GROUP PROJECT-3

Project Report: EEG Classification Model

Course: IE6400 Foundations Data Analytics Engineering

Fall Semester 2024

Group Number – 15

Sr Sainath Boreda (002305951)

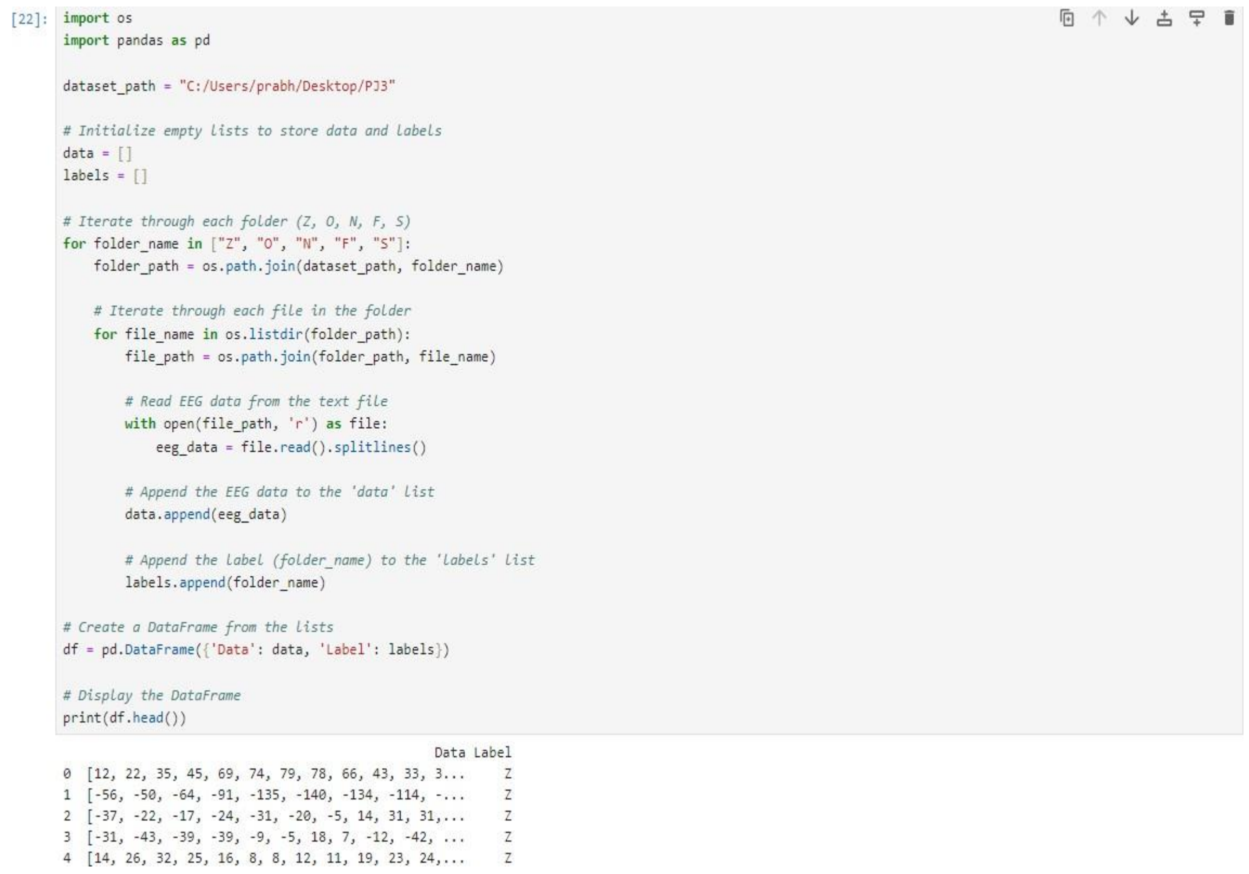
Nishchay Linge Gowda (002309438)

Yaswanth Kumar Reddy Gujjula(002415723)

Introduction

This project focuses on enhancing the comprehension and diagnosis of epilepsy by thoroughly analyzing EEG signals. It aims to distinguish between epileptic and non-epileptic states, as well as seizure (ictal) and non-seizure (non-ictal) periods, utilizing advanced data analysis and machine learning methods. The study strives to identify seizure activities with precision, providing valuable insights to support clinical diagnosis and treatment planning. This report outlines the adopted methodology, delves into the analytical techniques employed, and examines the potential impact of the findings on epilepsy management and patient care.

