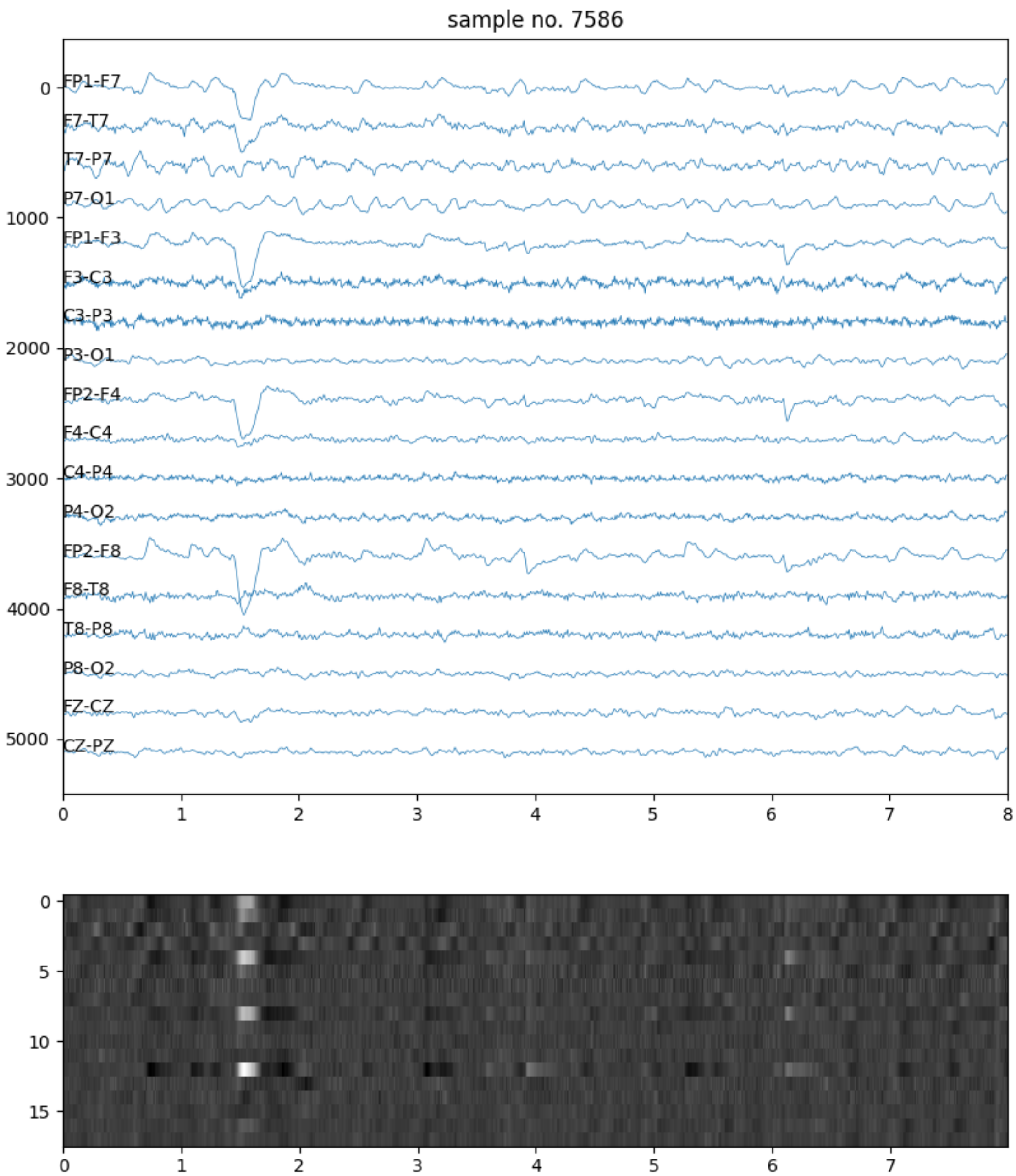
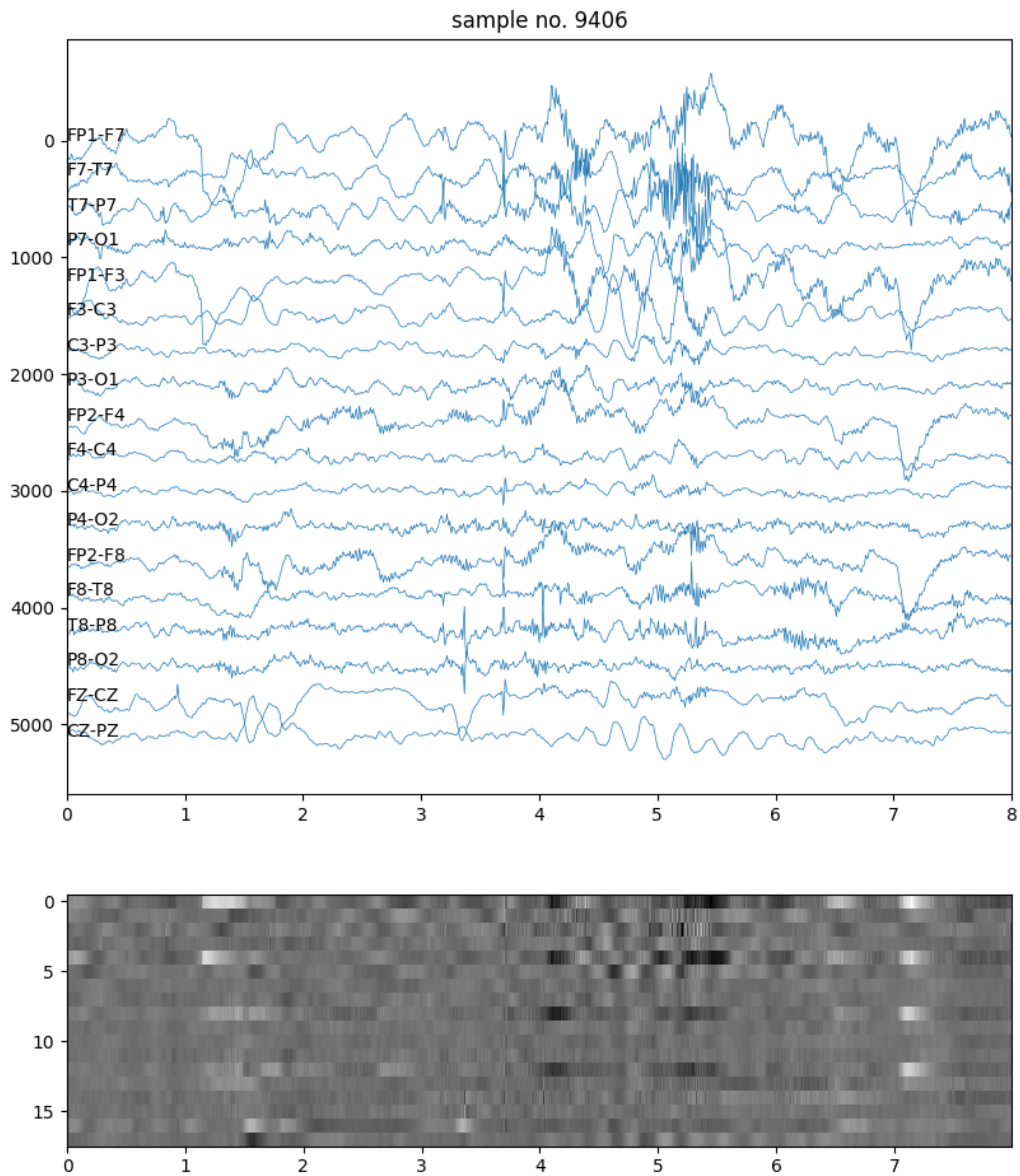
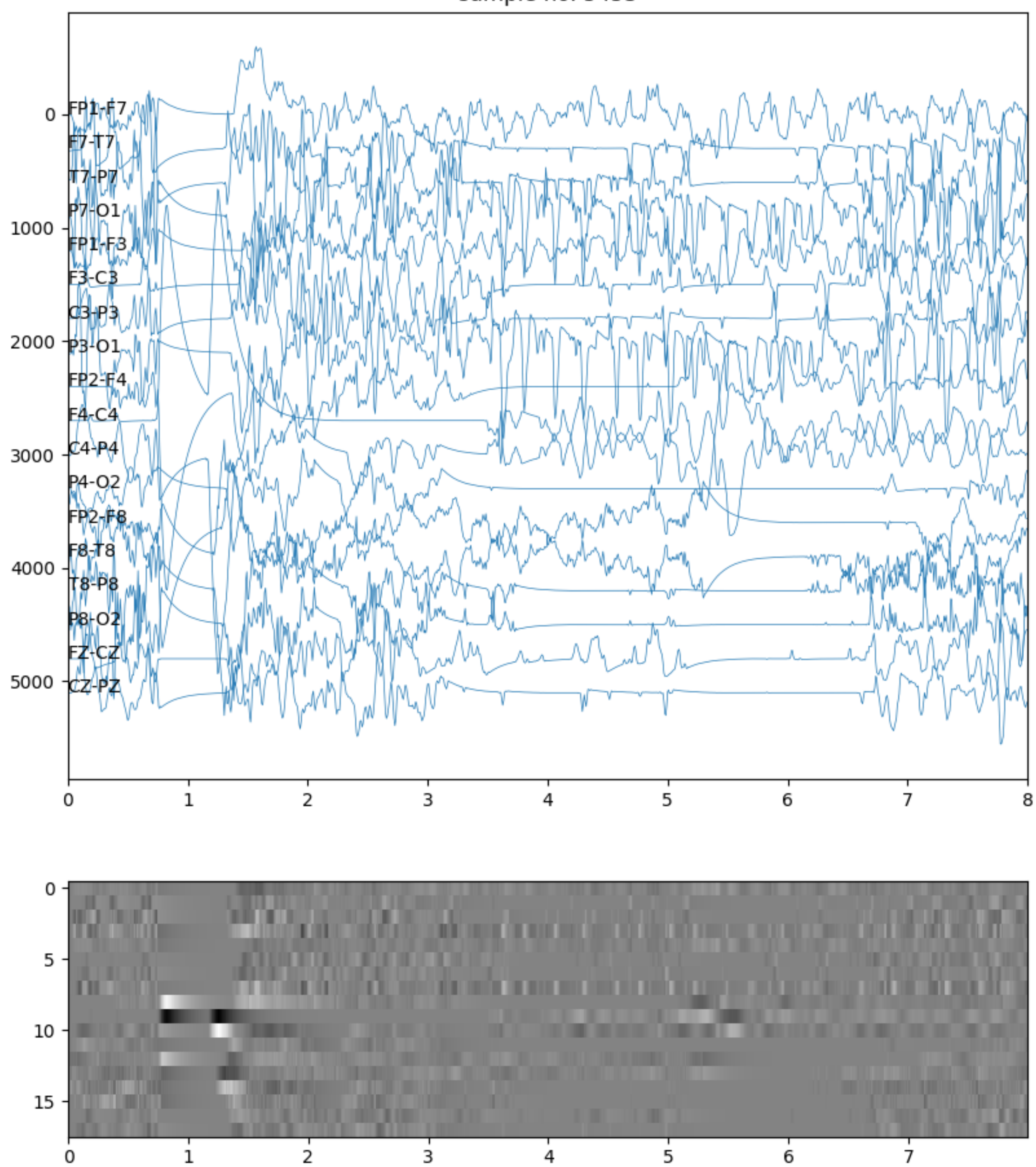


