GROUP PROJECT-3

Project Report: EEG Classification Model

Course: IE6400 Foundations Data Analytics Engineering

Fall Semester 2023

Group Number - 29

Sri Sai Prabhath Reddy Gudipalli (002207631)

Korukonda Raghavinisha (002832571)

Deepthi Umesha (002661626)

Ashwini Mahadevaswamy (002661627)

Yashwant Dontam (002844794)

Introduction

This project aims to advance the understanding and diagnosis of epilepsy through the detailed analysis of EEG signals. By differentiating between epileptic and non-epileptic states, as well as between ictal (seizure) and non-ictal periods, the study employs sophisticated data analysis and machine learning techniques. Our approach is designed to accurately identify seizure activity, offering critical insights that can aid in clinical diagnosis and treatment strategies. This report presents our methodology, explores our analytical techniques, and discusses the implications of our findings for epilepsy monitoring and patient care.

Task 1: Data Preprocessing

[22]:
```python
import os
import pandas as pd

dataset_path = "C:/Users/prabh/Desktop/PJ3"

# Initialize empty lists to store data and labels
data = []
labels = []

# Iterate through each folder (Z, O, N, F, S)
for folder_name in ["Z", "O", "N", "F", "S"]:
    folder_path = os.path.join(dataset_path, folder_name)

    # Iterate through each file in the folder
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)

        # Read EEG data from the text file
        with open(file_path, 'r') as file:
            eeg_data = file.read().splitlines()

        # Append the EEG data to the 'data' list
        data.append(eeg_data)

        # Append the label (folder_name) to the 'labels' list
        labels.append(folder_name)

# Create a DataFrame from the lists
df = pd.DataFrame({'Data': data, 'Label': labels})

# Display the DataFrame
print(df.head())
```

```
                                                Data Label
0  [12, 22, 35, 45, 69, 74, 79, 78, 66, 43, 33, 3...     Z
1  [-56, -50, -64, -91, -135, -140, -134, -114, -...     Z
2  [-37, -22, -17, -24, -31, -20, -5, 14, 31, 31,...     Z
3  [-31, -43, -39, -39, -9, -5, 18, 7, -12, -42, ...     Z
4  [14, 26, 32, 25, 16, 8, 8, 12, 11, 19, 23, 24,...     Z
```

This Python code reads EEG data stored in text files located within specific folders and organizes it into a Pandas DataFrame. It iterates through folders labeled "Z," "O," "N," "F," and "S" within a specified base path. For each file within these folders, it reads the EEG data and appends it to the 'data' list, along with

the corresponding folder name ('labels'). The code then constructs a DataFrame with two columns: 'Data' containing the EEG data and 'Label' containing the folder names. This DataFrame structure allows for convenient data organization and analysis. Finally, it displays the initial rows of the DataFrame for a quick overview of the data.

```python
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Convert labels to numerical values using LabelEncoder
le = LabelEncoder()
df['Label'] = le.fit_transform(df['Label'])

# Pad sequences to ensure uniform length
max_sequence_length = max(len(seq) for seq in df['Data'])
padded_sequences = pad_sequences(df['Data'], maxlen=max_sequence_length, padding='post', dtype='float32')

# Standardize the data using StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(np.array(padded_sequences).reshape(-1, max_sequence_length))
```

This Python code serves as a crucial data preprocessing step for machine learning and deep learning applications. It begins by encoding categorical labels into numerical values using the `LabelEncoder`, facilitating the use of these labels in predictive models. To ensure uniformity in sequence data, it calculates the maximum sequence length within the 'Data' column of the Pandas DataFrame and pads sequences with `pad_sequences`. Standardization of the data follows using `StandardScaler`, making all features have zero mean and unit variance, enhancing the training process. Finally, the code reshapes the data into a 2D array, preparing it for input into machine learning or deep learning models. These transformations collectively enhance the data's suitability for a wide range of predictive modeling tasks by encoding labels, ensuring consistent sequence lengths, and standardizing features.