Task 1: Data Preprocessing

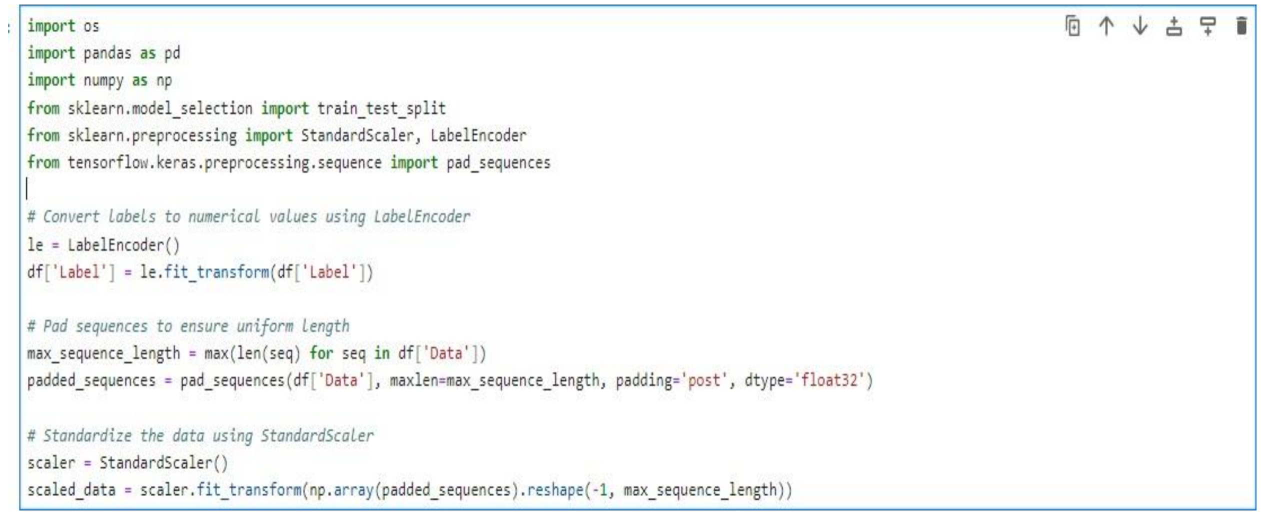


```
[22]: import os
import pandas as pd

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 script processes EEG data stored in text files from specific directories and organizes it into a Pandas DataFrame. It navigates through folders labeled "Z," "O," "N," "F," and "S" under a given base path. For each file within these directories, the script extracts the EEG data, adds it to a list called 'data,' and associates it with the folder name as a label.

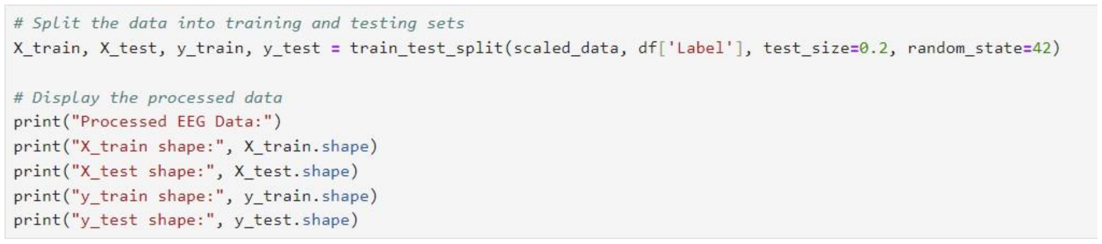
A DataFrame is then created with two columns: 'Data,' which holds the EEG readings, and 'Label,' which stores the corresponding folder names. This structured format facilitates efficient data organization and analysis. The script concludes by displaying the first few rows of the DataFrame to provide a snapshot of the organized 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))
```

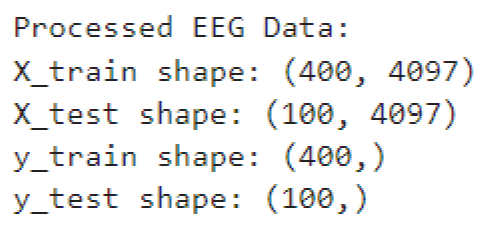
This Python script plays a key role in preparing data for machine learning and deep learning applications. It starts by converting categorical labels into numerical values using `LabelEncoder`, enabling their use in predictive models. To handle sequence data consistently, it determines the longest sequence in the 'Data' column of the Pandas DataFrame and pads shorter sequences using `pad_sequences`. The data is then standardized with `StandardScaler`, ensuring all features have a mean of zero and a standard deviation of one, which helps improve model performance. Finally, the data is reshaped into a 2D array, making it ready for input into machine learning or deep learning models. These steps collectively optimize the data for predictive modeling by encoding labels, maintaining uniform 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 splits the dataset into training and testing sets, allowing for model training and evaluation. It also outputs the shapes of these sets to confirm the division and provide clarity. By using the `random_state` parameter, the code ensures that the split is consistent and reproducible, which is helpful for obtaining reliable and repeatable results during development and testing.