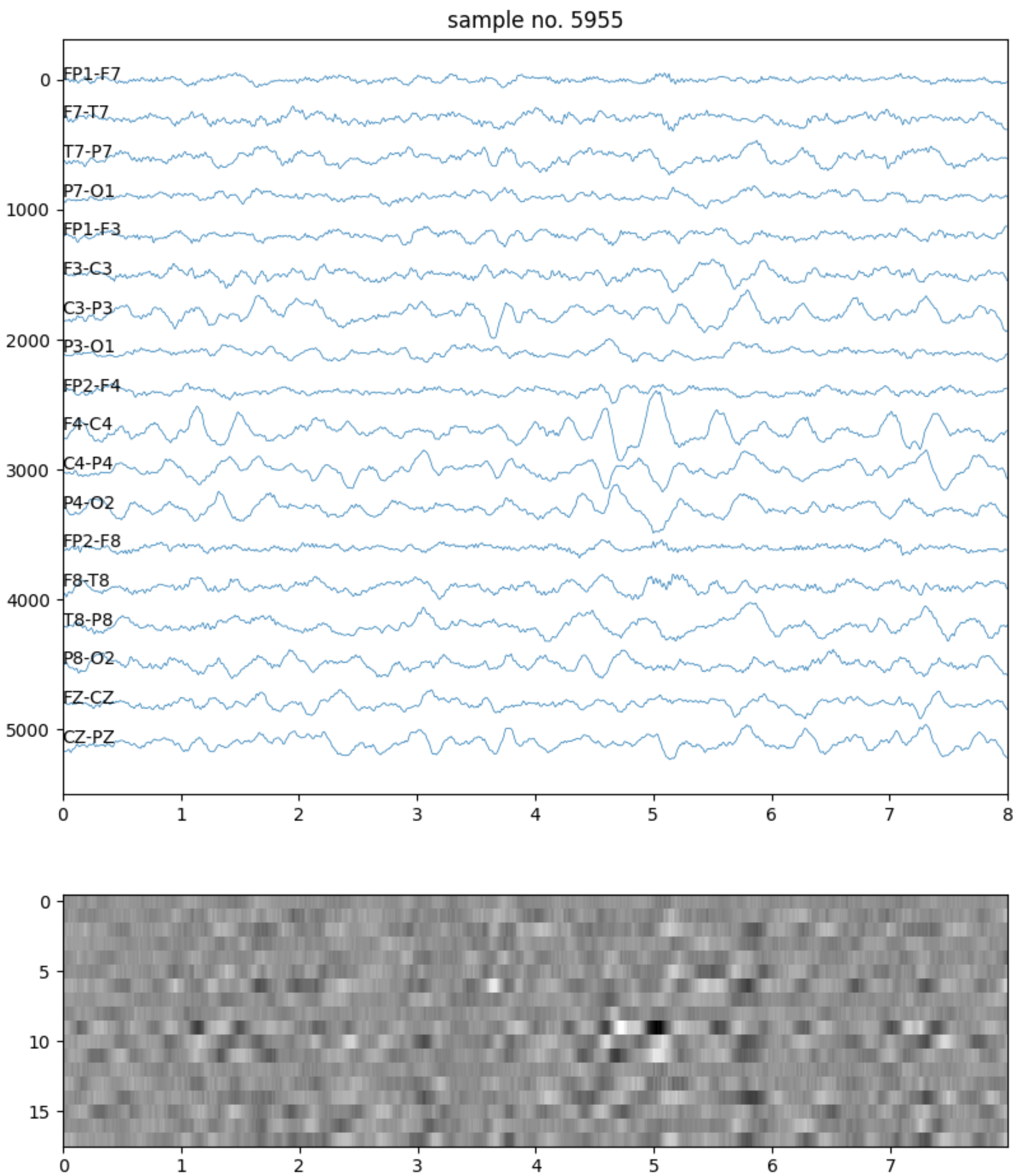
fig2: A sample of extracted signals


sample no. 7586
FP1-F7
F7-T7
T7-P7
P7-O1
FP1-F3
F3-C3
C3-P3
P3-O1
FP2-F4
F4-C4
C4-P4
P4-O2
FP2-F8
F8-T8
T8-P8
P8-O2
FZ-CZ
CZ-PZ
0
1000
2000
3000
4000
5000
0 1 2 3 4 5 6 7 8
0
5
10
15
0 1 2 3 4 5 6 7


sample no. 9406
FP1-F7
F7-T7
T7-P7
P7-O1
FP1-F3
F3-C3
C3-P3
P3-O1
FP2-F4
F4-C4
C4-P4
P4-O2
FP2-F8
F8-T8
T8-P8
P8-O2
FZ-CZ
CZ-PZ

