# EEG Classification using Convolutional Neural Networks

## Project group 30

**Francisco Chavez Gonzalez**

**Xueshi Bai**

**Santosh Kumar Kushal Yelamandala**

**Nixon Lobo**

**Pruthviraj Chintakindi**

—

Course: **Foundations for Data Analytics Engineering**

—

Professor: *Sivarit (Tony) Sultornsanee*

# Introduction and background information

Electroencephalography (EEG) is a non-invasive method to record electrical activity of the brain. It is widely used in both research and clinical settings for its ability to provide real-time monitoring of neural activity. The classification of EEG signals involves the identification and categorization of distinct patterns within these signals, which can correspond to specific brain states, responses to stimuli, or indicators of neurological disorders.

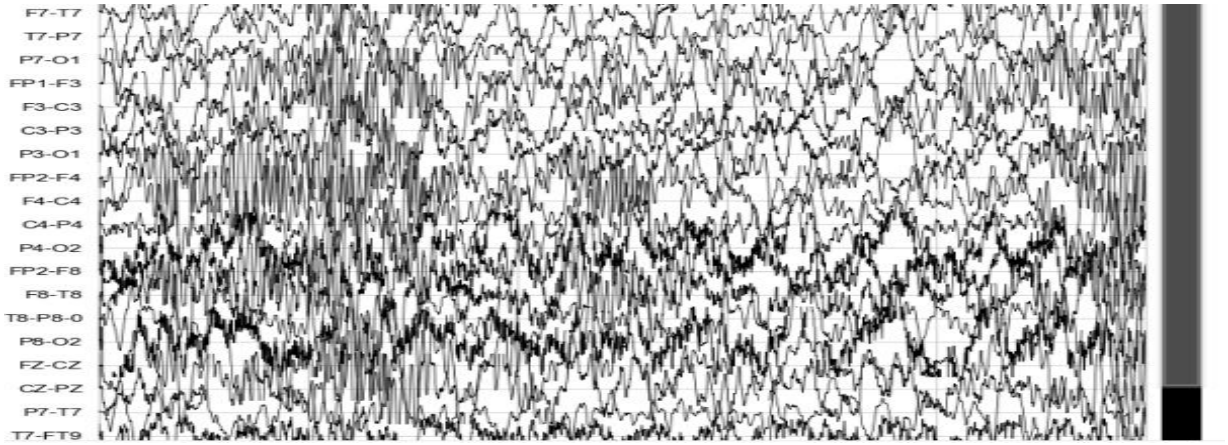
**Data resource: CHB-MIT EEG Database**
Because this topic is relatively new to us, in order to simplify the process we only selected a section of EDF data that includes the onset of disease and the absence of disease.

**Data Processing & Load EDF File**

```
import mne

# Path to your EEG data file
eeg_file = 'C:/neu/IE6400/chb01_03.edf'

# Reading EEG data
raw = mne.io.read_raw_edf(eeg_file, preload=True)
```

**View MNE Data Raw**

```
raw.plot()
```




## Extract the seizure segment and non-seizure segment

CHB01 summary file tells the seizure starting time (2996 seconds) and seizure ending time (3036 seconds). We also labelled the seizure segment as 1 and non-seizure segment as 0 for later EEG classification.

```
# Define the start and end times of the seizure in seconds
seizure_start = 2996
seizure_end = 3036

# Extract the seizure data
seizure_data = raw.copy().crop(tmin=seizure_start, tmax=seizure_end)
seizure_label = 1
```

Since we cut in the middle, we have two non-seizure segments. Label both of them as 0.

```
# Extract no seizure data
no_seizure_data_1 = raw.copy().crop(tmin=0, tmax=seizure_start)
no_seizure_data_2 = raw.copy().crop(tmin=seizure_end, tmax=None)
no_seizure_label = 0
```

Executed at 2023.12.15 14:46:17 in 150ms

Concatenate them into a single raw object.

```
# Concatenate no_seizure_raw1 to no_seizure_raw2
no_seizure_data = mne.concatenate_raws([no_seizure_data_1, no_seizure_data_2])
```

Executed at 2023.12.15 15:11:26 in 82ms

## Features Extraction

This step involves extracting features from both seizure and no-seizure segments of EEG data using MNE-Python. Two Raw objects (one for seizure and one for no seizure) follow a similar approach. These features can be time-domain features, frequency-domain features, or a combination of both.