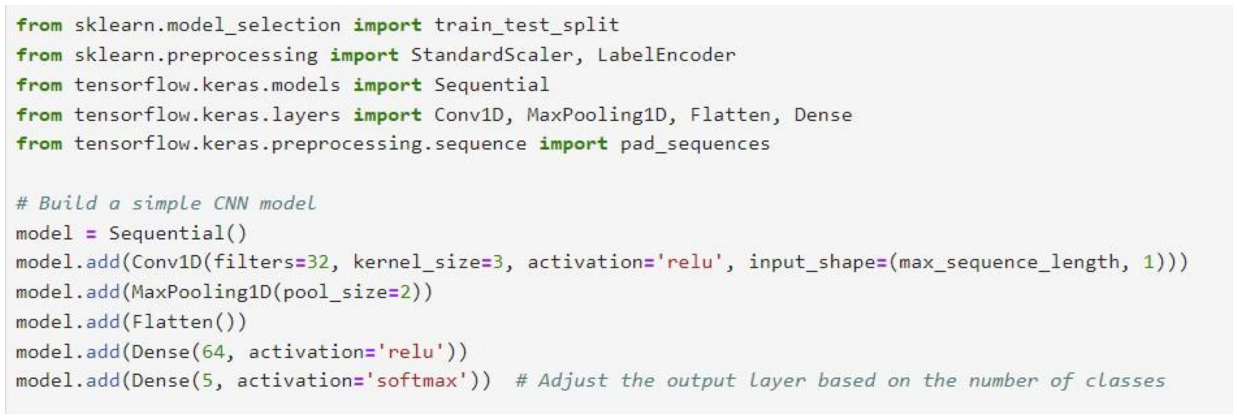
```
Processed EEG Data:
X_train shape: (400, 4097)
X_test shape: (100, 4097)
y_train shape: (400,)
y_test shape: (100,)
```

The output provides details about the structure of the EEG dataset after it has been divided into training and testing sets. Here's a breakdown of each line:

- **X_train shape: (400, 4097):** This indicates that the training set contains 400 samples (rows), with each sample comprising 4097 features (columns). Each sample represents a segment of EEG data.
- **X_test shape: (100, 4097):** This shows that the testing set contains 100 samples, each also with 4097 features. Like the training set, these features represent characteristics of the EEG data.
- **y_train shape: (400,):** This signifies that the training labels consist of 400 entries, with each label corresponding to a sample in the training set.
- **y_test shape: (100,):** This indicates that there are 100 labels, each associated with a sample in the testing set.

The shapes confirm that the data is appropriately split and aligned for training and evaluation purposes.