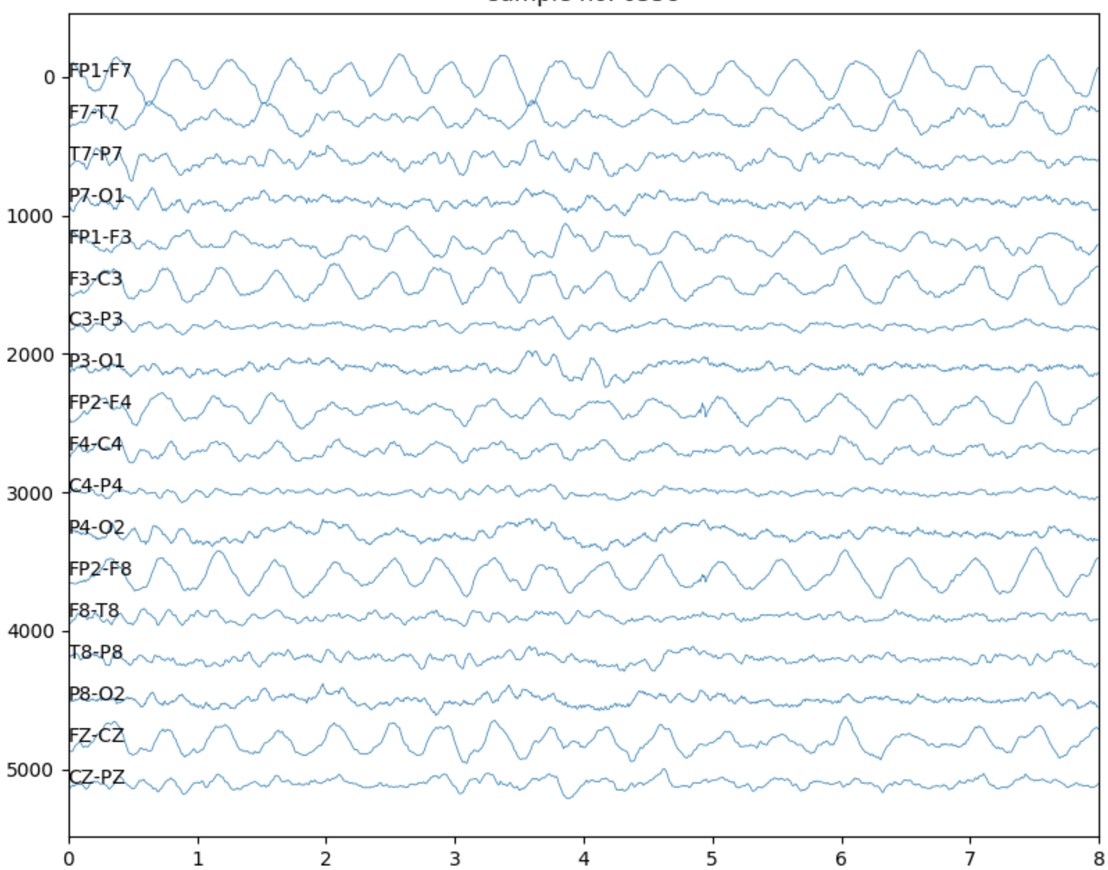
sample no. 5453

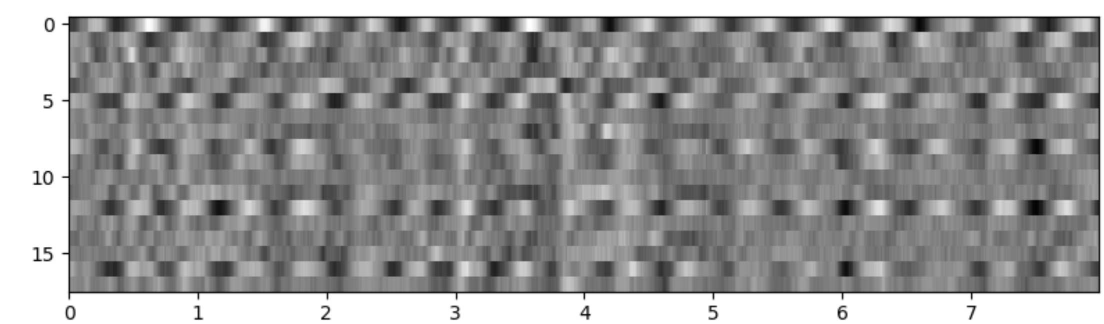


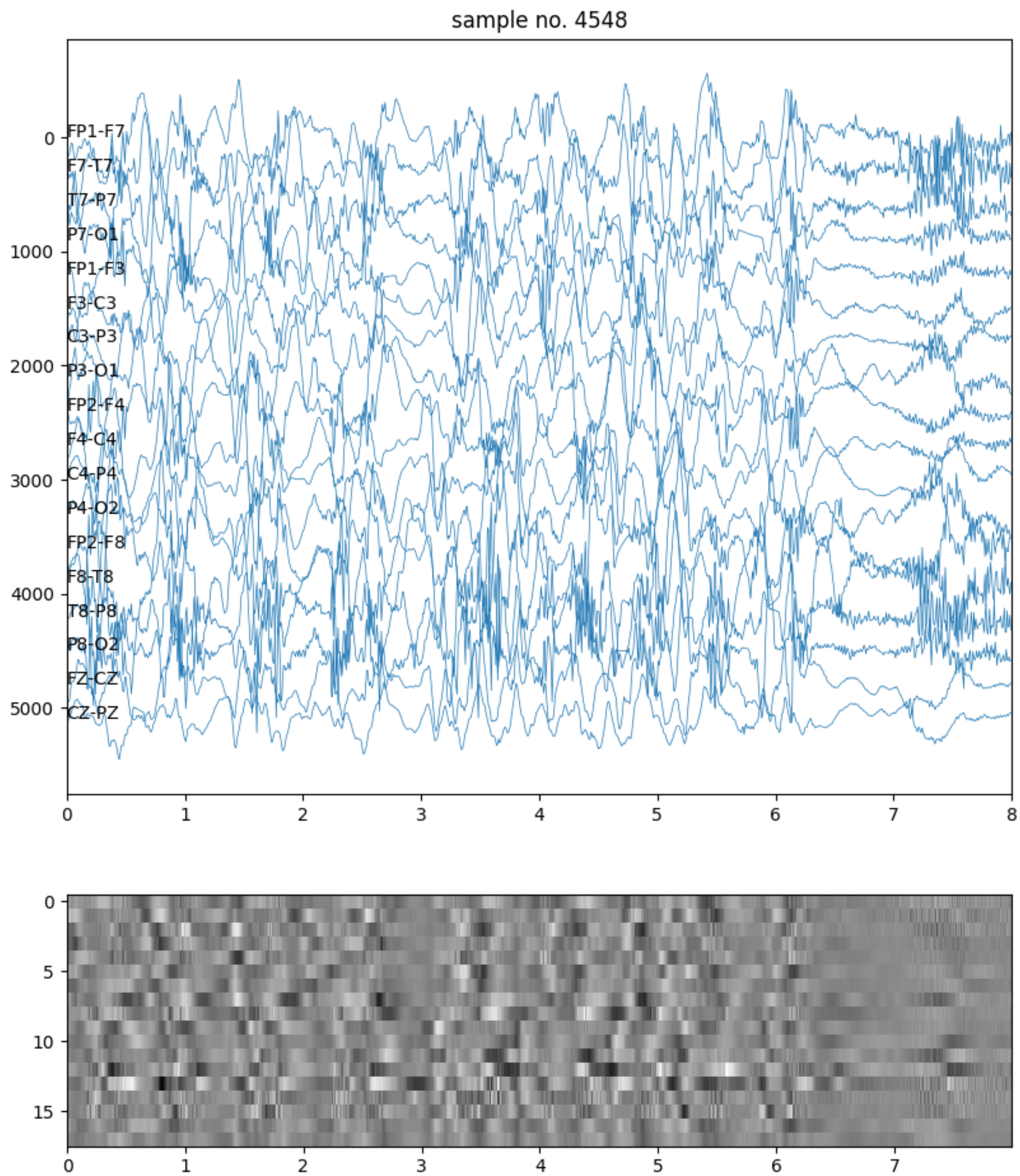


sample no. 5955
FP1-F7
F7-T7
T7-P7
P7-O1
FP1-F3
F3-C3
C3-P3
P3-O1
FP2-F4
F4-C4
C4-P4
P4-O2
FP2-F8
F8-T8
T8-P8
P8-O2
FZ-CZ
CZ-PZ
0
1000
2000
3000
4000
5000
0 1 2 3 4 5 6 7 8
0
5
10
15
0 1 2 3 4 5 6 7