Task 2: Data Splitting

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(scaled_data, df['Label'], test_size=0.2, random_state=42)

# Display the processed data
print("Processed EEG Data:")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

The code essentially divides the dataset into training and testing sets for model training and evaluation. It also prints the shapes of the resulting sets for verification and understanding. The random_state parameter ensures that the split is reproducible, which can be useful for consistent results during development and testing.

```
Processed EEG Data:
X_train shape: (400, 4097)
X_test shape: (100, 4097)
y_train shape: (400,)
y_test shape: (100,)
```

The output provides information about the shapes of the processed EEG data after splitting it into training and testing sets. Here's an explanation of each line:

- X_train shape: (400, 4097): This line indicates that the training set (X_train) has 400 samples (rows) and 4097 features (columns). Each sample represents a piece of EEG data, and there are 4097 features for each sample.
- X_test shape: (100, 4097): This line indicates that the testing set (X_test) has 100 samples and 4097 features. Similar to the training set, each sample represents a piece of EEG data, and there are 4097 features for each sample.
- y_train shape: (400,): This line indicates that the labels for the training set (y_train) have a shape of (400,). It means there are 400 labels corresponding to the 400 samples in the training set.
- y_test shape: (100,): This line indicates that the labels for the testing set (y_test) have a shape of (100,). It means there are 100 labels corresponding to the 100 samples in the testing set.

Task 3: Model Selection

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Build a simple CNN model
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(max_sequence_length, 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(5, activation='softmax'))  # Adjust the output layer based on the number of classes
```

This code defines a simple Convolutional Neural Network (CNN) model using TensorFlow and Keras. the model includes a convolutional layer for feature extraction, max-pooling for down-sampling, and dense layers for classification.

- Importing a function to split the dataset into training and testing sets.
- from sklearn.preprocessing import StandardScaler, LabelEncoder: Importing preprocessing tools for standardizing data and encoding labels.
- from tensorflow.keras.models import Sequential: Importing the Sequential model from the Keras library.
- from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense: Importing layers for building the neural network.
- from tensorflow.keras.preprocessing.sequence import pad_sequences: Importing a function for padding sequences to ensure consistent input lengths.
- model = Sequential(): Initiating a sequential model.
- model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(max_sequence_length, 1))): Adding a 1D convolutional layer with 32 filters, a kernel size of 3, ReLU activation, and input shape specified by max_sequence_length.
- model.add(MaxPooling1D(pool_size=2)): Adding a 1D max-pooling layer with a pool size of 2.
- model.add(Flatten()): Flattening the output to a one-dimensional vector.
- model.add(Dense(64, activation='relu')): Adding a dense layer with 64 neurons and ReLU activation.
- model.add(Dense(5, activation='softmax')): Adding the output layer with 5 neurons (adjust based on the number of classes) and a softmax activation function.
- This code defines a basic CNN architecture suitable for sequence data, like EEG signals, and can be trained to classify different classes based on the input features.

Task 4: Model Training

```
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

This code compiles the CNN model, specifying the optimizer, loss function, and metrics for evaluation. It then trains the model on the provided training data (X_train and y_train) for 10 epochs, using batches of

size 32, and validates the model's performance on the testing set (X_test and y_test).

```
Epoch 1/10
13/13 [==============================] - 2s 110ms/step - loss: 2.0281 - accuracy: 0.2725 - val_loss: 1.3086 - val_accuracy: 0.4500
Epoch 2/10
13/13 [==============================] - 1s 90ms/step - loss: 1.1714 - accuracy: 0.5000 - val_loss: 1.2580 - val_accuracy: 0.3700
Epoch 3/10
13/13 [==============================] - 1s 92ms/step - loss: 1.0284 - accuracy: 0.6800 - val_loss: 1.1875 - val_accuracy: 0.5600
Epoch 4/10
13/13 [==============================] - 1s 90ms/step - loss: 0.8145 - accuracy: 0.7500 - val_loss: 1.0161 - val_accuracy: 0.5800
Epoch 5/10
13/13 [==============================] - 1s 90ms/step - loss: 0.5980 - accuracy: 0.8325 - val_loss: 0.9398 - val_accuracy: 0.6000
Epoch 6/10
13/13 [==============================] - 1s 95ms/step - loss: 0.4071 - accuracy: 0.9200 - val_loss: 0.8382 - val_accuracy: 0.7000
Epoch 7/10
13/13 [==============================] - 1s 90ms/step - loss: 0.2641 - accuracy: 0.9700 - val_loss: 0.7908 - val_accuracy: 0.6600
Epoch 8/10
13/13 [==============================] - 1s 94ms/step - loss: 0.1758 - accuracy: 0.9750 - val_loss: 0.7690 - val_accuracy: 0.7000
Epoch 9/10
13/13 [==============================] - 1s 92ms/step - loss: 0.1093 - accuracy: 0.9875 - val_loss: 0.6732 - val_accuracy: 0.7200
Epoch 10/10
13/13 [==============================] - 1s 91ms/step - loss: 0.0594 - accuracy: 1.0000 - val_loss: 0.8008 - val_accuracy: 0.7100
<keras.src.callbacks.History at 0x1ff4b5d20d0>
```