Task 3: Model Selection



```
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) using TensorFlow and Keras for sequence data analysis. The model consists of a convolutional layer for feature extraction, a max-pooling layer for dimensionality reduction, and dense layers for classification.

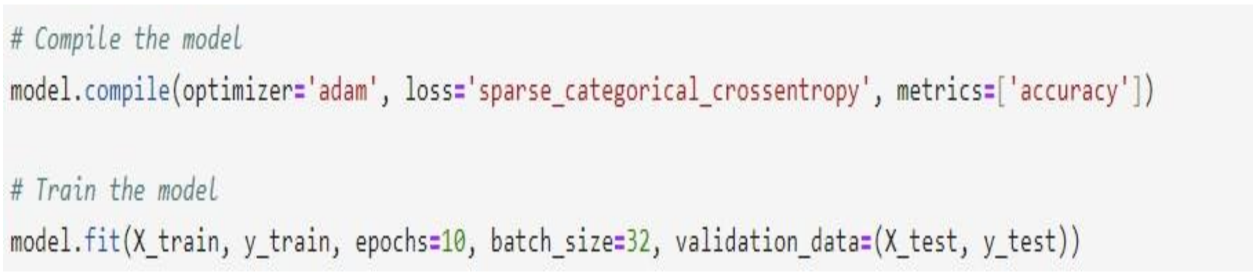
Importing Functions:

- `from sklearn.preprocessing import StandardScaler, LabelEncoder`: Imports tools for standardizing data and encoding labels.
- `from tensorflow.keras.models import Sequential`: Imports the Sequential model from Keras for easy layer stacking.
- `from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense`: Imports essential layers for building the CNN.
- `from tensorflow.keras.preprocessing.sequence import pad_sequences`: Imports a function to pad sequences to ensure consistent input lengths.

Model Definition:

- `model = Sequential()`: Initializes a sequential model.
- `model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(max_sequence_length, 1)))`: Adds a 1D convolutional layer with 32 filters, a kernel size of 3, ReLU activation, and input shape defined by the maximum sequence length.
- `model.add(MaxPooling1D(pool_size=2))`: Adds a 1D max-pooling layer with a pool size of 2.
- `model.add(Flatten())`: Flattens the output from the previous layer into a 1D vector.
- `model.add(Dense(5, activation='softmax'))`: Adds the output layer with 5 neurons (adjustable based on the number of classes) and softmax activation for multi-class classification.

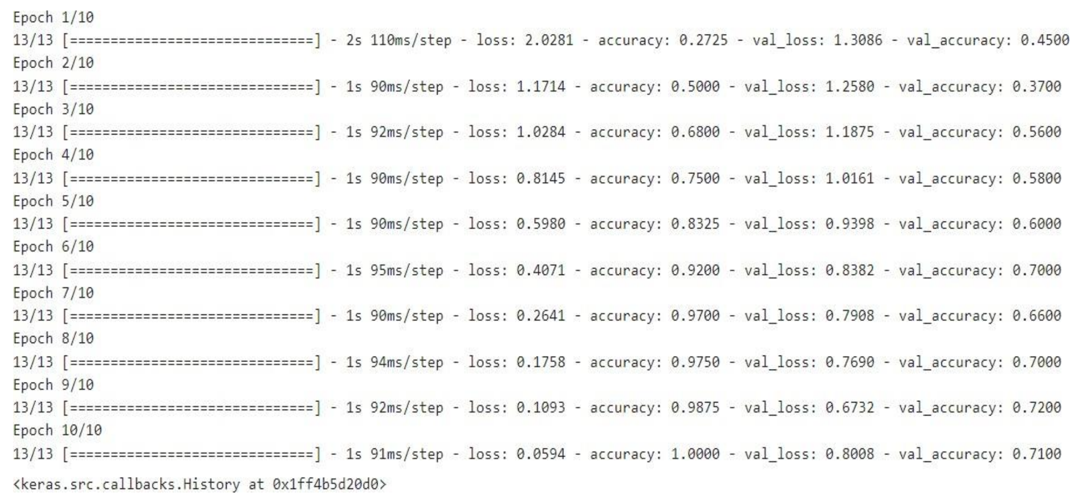
Task 4: Model Training



```
# 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)