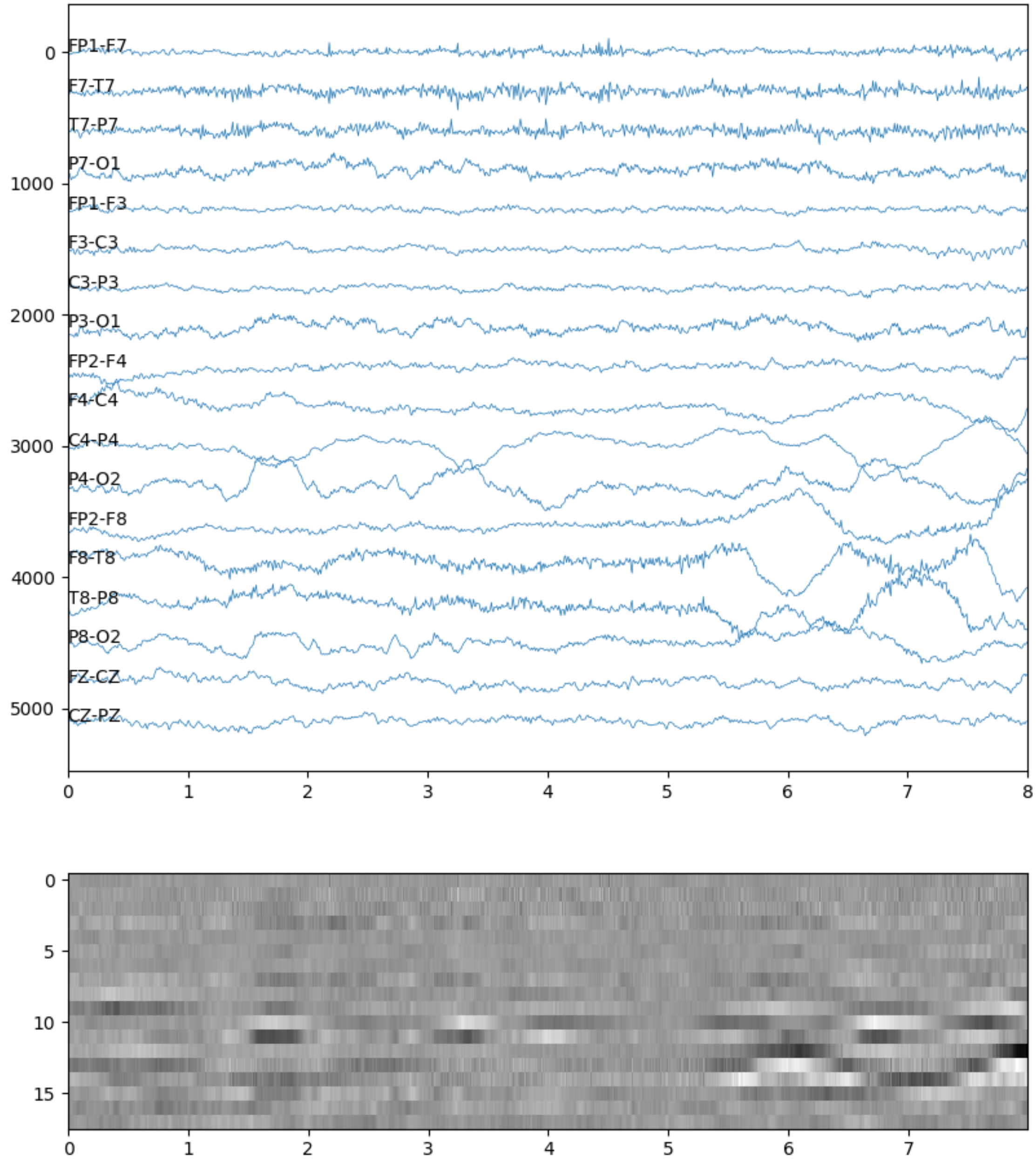
sample no. 6358



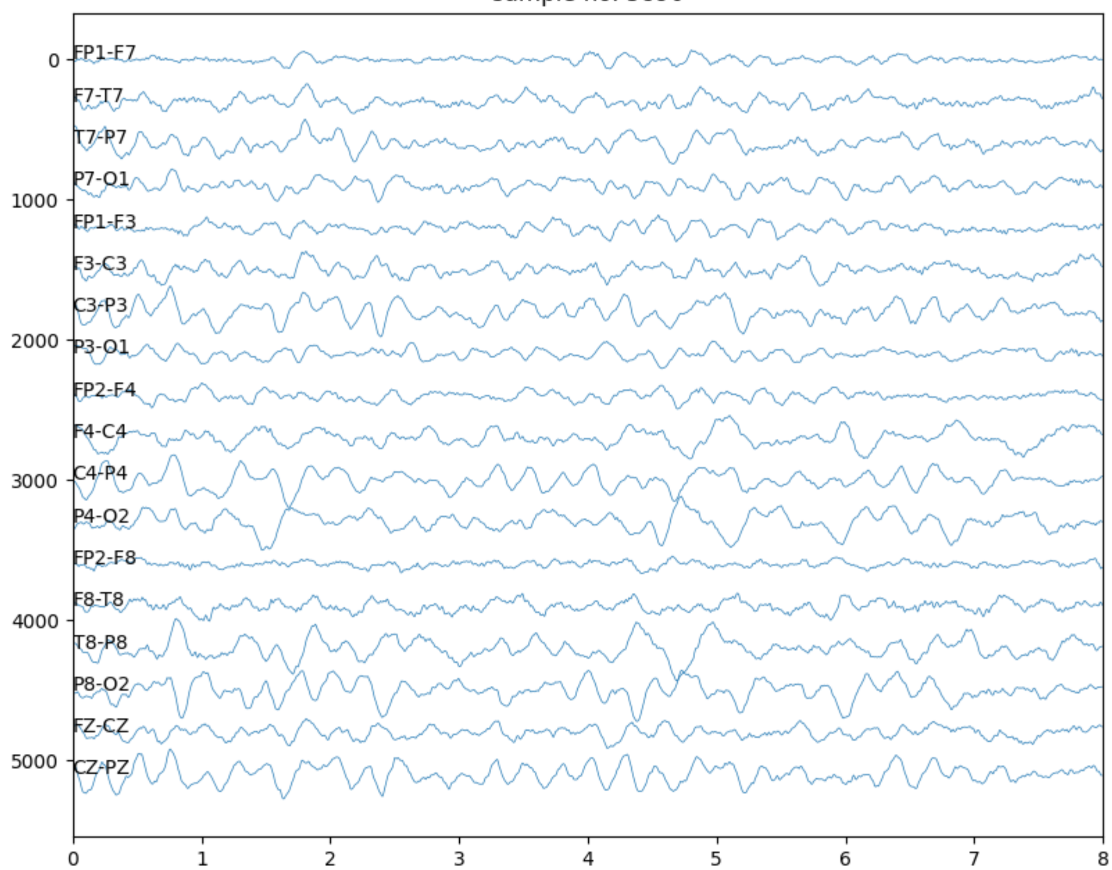


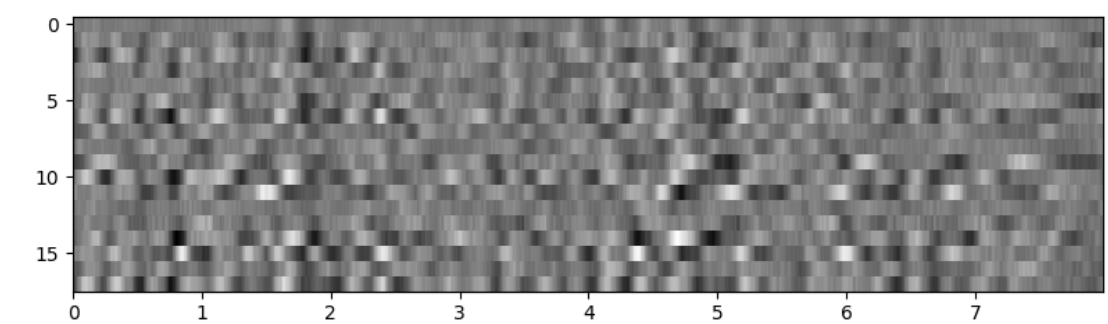

sample no. 4548
FP1-F7
F7-T7
T7-P7
P7-O1
FP1-F3
F3-C3
C3-P3
P3-O1
FP2-F4
F4-C4
C4-P4
P4-O2
FP2-F8
F8-T8
T8-P8
P8-O2
FZ-CZ
CZ-PZ