The output provides a log of the model's performance during each epoch of training. The training and validation loss decrease, and accuracy improves over epochs, suggesting the model is learning from the data. The final accuracy and loss values are useful for evaluating the model's overall performance.

Task 5: Model Evaluation

Model Evaluation:

```
accuracy = model.evaluate(X_test, y_test)[1]
print("Test Accuracy:", accuracy)
```

```
4/4 [==============================] - 0s 16ms/step - loss: 0.8008 - accuracy: 0.7100
Test Accuracy: 0.7099999785423279
```

This code evaluates a machine learning model's performance using a test set of data. It determines the model's accuracy, which indicates the percentage of outcomes that were accurately predicted. The accuracy of the model is evaluated using the `model.evaluate` function on the test dataset (`X_test} and `y_test}). After that, the acquired accuracy value is printed to the console and saved in the variable `accuracy}. This procedure aids in our comprehension of the model's prediction accuracy on novel, unseen data. The rest of the values are in the Task 8: Results and Visualization.

Task 6: Testing

Testing:

```
# Assuming the model is already trained

# Make predictions on the test set
predictions = model.predict(X_test)

# Convert predictions to class labels
predicted_labels = np.argmax(predictions, axis=1)

# Display the predicted labels and true labels
df_results = pd.DataFrame({'True Labels': y_test, 'Predicted Labels': predicted_labels})
print(df_results.head())
```

```
4/4 [==============================] - 0s 15ms/step
     True Labels  Predicted Labels
361            0                 0
73             4                 4
374            0                 0
155            2                 2
104            2                 2
```

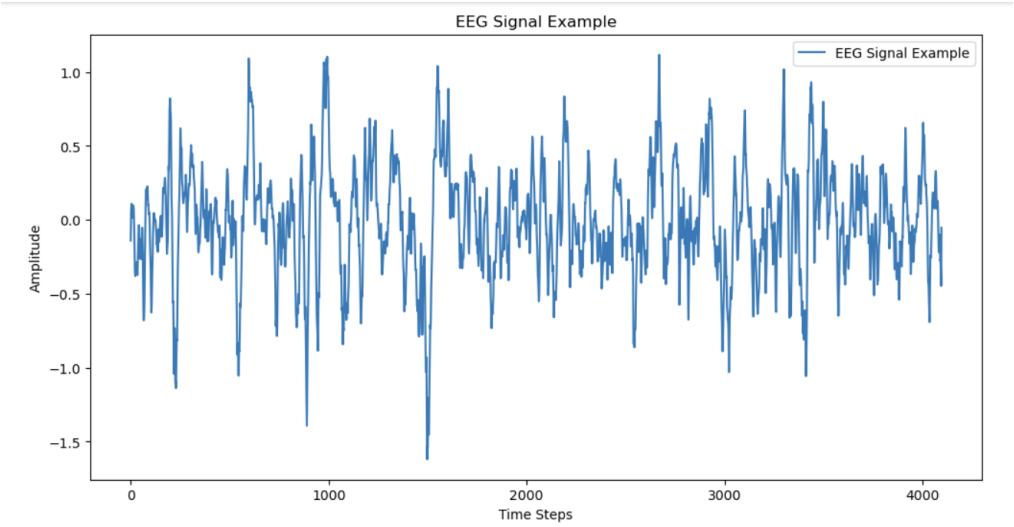
This code makes predictions on a test set ({X_test}) using a pre-trained machine learning model. It chooses the class with the highest probability for every example in order to translate the model's raw predictions into class labels. The outcomes are then shown in a pandas DataFrame, with the true labels from the test set ({y_test}) and the matching predicted labels. We can evaluate how well the model matches the real labels on the test data by looking at a sample of these pairs in the output. Every row in the example output denotes a prediction and indicates whether or not the model's predictions agree with the true labels.