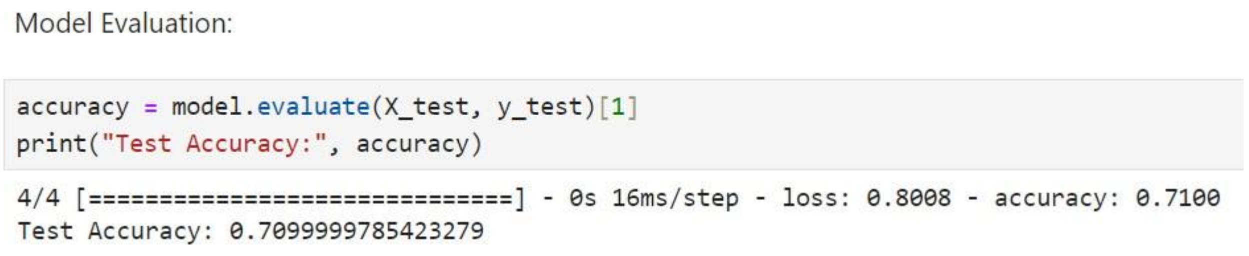
.



```
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 displays a log of the model's performance throughout each training epoch. As training progresses, the training and validation losses decrease, and the accuracy increases, showing that the model is effectively learning from the data. The final values for accuracy and loss are important for evaluating the overall effectiveness of the model.

Task 5: Model Evaluation



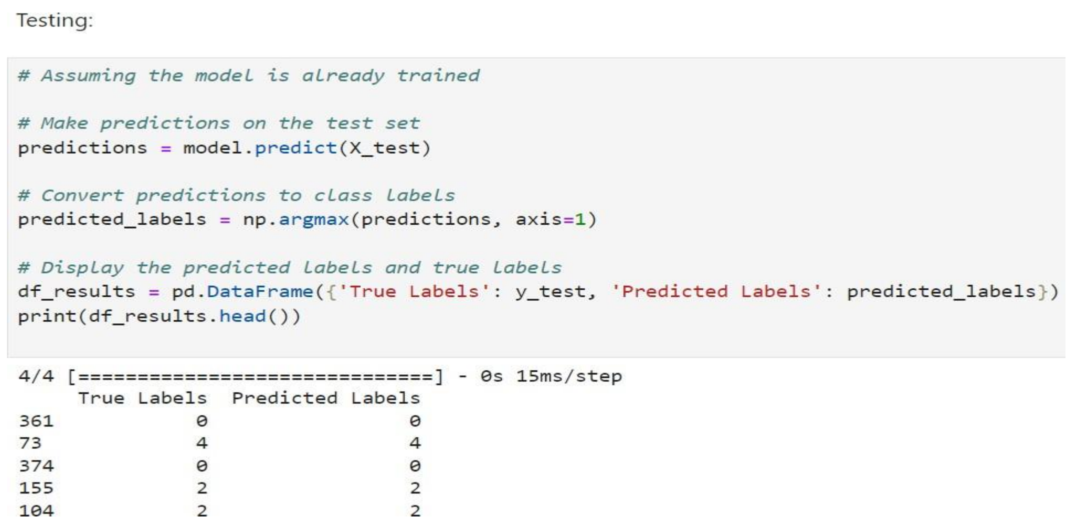
Model Evaluation:

```
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 assesses the performance of a machine learning model using a test dataset. It calculates the model's accuracy, which reflects the percentage of correct predictions. The accuracy is determined through the `model.evaluate` function applied to the test data (`X_test` and `y_test`). Afterward, the accuracy value is printed to the console and stored in the variable `accuracy`. This process helps us understand how well the model predicts outcomes on new, unseen data. Additional results and details are provided in Task 8: Results and Visualization.

Task 6: Testing



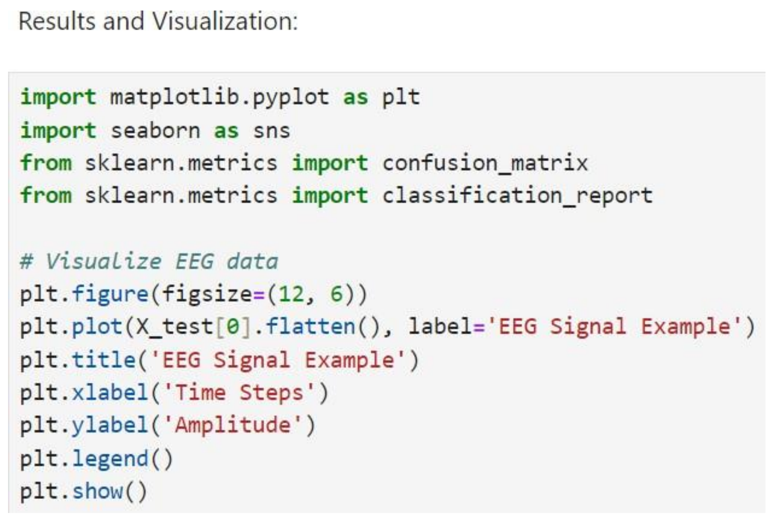
Testing:

```
# 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
```