sample no. 8888

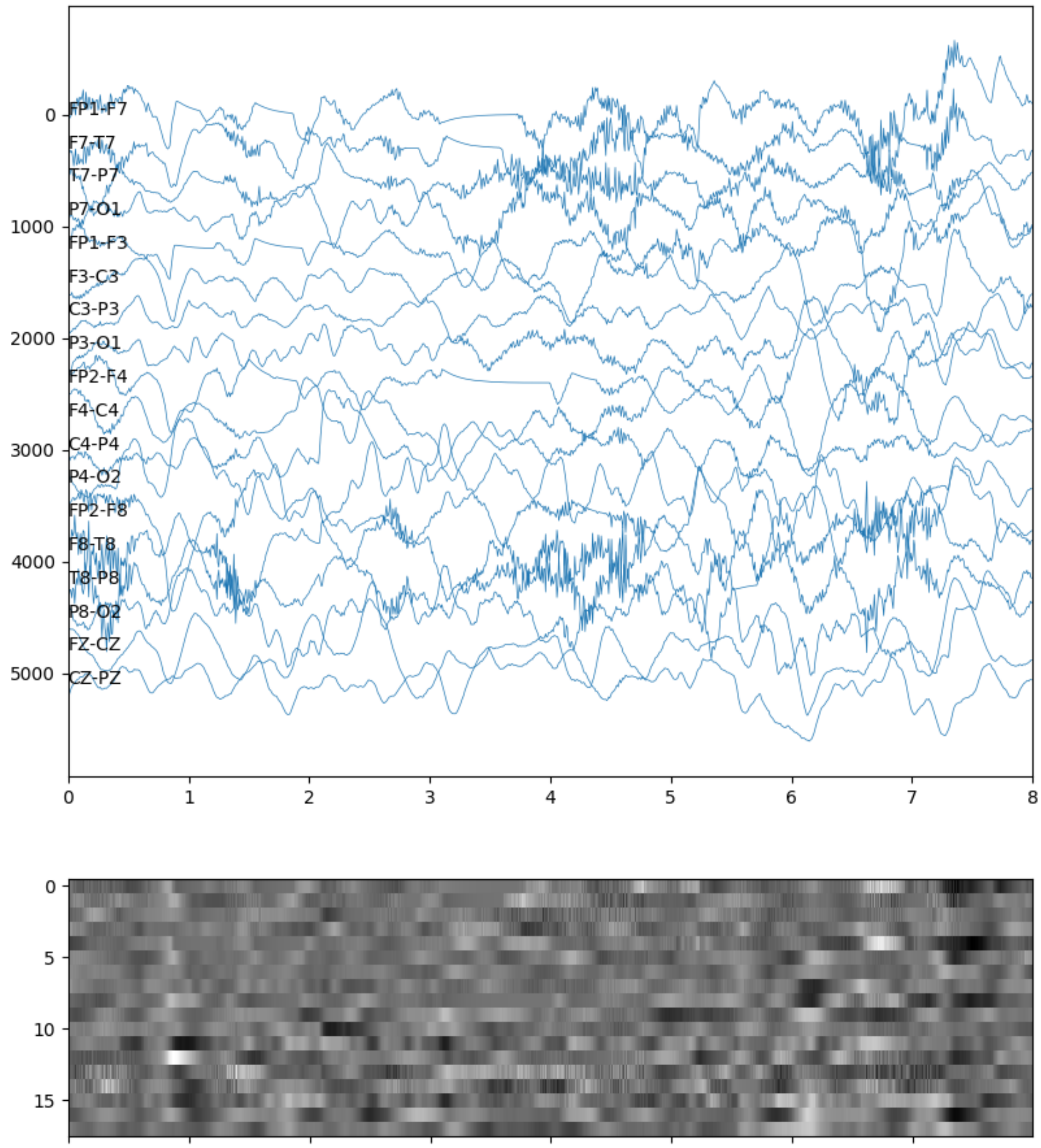


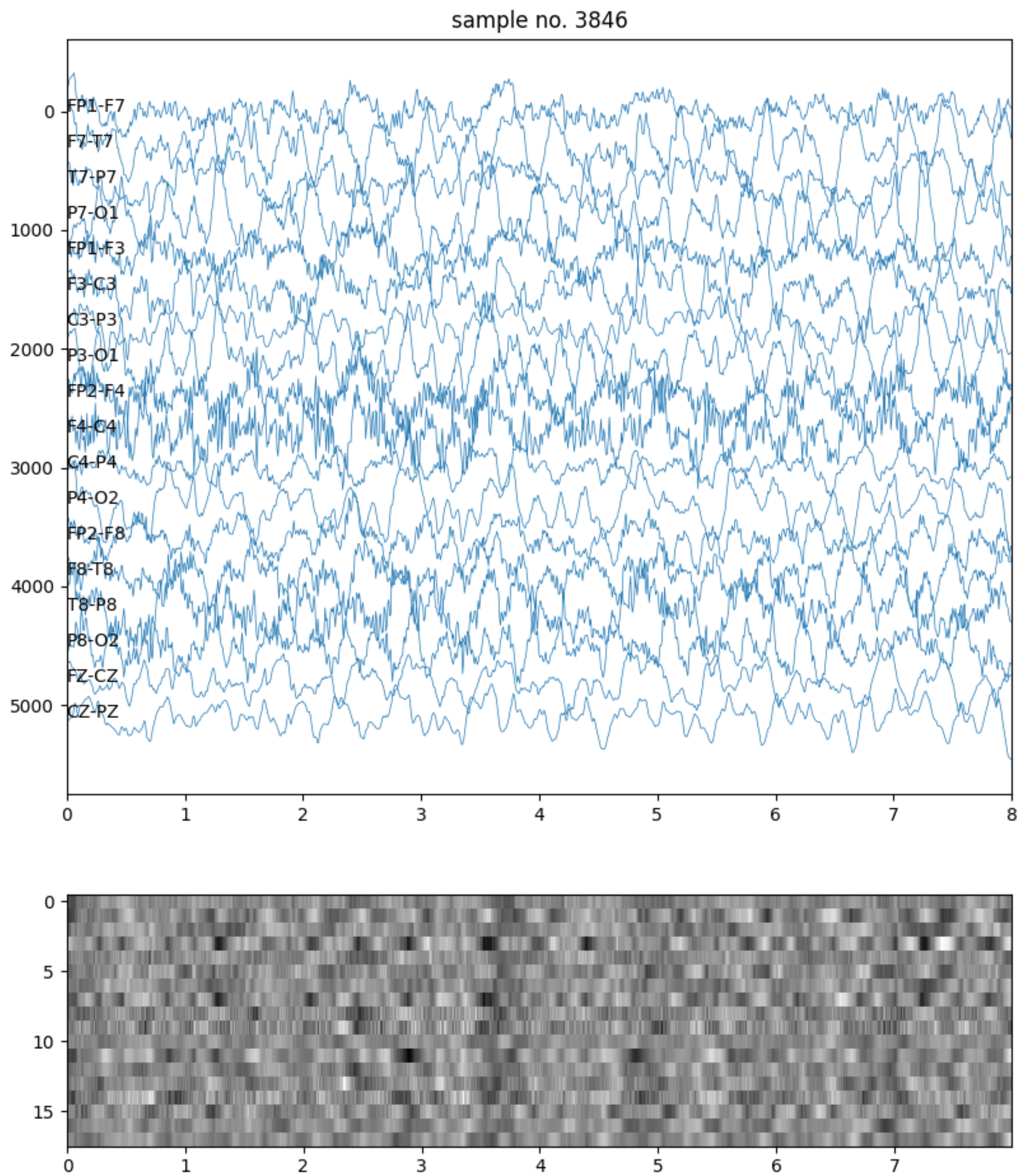
sample no. 5890

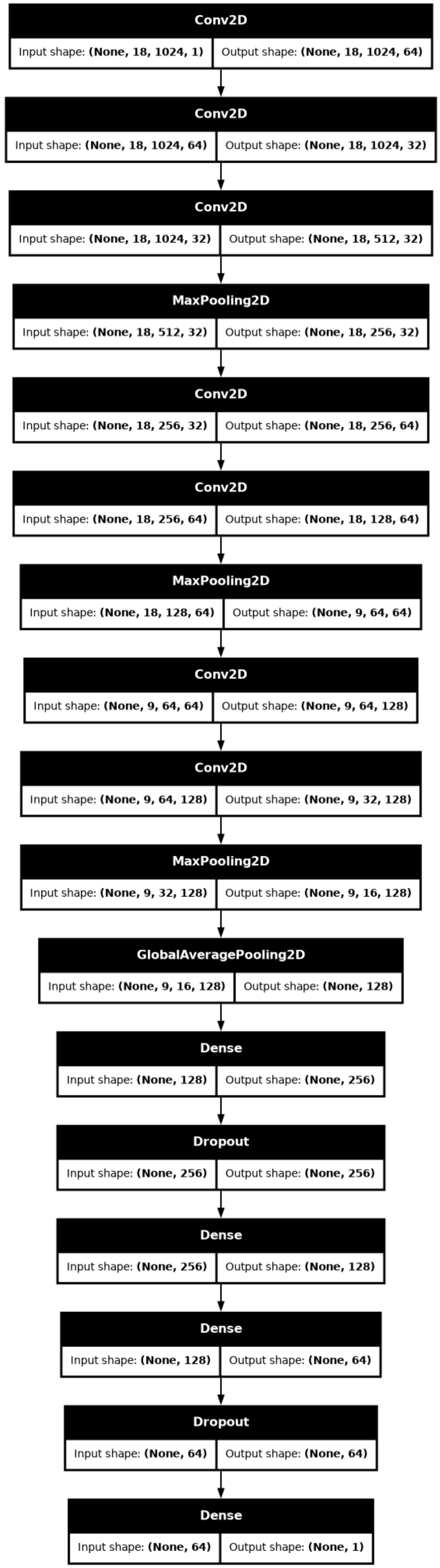




sample no. 3855




sample no. 3846
FP1-F7
F7-T7
T7-P7
P7-O1
FP1-F3
F3-C3
C3-P3
P3-O1
FP2-F4
F4-C4
C4-P4
P4-O2
FP2-F8
F8-T8
T8-P8
P8-O2
FZ-CZ
CZ-PZ