Time-Domain Features: Mean, standard deviation, variance, skewness, kurtosis, zero-crossing rate.

```
def extract_time_domain_features(data):
    return {
        'mean': np.mean(data, axis=1),
        'std': np.std(data, axis=1),
        'variance': np.var(data, axis=1),
        'skewness': skew(data, axis=1),
        'kurtosis': kurtosis(data, axis=1)
    }
```

Frequency-Domain Features: Power spectral density, band power. Calculate PSD for Each Channel and correctly Index Frequency Bands

```
def extract_frequency_domain_features(data, sfreq):
    # Calculate PSD for each channel
    psd_list = []
    for channel_data in data:
        freqs, psd = welch(channel_data, sfreq, nperseg=4*sfreq)
        psd_list.append(psd)
    psd_array = np.array(psd_list)

    # Extract features from specific frequency bands
    alpha_band = (8, 12)
    idx_alpha = np.logical_and(freqs >= alpha_band[0], freqs <= alpha_band[1])

    # Calculate band power
    alpha_power = np.trapz(psd_array[:, idx_alpha], freqs[idx_alpha], axis=1)

    return {'alpha_power': alpha_power}
```

Combine the extracted features into a single dataset, ensuring that each feature vector is appropriately labeled as seizure (1) or non-seizure (0). Finally, combine the features into one data frame

```
# Extract features
seizure_features_df = extract_features(seizure_data)
no_seizure_features_df = extract_features(no_seizure_data)

# Add labels for classification
seizure_features_df['label'] = 1  # Seizure
no_seizure_features_df['label'] = 0  # No seizure

# Combine the features into one DataFrame
combined_features_df = pd.concat([seizure_features_df, no_seizure_features_df], ignore_index=True)
```

## Model Selection and Model Training

The strategy is to use a 80-10-10 split. The training set is used for model training, the validation set for model tuning and performance validation, and the test set for evaluating the model's performance on unseen data.

1. Combine and Shuffle the Data: First, combine the seizure and no-seizure data, and shuffle them to ensure a good mix. It's important to shuffle the data to avoid any biases that might arise from the order in which data was collected.
2. Split the Data: Split the combined dataset into training, validation, and test sets.

```
from sklearn.model_selection import train_test_split

# Separate features and labels
X = combined_features_df.drop('label', axis=1)
y = combined_features_df['label']

# Split the data into 80% training and 20% remaining
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Split the remaining 20% into half validation and half test
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)
```

Executed at 2023.12.15 15:48:45 in 9ms

We use Convolutional Neural Networks (CNNs) for EEG classification. CNNs are known for their ability to automatically and effectively extract features from data. In the context of EEG signals, CNNs can identify and learn spatial and temporal features that are significant for classification tasks without the need for manual feature engineering.

1. Define the CNNs model
   Match the Output Layer to the Label Format:
   Since our labels are encoded as binary (0 or 1), we built the model to have a single output neuron with a sigmoid activation function.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

def build_cnn_1d_binary(input_shape):
    model = Sequential()

    model.add(Conv1D(32, 2, activation='relu', input_shape=input_shape))
    model.add(Dropout(0.25))

    model.add(Conv1D(64, 2, activation='relu'))
    model.add(MaxPooling1D(1))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    # Adjusted for binary classification
    model.add(Dense(1, activation='sigmoid'))

    return model
```

2. Compile the model
   Choose an optimizer, loss function, and metrics. Compile the model.

```python
input_shape = (6, 1)  # Assuming 6 features per sample

model = build_cnn_1d_binary(input_shape)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Executed at 2023.12.15 15:56:02 in 79ms

3. Train the model
   Train the model using the training data, , while validating using the validation data.

```python
# Training the model
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_val, y_val))
```

Executed at 2023.12.15 15:56:34 in 2s 178ms

## Evaluation results and discussion

We evaluate the model's performance on the validation set. The evaluation metrics for the classification task include accuracy, precision, recall, and F1-score to gauge the model's ability to correctly classify seizure and no-seizure instances.
First, make predictions on the validation set, and then compute the metrics.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Make predictions on the validation set
y_val_pred = model.predict(X_val)
y_val_pred = (y_val_pred > 0.5).astype(int)  # Convert probabilities to binary predictions

# Compute the metrics
accuracy = accuracy_score(y_val, y_val_pred)
precision = precision_score(y_val, y_val_pred)
recall = recall_score(y_val, y_val_pred)
f1 = f1_score(y_val, y_val_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

Executed at 2023.12.15 15:58:25 in 190ms

```
1/1 [==============================] - 0s 108ms/step
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
```

## Assess the final model on the test set

1. Make Predictions on the Test Set

```
y_test_pred = model.predict(X_test)
y_test_pred = (y_test_pred > 0.5).astype(int)  # Convert probabilities to binary predictions
```