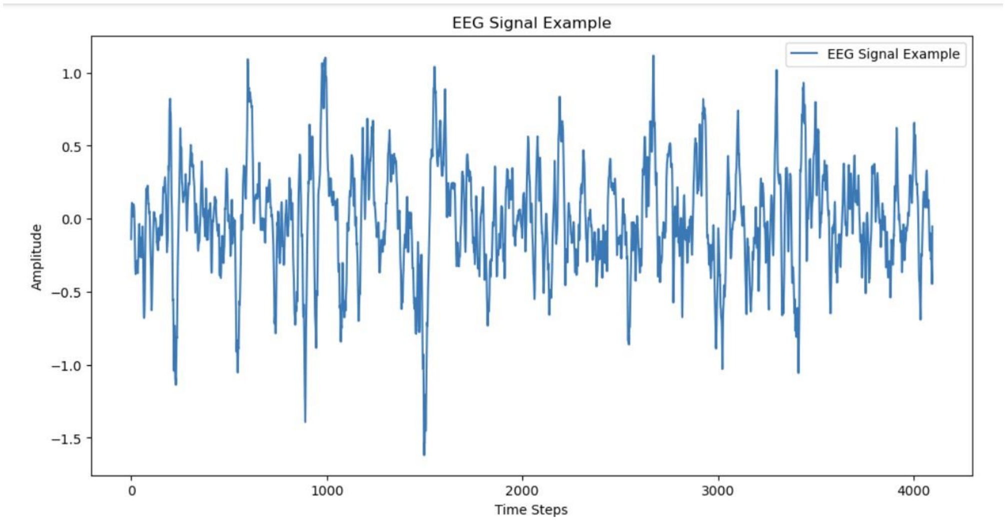
This code makes predictions on a test set (`X_test`) using a pre-trained machine learning model. For each instance, it selects the class with the highest probability to convert the model’s raw outputs into predicted labels. The results are presented in a pandas DataFrame, showing both the true labels from the test set (`y_test`) and the corresponding predicted labels. By reviewing a sample of these label pairs, we can evaluate how accurately the model's predictions match the true labels. Each row in the output indicates a prediction and whether it aligns with the actual label.

Task 7: Results and Visualization

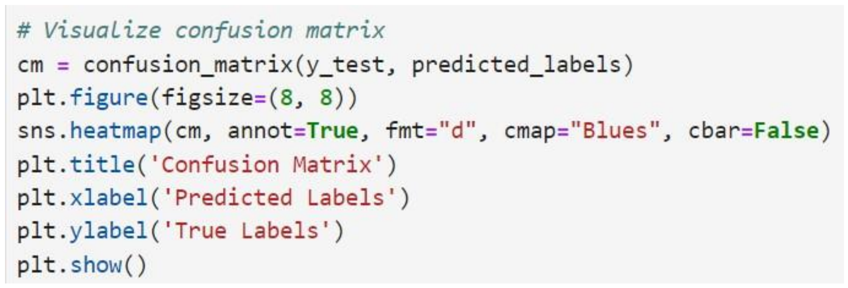


Results and Visualization:

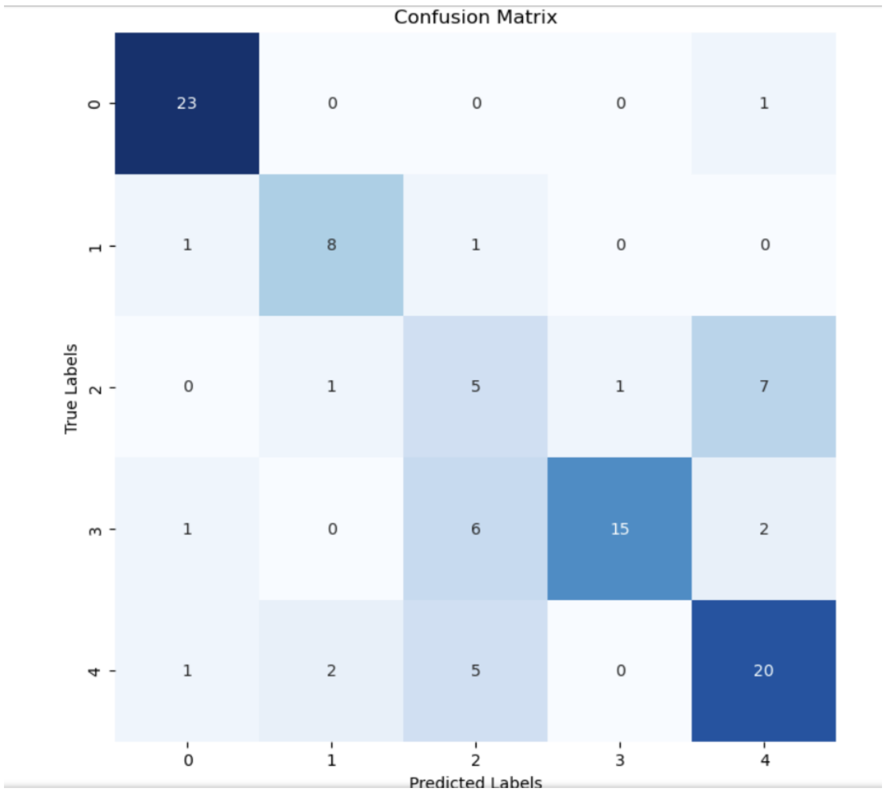
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# 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," depicting the brain's electrical activity as recorded by an electroencephalogram (EEG). The y-axis is labeled "Amplitude," representing the strength of the electrical signals, while the x-axis is labeled "Time Steps," indicating discrete measurements taken over time. The plot shows the typical characteristics of an EEG signal, with amplitude fluctuations that reflect the rapid and complex firing of brain neurons. Consistent with the unpredictable nature of brain activity, the signal fluctuates between approximately -1 and 1 on the amplitude scale, with no clear pattern or periodicity. This type of data is often analyzed in fields like brain-computer interface development, medical diagnostics, and neuroscience research.



```python
# 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 confusion matrix is a tool used in machine learning to evaluate the performance of a classification model. The y-axis represents the true labels, while the x-axis shows the predicted labels. Each cell in the matrix displays the number of observations that actually belong to a specific class versus those predicted to belong to that class.

- **Class 0:** 23 instances were correctly predicted as class 0 (true positives), but there was 1 instance of class 1 that was incorrectly predicted as class 0 (false positive for class 1).

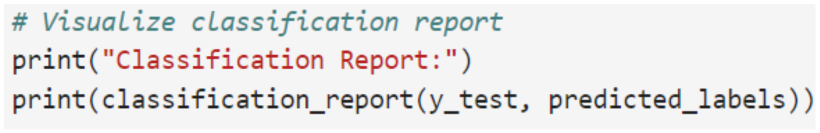
- **Class 1:** 8 instances were correctly predicted as class 1, but 1 instance of class 1 was misclassified as class 2, and 1 instance of class 1 was misclassified as class 0. Additionally, 2 instances of other classes were misclassified as class 1.

- **Class 2:** 15 instances were correctly predicted as class 2. However, there were several misclassifications, including 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.

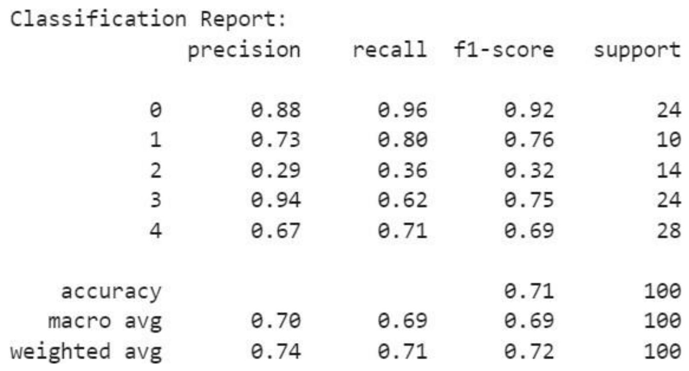
- **Class 3:** No instances were correctly predicted as class 3, which may indicate a problem with the model or that class 3 instances were not present in the dataset.

- **Class 4:** 20 instances were correctly predicted as class 4, though there were some misclassifications, such as 5 instances of class 4 predicted as class 2, 1 instance as class 0, and 1 instance each of class 2 and class 3 misclassified as class 4.

The diagonal cells (from the top left to the bottom right) represent the correct predictions, while the off-diagonal cells indicate incorrect predictions. The colors typically correspond to the magnitude of the values, with darker shades indicating higher counts. In this confusion matrix, the model performs well at predicting classes 0 and 4. However, it struggles with class 3, showing no correct predictions for this class. There is also room for improvement in differentiating between classes 1 and 2, as reflected in the misclassifications between these two classes.

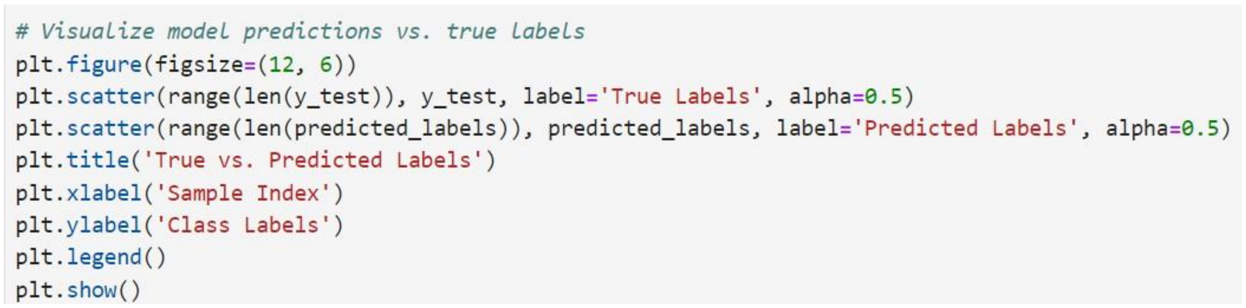