Conv2D
Input shape: (None, 18, 1024, 1)
Output shape: (None, 18, 1024, 64)
Conv2D
Input shape: (None, 18, 1024, 64)
Output shape: (None, 18, 1024, 32)
Conv2D
Input shape: (None, 18, 1024, 32)
Output shape: (None, 18, 512, 32)
MaxPooling2D
Input shape: (None, 18, 512, 32)
Output shape: (None, 18, 256, 32)
Conv2D
Input shape: (None, 18, 256, 32)
Output shape: (None, 18, 256, 64)
Conv2D
Input shape: (None, 18, 256, 64)
Output shape: (None, 18, 128, 64)
MaxPooling2D
Input shape: (None, 18, 128, 64)
Output shape: (None, 9, 64, 64)
Conv2D
Input shape: (None, 9, 64, 64)
Output shape: (None, 9, 64, 128)
Conv2D
Input shape: (None, 9, 64, 128)
Output shape: (None, 9, 32, 128)
MaxPooling2D
Input shape: (None, 9, 32, 128)
Output shape: (None, 9, 16, 128)
GlobalAveragePooling2D
Input shape: (None, 9, 16, 128)
Output shape: (None, 128)
Dense
Input shape: (None, 128)
Output shape: (None, 256)
Dropout
Input shape: (None, 256)
Output shape: (None, 256)
Dense
Input shape: (None, 256)
Output shape: (None, 128)
Dense
Input shape: (None, 128)
Output shape: (None, 64)
Dropout
Input shape: (None, 64)
Output shape: (None, 64)
Dense
Input shape: (None, 64)
Output shape: (None, 1)

super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 18, 1024, 64)	576
conv2d_1 (Conv2D)	(None, 18, 1024, 32)	16,416
conv2d_2 (Conv2D)	(None, 18, 512, 32)	8,224
max_pooling2d (MaxPooling2D)	(None, 18, 256, 32)	0
conv2d_3 (Conv2D)	(None, 18, 256, 64)	16,448
conv2d_4 (Conv2D)	(None, 18, 128, 64)	32,832
max_pooling2d_1 (MaxPooling2D)	(None, 9, 64, 64)	0
conv2d_5 (Conv2D)	(None, 9, 64, 128)	131,200
conv2d_6 (Conv2D)	(None, 9, 32, 128)	262,272
max_pooling2d_2 (MaxPooling2D)	(None, 9, 16, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 256)	33,024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dense_2 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Total params: 542,209 (2.07 MB)
Trainable params: 542,209 (2.07 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/20

Fig: Model summary

Code for prediction:

```
from scipy.signal import find_peaks
from sklearn import metrics
from typing import List
import matplotlib.pyplot as plt
import mne
import numpy as np
import os
import glob
import random
import re
import tqdm
import wfdb
from tensorflow.keras.models import load_model

# Define channel labels
ch_labels: List[str] = [
    "FP1-F7",
    "F7-T7",
    "T7-P7",
    "P7-O1",
    "FP1-F3",
    "F3-C3",
    "C3-P3",
    "P3-O1",
    "FP2-F4",
    "F4-C4",
    "C4-P4",
    "P4-O2",
    "FP2-F8",
    "F8-T8",
    "T8-P8",
    "P8-O2",
    "FZ-CZ",
    "CZ-PZ",
]

# Define base path to data
base_path: str = "/home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit/"


# Walk through the directory and print all filenames
def list_files_in_directory(path: str) -> None:
    for dirname, _, filenames in os.walk(path):
        for filename in filenames:
            print(os.path.join(dirname, filename))
```

```
list_files_in_directory(base_path)

# Gather folders and extract patient numbers
folders: List[str] = sorted(glob.glob(f"{base_path}/*/"))
n_patient: List[str] = [folder.rstrip("/").split("/")[-1][-2:] for folder in folders]

print("Patients:", *n_patient)

# Set seed for reproducibility
random.seed(2024)

# Split patients into train and test sets
# Patients are assigned to training and test groups through a random selection process.
ratio_train: float = 0.3
test_patient_str: List[str] = sorted(
    random.sample(n_patient, round(ratio_train * len(n_patient)))
)

print("Test PT:", *test_patient_str)


# Collect file paths for train and test sets
def collect_files(patient_str_list: List[str], path: str) -> List[str]:
    return [
        edf_file
        for patient in patient_str_list
        for edf_file in glob.glob(f"{path}/chb{patient}/*.edf")
    ]


files_test: List[str] = collect_files(test_patient_str, base_path)

print("Number of test files:", len(files_test))


# Model: CHB_MIT_sz_detec_demo.keras
model = load_model("CHB_MIT_sz_detec_demo.keras")

def sampling_data_pred(f, verbose=True):
    if verbose == True:
        print("{}: Reading. ".format(f))
    temp_edf = mne.io.read_raw_edf(f)
    temp_array_signals, temp_is_seizure_ind = np.array([]), np.array([])

    if sum(
        [
            any([0 if re.match(c, l) == None else 1 for l in temp_edf.ch_names])
```

```
            for c in ch_labels
        ]
    ) == len(ch_labels):
        ch_mapping = {
            sorted([l for l in temp_edf.ch_names if re.match(c, l) != None])[0]: c
            for c in ch_labels
        }
        temp_edf.rename_channels(ch_mapping)

        temp_is_seizure = np.zeros((temp_edf.n_times,))
# pyright: reportOperatorIssue=false
        temp_signals = temp_edf.get_data(picks=ch_labels) * 1e6

        if os.path.exists(f + ".seizures"):
            if verbose == True:
                print("seizure exists.", end=" ")
            temp_annotation = wfdb.rdann(f, "seizures")
            for i in range(int(temp_annotation.sample.size / 2)):
                temp_is_seizure[
                    temp_annotation.sample[i * 2] : temp_annotation.sample[i * 2 + 1]
                ] = 1

        temp_len = temp_edf.n_times

        time_window = 8
        time_step = 4
        fs = int(1 / (temp_edf.times[1] - temp_edf.times[0]))
        step_window = time_window * fs
        step = time_step * fs

        # sampling all signals
        temp_array_signals = np.array(
            [
                temp_signals[:, i * step : i * step + step_window]
                for i in range((temp_len - step_window) // step)
            ]
        )
        temp_is_seizure_ind = np.array(
            [
                temp_is_seizure[i * step : i * step + step_window].sum() / step_window
                for i in range((temp_len - step_window) // step)
            ]
        )
    else:
        if verbose == True:
            print("EEG {}: Not appropriate channel labels. Reading skipped.".format(n))

    return temp_array_signals, temp_is_seizure_ind
```

```
# reading files and prediction

list_pred = []
list_true = []

for f in tqdm.tqdm(files_test):
    array_signals, array_is_seizure = sampling_data_pred(f, verbose=False)
    array_signals = array_signals[:, :, ::2, np.newaxis]

    list_pred.append(model.predict(array_signals, verbose=0))
    list_true.append(array_is_seizure)


# threshold = 0.5
report = metrics.classification_report(
    np.concatenate(list_true) > 0, np.concatenate(list_pred) > 0.5
)
print(report)


# threshold = 0.9
report = metrics.classification_report(
    np.concatenate(list_true) > 0, np.concatenate(list_pred) > 0.9
)
print(report)

roc = metrics.roc_curve(np.concatenate(list_true) > 0, np.concatenate(list_pred))
auc = metrics.roc_auc_score(np.concatenate(list_true) > 0, np.concatenate(list_pred))

plt.figure(figsize=(4, 4))
plt.plot(
    roc[0][np.argmin(np.abs(roc[2] - 1)) :], roc[1][np.argmin(np.abs(roc[2] - 1)) :]
)
plt.xlabel("FPR: false positive rate")
plt.ylabel("TPR: true positive rate")
plt.title("ROC curve: AUC score = {:.2f}".format(auc))

th = [0.1, 0.2, 0.5, 0.9, 0.95, 1.0]
ind = [np.argmin(np.abs(roc[2] - l)) for l in th]
plt.scatter(roc[0][ind], roc[1][ind], s=15)
for i, l in enumerate(ind):
    plt.annotate("{}".format(th[i]), xy=(roc[0][l], roc[1][l]))
# plt.plot([0, 1, 1, 0, 0], [0, 0, 1, 1, 0], color='black', linewidth=1)
plt.ylim(-0.05, 1.05)
plt.xlim(-0.05, 1.05)
plt.grid()
# plt.axis('off')
plt.show()
```

```
for i, f in enumerate(files_test):
    if os.path.exists(f + ".seizures"):
        print("Index = {} has seizures: {}".format(i, f))


def moving_ave(a, n):
    if len(a.shape) != 1:
        print("Not 1 dimension array. return nothing.")
        return
    temp = np.zeros(a.size - n)
    for i in range(n):
        temp = temp + a[i : -n + i]
    temp = temp / n

    return temp


# get signals and labels from test data.
n = 100
array_signals, array_is_seizure = sampling_data_pred(files_test[n])

# preprocess
array_signals = array_signals[:, :, ::2, np.newaxis]

if model is None:
    print("Model not found. Exit.")
    exit()

# use deep learning model
pred = model.predict(array_signals)


time_window = 8
time_step = 4
mv_win = 3

fig, ax = plt.subplots(figsize=(12, 2))

ax.plot(
    np.arange(pred.size) * time_step,
    pred.flatten(),
    alpha=0.7,
    label="deep learning model pred",
)
ax.plot(np.arange(pred.size) * time_step, array_is_seizure, alpha=0.7, label="True label")

pred_moving_ave = moving_ave(pred.flatten(), mv_win)
pred_peaks, _ = find_peaks(pred_moving_ave, height=0.95, distance=6)
```

```
ax.plot(
    np.arange(pred.size - mv_win) * time_step,
# pyright: reportArgumentType=false
    pred_moving_ave,
    alpha=0.9,
    label="pred - moving ave",
    color="tab:pink",
    zorder=0,
)

# pyright: reportOptionalSubscript=false
ax.scatter(pred_peaks * time_step, pred_moving_ave[pred_peaks], s=20, color="tab:red")

ax.set_xlabel("time (s)")
ax.set_ylabel("p")
ax.set_xlim(0, pred.size * time_step + 500)
ax.legend(loc="upper right")
plt.show()

if pred_peaks.size == 0:
    print("No seizure detected.")
else:
    f = files_test[n]
    temp_edf = mne.io.read_raw_edf(f)
    temp_labels = temp_edf.ch_names
    temp_signals = None

    if sum(
        [
            any([0 if re.match(c, l) == None else 1 for l in temp_edf.ch_names])
            for c in ch_labels
        ]
    ) == len(ch_labels):
        ch_mapping = {
            sorted([l for l in temp_edf.ch_names if re.match(c, l) != None])[0]: c
            for c in ch_labels
        }
        temp_edf.rename_channels(ch_mapping)
        # temp_edf = temp_edf.pick(ch_labels)

        temp_is_seizure = np.zeros((temp_edf.n_times,))
        temp_signals = temp_edf.get_data(picks=ch_labels) * 1e6

    fs = int(1 / (temp_edf.times[1] - temp_edf.times[0]))
    for n_peak in range(pred_peaks.size):
        ind_peak = pred_peaks[n_peak] * time_step * fs
        backward_steps = 30 * fs
        forward_steps = 15 * fs
        vertical_width = 500
```

```
    fig, ax = plt.subplots(figsize=(10, 6))

    if temp_signals is not None:
        for i in range(temp_signals.shape[0]):
            ax.plot(
                np.arange(ind_peak - backward_steps, ind_peak + forward_steps) / fs,
                temp_signals[i, ind_peak - backward_steps : ind_peak + forward_steps]
                + i * vertical_width,
                linewidth=0.5,
                color="tab:blue",
            )
            ax.annotate(
                ch_labels[i], xy=((ind_peak - backward_steps) / fs, i * vertical_width)
            )
        ax.axvline(
            x=ind_peak / fs, color="tab:red", alpha=0.5, label="Seizure detection point"
        )
        ax.invert_yaxis()
        ax.legend(loc="upper right")
        plt.show()
# ax.set_xlim(0, 8)

temp_edf.close()
```