Executed at 2023.12.15 16:04:00 in 90ms

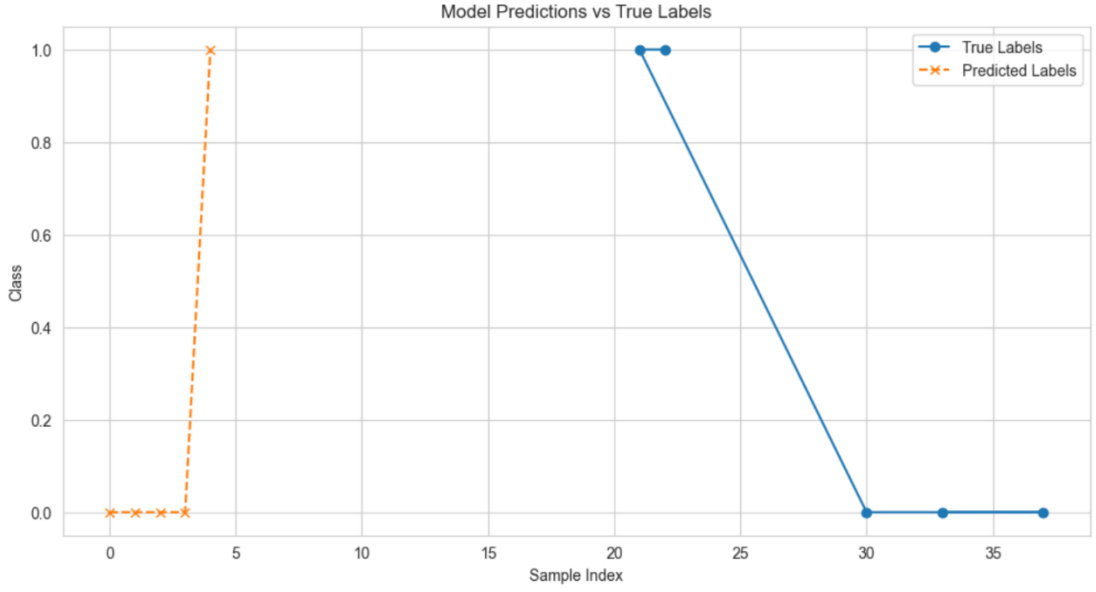
2. Compute Key Metrics

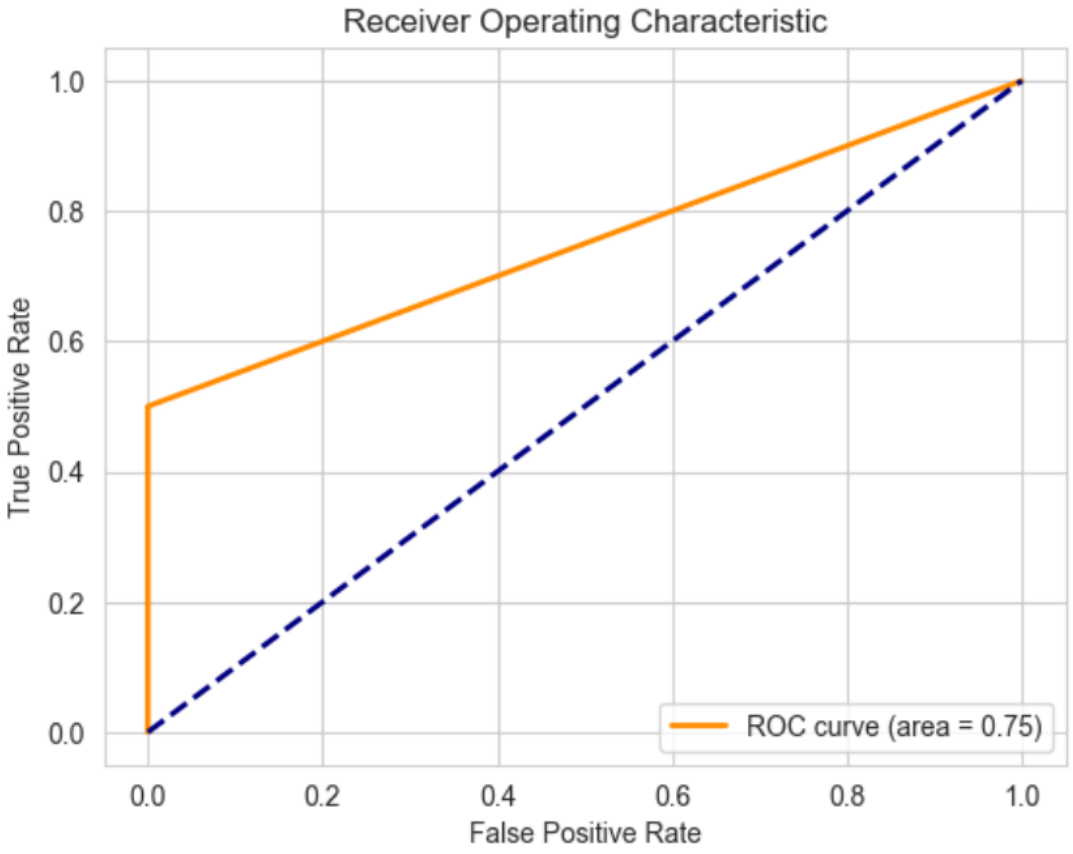
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test Precision: {precision:.4f}")
print(f"Test Recall: {recall:.4f}")
print(f"Test F1-Score: {f1:.4f}")

```

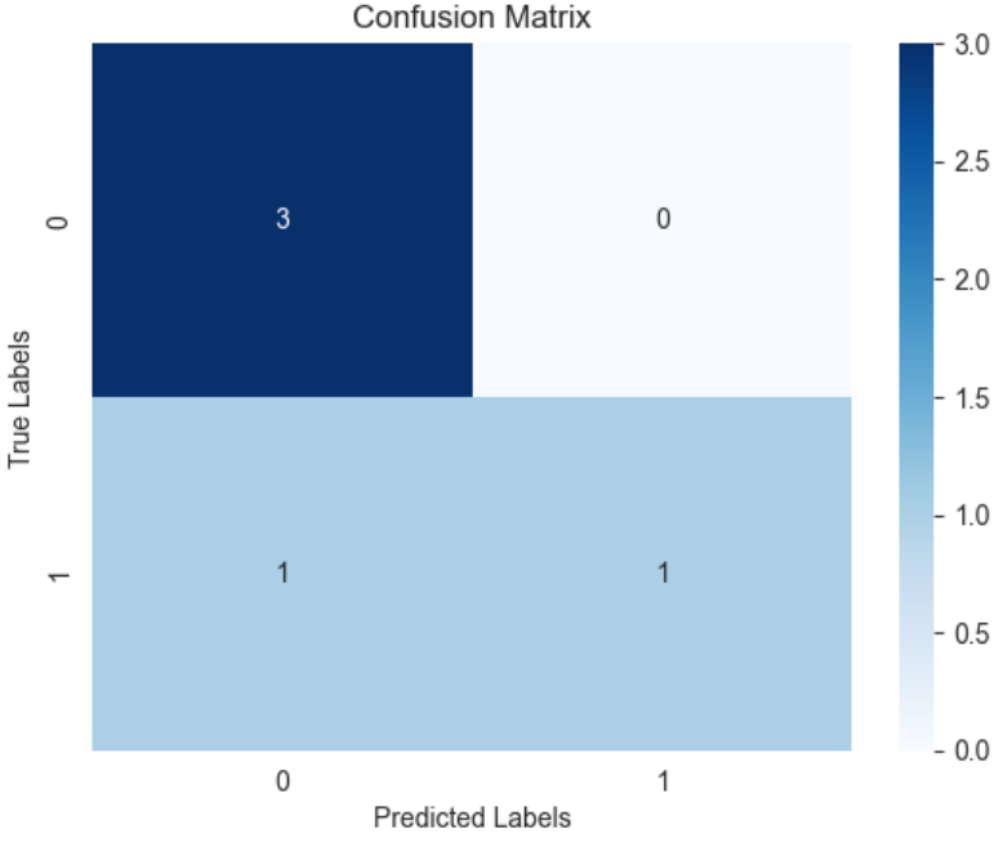
Executed at 2023.12.15 16:04:12 in 106ms

```
Test Accuracy: 0.8000
Test Precision: 1.0000
Test Recall: 0.5000
Test F1-Score: 0.6667
```

## Prediction Visualization




Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.75)


Confusion Matrix
True Labels
Predicted Labels
3
0
1
1
0
1
3.0
2.5
2.0
1.5
1.0
0.5
0.0

## Conclusion and future work

Overall, this project demonstrated a comprehensive approach to EEG data analysis for seizure detection, from data preprocessing and feature extraction to machine learning model development and evaluation. The whole process includes loading and preprocessing EEG data, feature extraction, preparing data for machine learning, training the CNN model, model evaluation, final assessment on test set, and result visualizations. The use of CNNs, in conjunction with careful preprocessing and evaluation strategies, highlighted the potential of machine learning techniques in distinguishing complex patterns in EEG signals, such as those differentiating seizure and no-seizure states.

Our future work includes solving model overfitting which happens when the model learns the training data too well, including noise and outliers, and thus performs exceptionally well on the training and validation data but poorly on unseen data. We can check the model's performance on a completely separate test set that was not used during training or validation. We also aim to expanding the dataset by incorporating additional EEG recordings. More data can help the model learn a broader range of patterns and nuances, especially those that are less frequent.