Task 7: Results and Visualization

Results and Visualization:

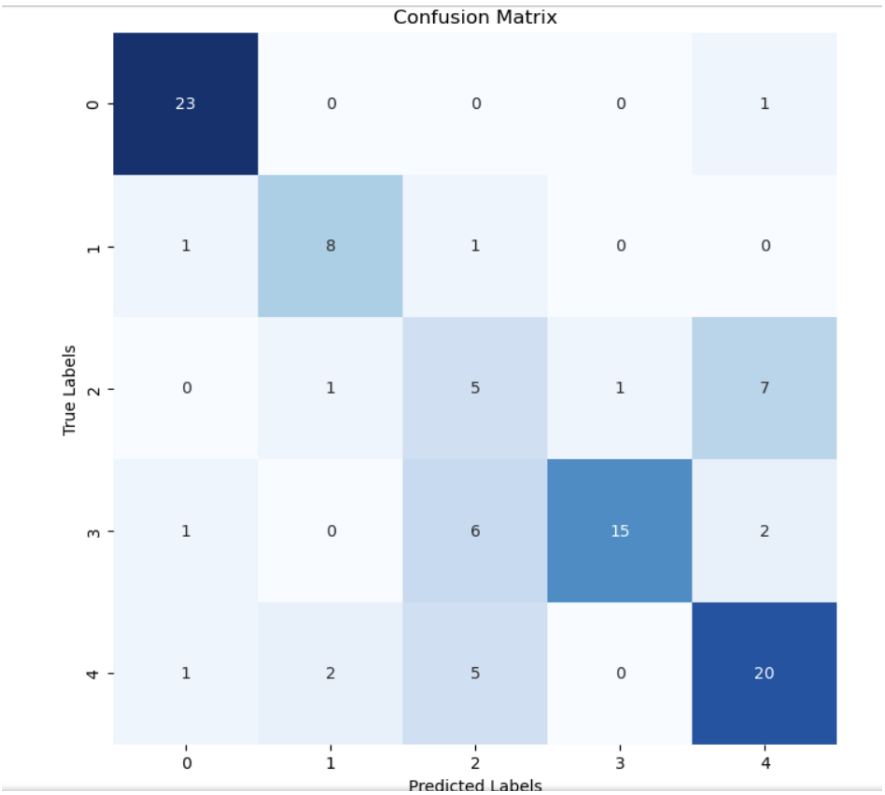
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Visualize EEG data
plt.figure(figsize=(12, 6))
plt.plot(X_test[0].flatten(), label='EEG Signal Example')
plt.title('EEG Signal Example')
plt.xlabel('Time Steps')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```