100%|

	precision	recall	f1-score	support
False	1.00	0.97	0.98	293126
True	0.05	0.93	0.10	502
accuracy			0.97	293628
macro avg	0.53	0.95	0.54	293628
weighted avg	1.00	0.97	0.98	293628

	precision	recall	f1-score	support
False	1.00	0.99	1.00	293126
True	0.17	0.82	0.28	502
accuracy			0.99	293628
macro avg	0.58	0.91	0.64	293628
weighted avg	1.00	0.99	1.00	293628

Fig: model metrics summary

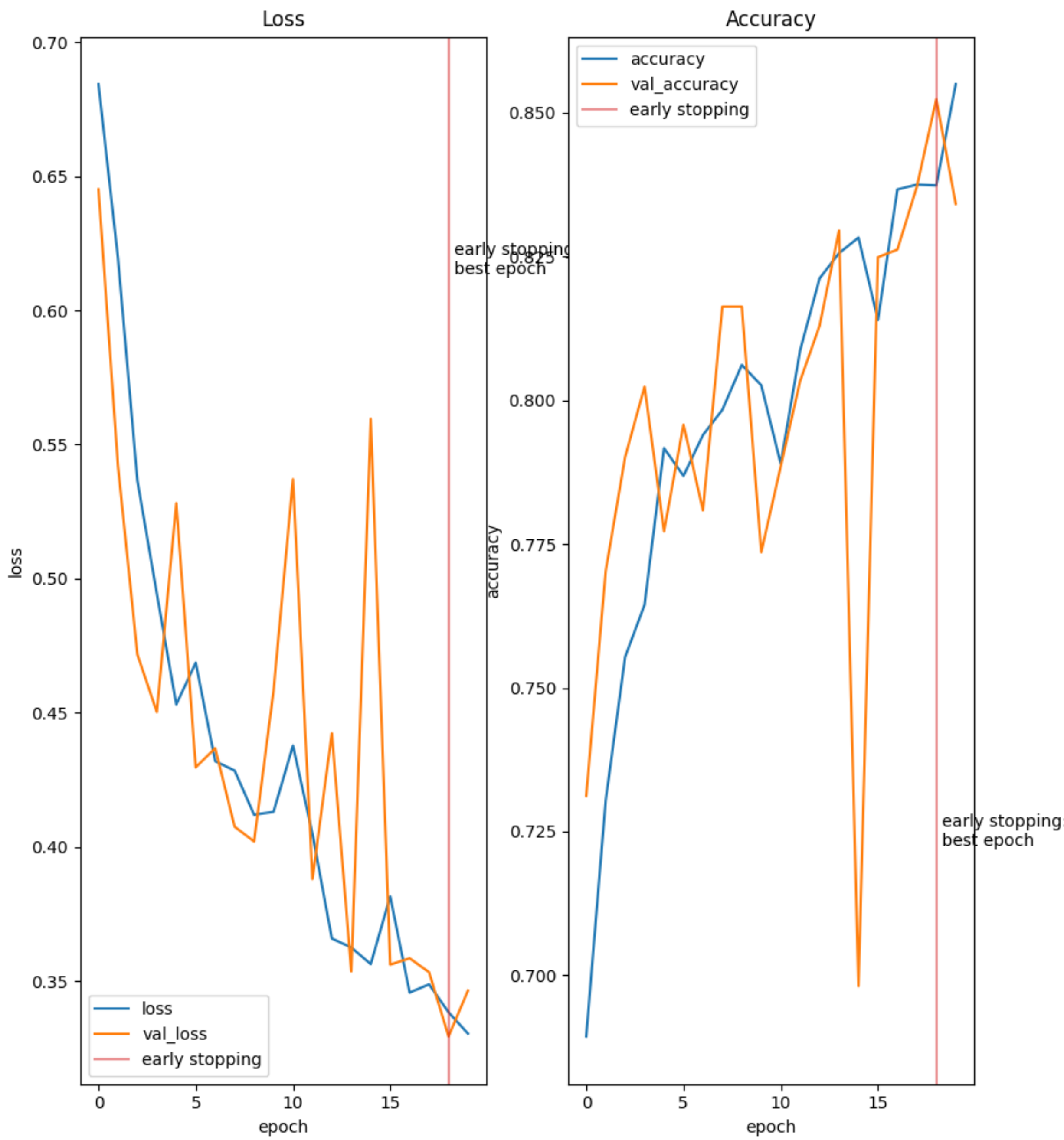


Fig: Loss and Accuracy graph

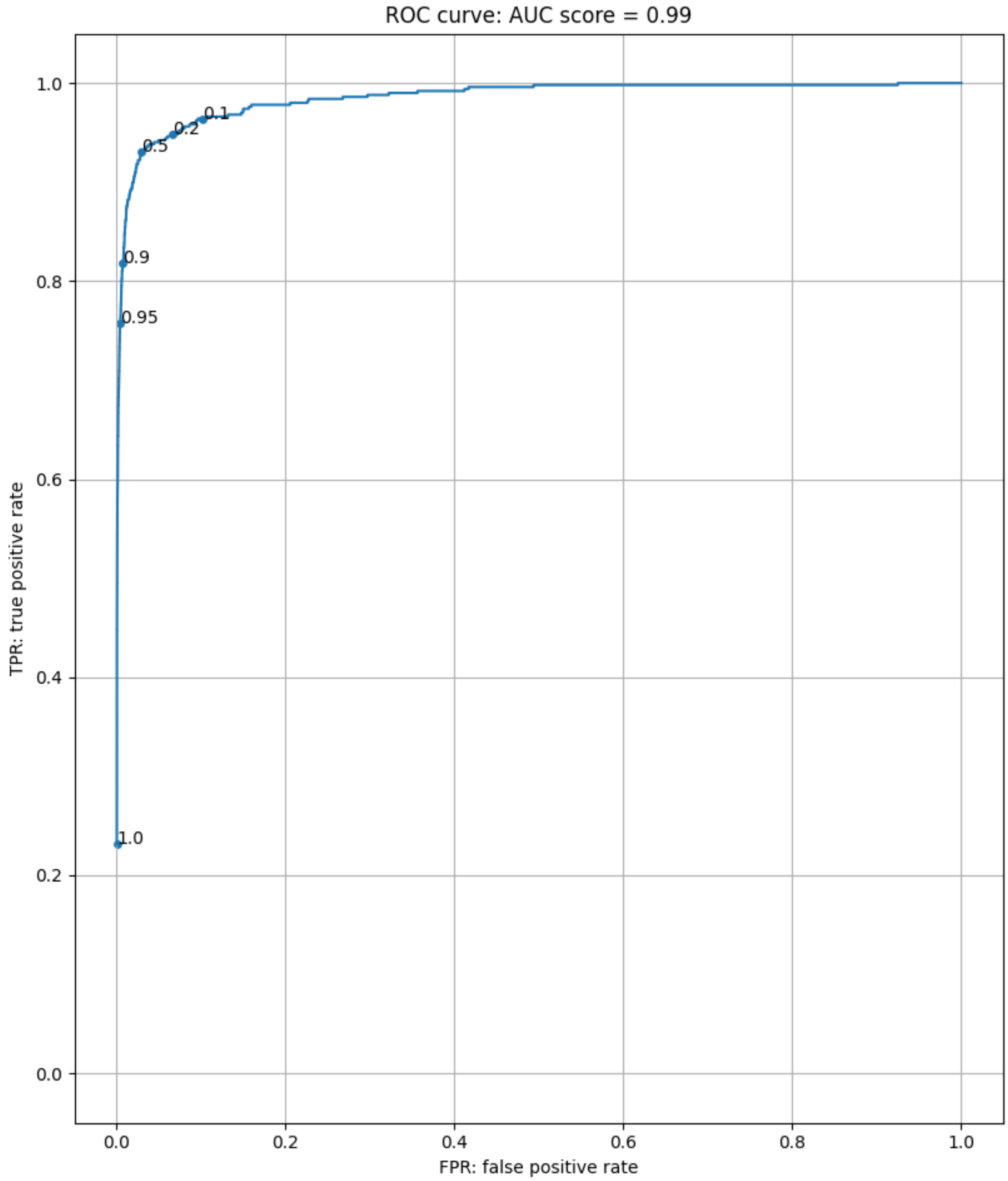
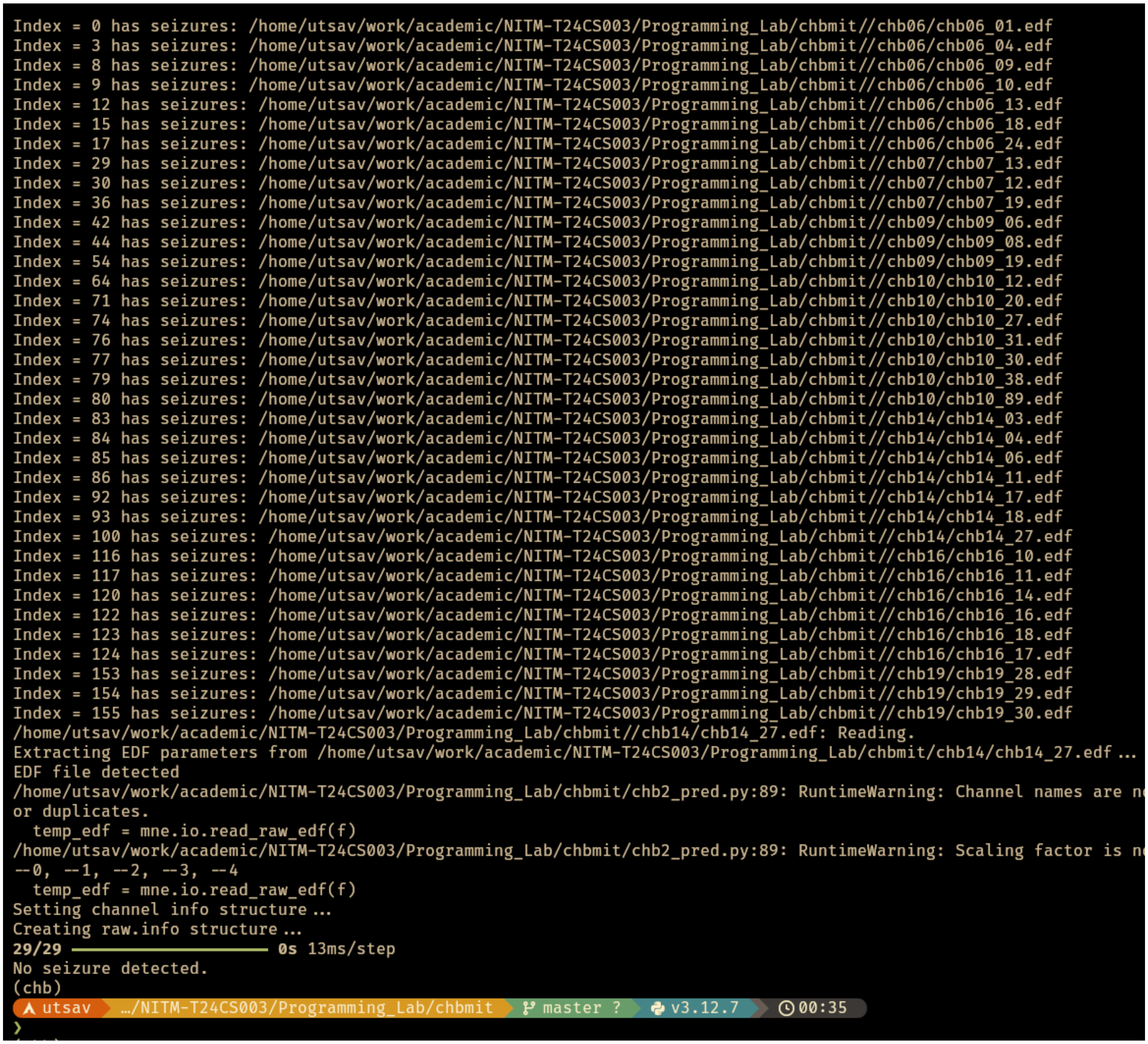


Fig: ROC curve with AUC score



```
Index = 0 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb06/chb06_01.edf
Index = 3 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb06/chb06_04.edf
Index = 8 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb06/chb06_09.edf
Index = 9 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb06/chb06_10.edf
Index = 12 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb06/chb06_13.edf
Index = 15 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb06/chb06_18.edf
Index = 17 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb06/chb06_24.edf
Index = 29 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb07/chb07_13.edf
Index = 30 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb07/chb07_12.edf
Index = 36 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb07/chb07_19.edf
Index = 42 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb09/chb09_06.edf
Index = 44 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb09/chb09_08.edf
Index = 54 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb09/chb09_19.edf
Index = 64 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb10/chb10_12.edf
Index = 71 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb10/chb10_20.edf
Index = 74 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb10/chb10_27.edf
Index = 76 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb10/chb10_31.edf
Index = 77 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb10/chb10_30.edf
Index = 79 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb10/chb10_38.edf
Index = 80 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb10/chb10_89.edf