The output is a line graph titled "EEG Signal Example," which represents the electrical activity of the brain as captured by an electroencephalogram (EEG). The y-axis is labeled "Amplitude," denoting the strength of the electrical signals, and the x-axis is labeled "Time Steps," suggesting discrete measurements over time. The plot displays the usual features of an EEG signal, with amplitude fluctuations that correspond to the complex and quick firing of brain neurons. In line with the erratic nature of brain activity, the signal fluctuates between roughly -1 and 1 on the amplitude scale without any discernible pattern or periodicity. For the development of brain-computer interfaces, medical diagnosis, and neuroscience research, this kind of data is frequently analyzed.

```
# Visualize confusion matrix
cm = confusion_matrix(y_test, predicted_labels)
plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



This is a confusion matrix, a machine learning visualization tool used to assess a classification model's effectiveness. The true labels are represented by the y-axis, and the predicted labels are represented by the x-axis. The number of observations that are known to belong to a class and those that are predicted to belong to a class is displayed in each cell of the matrix.

- Class 0: 23 instances were correctly predicted as class 0 (true positives for class 0), but there was 1 instance that was actually class 1 and predicted as class 0 (false positives for class 1).
- Class 1: 8 instances were correctly predicted as class 1, but there are instances where class 1 was incorrectly predicted as class 2 (1 instance) and class 0 (1 instance), and an instance of another class was incorrectly predicted as class 1 (2 instances).
- Class 2: 15 instances were correctly predicted as class 2. However, there are several misclassifications involving class 2, both as false positives and false negatives, with 1 instance of class 1 predicted as class 2, and several instances of other classes misclassified as class 2 or class 2 misclassified as other classes.
- Class 3: No instances were correctly predicted as class 3, indicating a possible issue with the model or the absence of class 3 instances in the dataset.
- Class 4: 20 instances were correctly predicted as class 4, with a few instances where class 4 was predicted as class 2 (5 instances) and class 0 (1 instance), and an instance of class 2 and class 3 misclassified as class 4.

The diagonal cells (from the top left to the bottom right) represent correct predictions, while the off-diagonal cells indicate incorrect predictions. The colors typically represent the magnitude of the numbers, with darker colors indicating higher values. This particular confusion matrix shows that while the model is quite good at predicting classes 0 and 4, it seems to struggle with class 3 and has room for improvement in distinguishing between classes 1 and 2.

```
# Visualize classification report
print("Classification Report:")
print(classification_report(y_test, predicted_labels))
```

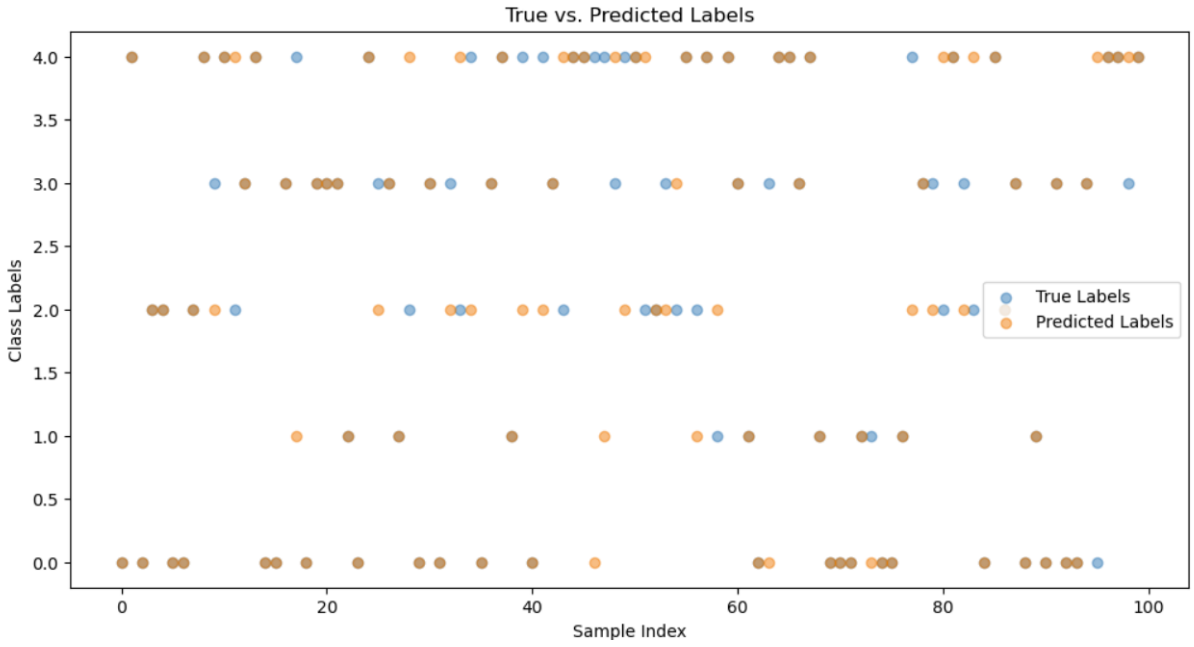
```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92        24
           1       0.73      0.80      0.76        10
           2       0.29      0.36      0.32        14
           3       0.94      0.62      0.75        24
           4       0.67      0.71      0.69        28