Index = 83 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb14/chb14_03.edf
Index = 84 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb14/chb14_04.edf
Index = 85 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb14/chb14_06.edf
Index = 86 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb14/chb14_11.edf
Index = 92 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb14/chb14_17.edf
Index = 93 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb14/chb14_18.edf
Index = 100 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb14/chb14_27.edf
Index = 116 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb16/chb16_10.edf
Index = 117 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb16/chb16_11.edf
Index = 120 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb16/chb16_14.edf
Index = 122 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb16/chb16_16.edf
Index = 123 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb16/chb16_18.edf
Index = 124 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb16/chb16_17.edf
Index = 153 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb19/chb19_28.edf
Index = 154 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb19/chb19_29.edf
Index = 155 has seizures: /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb19/chb19_30.edf
/home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit//chb14/chb14_27.edf: Reading.
Extracting EDF parameters from /home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit/chb14/chb14_27.edf ...
EDF file detected
/home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit/chb2_pred.py:89: RuntimeWarning: Channel names are n
or duplicates.
  temp_edf = mne.io.read_raw_edf(f)
/home/utsav/work/academic/NITM-T24CS003/Programming_Lab/chbmit/chb2_pred.py:89: RuntimeWarning: Scaling factor is n
--0, --1, --2, --3, --4
  temp_edf = mne.io.read_raw_edf(f)
Setting channel info structure ...
Creating raw.info structure ...
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
No seizure detected.
(chb)
 utsav  …/NITM-T24CS003/Programming_Lab/chbmit  master ?  v3.12.7  00:35
❯
```

Fig: Prediction results of the model