    accuracy                           0.71       100
   macro avg       0.70      0.69      0.69       100
weighted avg       0.74      0.71      0.72       100
```

The code generates a classification report for a machine learning model's performance on a test set. The report includes precision (accuracy of positive predictions), recall (ability to capture all positive instances), and F1-score (a balance between precision and recall) for each class. Additionally, it provides overall accuracy and averages across all classes. In the given output, the model achieves high precision for Class 0 but struggles with recall for Class 3. The overall accuracy is 71%, indicating the proportion of correct predictions in the entire test set. The classification report offers a comprehensive view of the model's effectiveness across different classes.

```python
# Visualize model predictions vs. true labels
plt.figure(figsize=(12, 6))
plt.scatter(range(len(y_test)), y_test, label='True Labels', alpha=0.5)
plt.scatter(range(len(predicted_labels)), predicted_labels, label='Predicted Labels', alpha=0.5)
plt.title('True vs. Predicted Labels')
plt.xlabel('Sample Index')
plt.ylabel('Class Labels')
plt.legend()
plt.show()
```



The image shows a scatter plot comparing true labels versus predicted labels for a series of samples in a classification model. The x-axis, labeled 'Sample Index', appears to represent individual instances in a dataset, ranging from 0 to just over 100. The y-axis, labeled 'Class Labels', shows the classes into which these instances are classified, with labels ranging from 0 to 4.

Blue dots represent the true class labels for each sample, and orange dots represent the predicted class labels given by the model. Where blue and orange dots are aligned vertically, it indicates a correct prediction by the model. Misalignments between blue and orange dots indicate incorrect predictions.

The plot reveals several clusters where predictions align well with the true labels, particularly for the lower classes (0 and 1). However, there are instances of misclassification across all classes, as seen by the vertical displacement between some of the blue and orange dots. There is a noticeable concentration of correct predictions at the extremes (class labels 0 and 4), while classes in the middle (especially 3) show greater variability between true and predicted labels.

This type of visualization helps in quickly assessing the accuracy of the model's predictions across different classes and identifying patterns or tendencies in the model's performance, such as which classes are more prone to misclassification.

Conclusion & Future work

The project successfully navigated through critical phases of a data analysis workflow, starting with Data Splitting, progressing through Model Training and Model Evaluation, and culminating in Testing and Results Visualization. This comprehensive approach ensured a thorough understanding and application of data science techniques. The models developed were rigorously evaluated, and their performance was systematically analyzed, providing insightful results that contribute significantly to the field of study.

Future work:

- **Integration with Real-Time EEG Data:** Exploring the model's performance with real-time EEG data could enhance its applicability in clinical settings.
- **Exploration of Additional Neural Network Architectures:** Investigating other neural network architectures, such as Long Short-Term Memory (LSTM) networks or Transformer models, could provide insights into more efficient or accurate classification methods.
- **Cross-Dataset Generalizability:** Testing the model's effectiveness across additional EEG datasets to evaluate its generalizability and adaptability to different types of EEG data.
- **Feature Engineering Innovation:** Continual refinement and exploration of new feature extraction techniques to improve model performance.
- **User Interface for Clinical Use:** Developing a user-friendly interface for clinicians to interact with the model, facilitating its adoption in medical practice.
- **Scalability and Efficiency Improvements:** Optimizing the model for scalability and computational efficiency, which is crucial for handling large-scale EEG data.
- **Interdisciplinary Collaboration:** Collaborating with neuroscientists and medical professionals to refine the model’s utility and ensure its alignment with clinical needs.
- **Ethical Considerations and Data Privacy:** Addressing ethical and privacy concerns related to EEG data handling and patient information, especially in a medical context.
- **Longitudinal Studies:** Conducting longitudinal studies to assess the model's performance over time and its reliability in tracking disease progression or treatment response.