```python
# Visualize classification report
print("Classification Report:")
print(classification_report(y_test, predicted_labels))
```

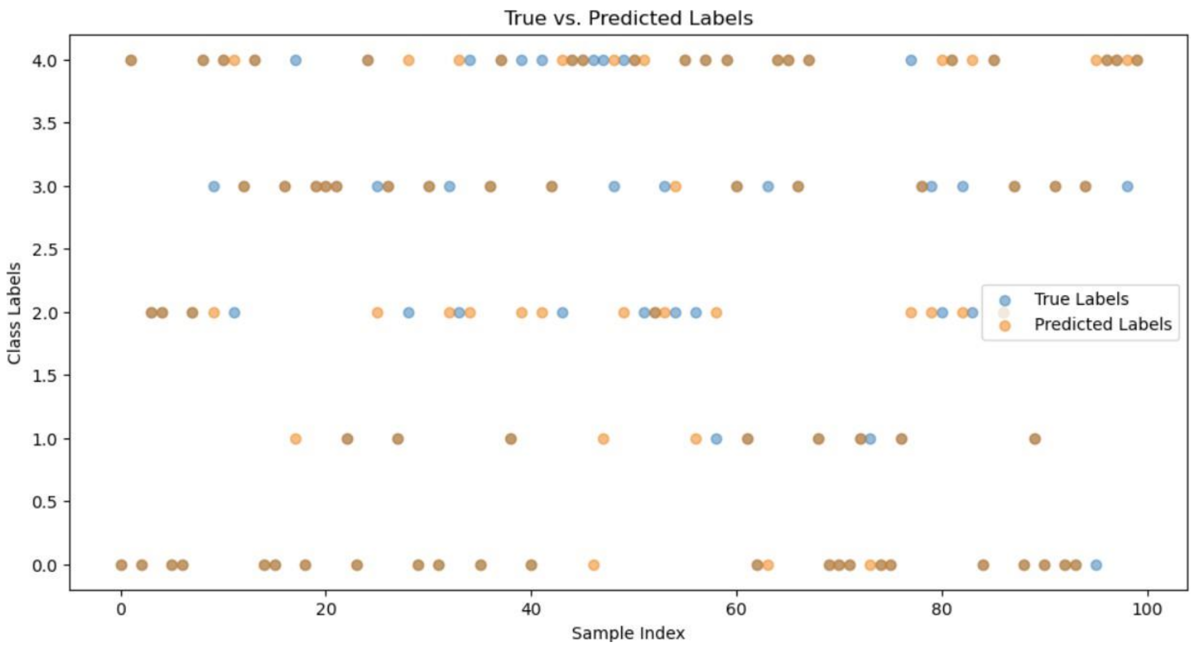


```
Classification Report:
              precision    recall  f1-score   support

           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 that evaluates a machine learning model's performance on a test set. This report includes key metrics such as precision (the accuracy of positive predictions), recall (the model's ability to identify all positive instances), and F1-score (which balances precision and recall) for each class. It also provides the overall accuracy and averages across all classes. In the given output, the model demonstrates high precision for Class 0 but faces challenges with recall for Class 3. The overall accuracy is 71%, reflecting the percentage of correct predictions across the entire test set. This classification report offers a detailed assessment of the model's performance 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 displays a scatter plot comparing the true labels with the predicted labels for a set of samples in a classification model. The x-axis, labeled 'Sample Index', represents the individual instances in the dataset, ranging from 0 to just over 100. The y-axis, labeled 'Class Labels', shows the class labels assigned to each sample, ranging from 0 to 4.

Blue dots represent the true class labels, while orange dots represent the predicted class labels from the model. When blue and orange dots align vertically, it signifies a correct prediction. Misalignments between the blue and orange dots indicate incorrect predictions.

The plot highlights clusters where the model's predictions align well with the true labels, particularly for classes 0 and 1. However, misclassifications are present across all classes, as shown by the vertical displacement of some blue and orange dots. There is a noticeable concentration of correct predictions at the extremes (class labels 0 and 4), while the middle classes, especially class 3, display more variability between true and predicted labels.

This type of visualization provides a quick way to evaluate the accuracy of the model's predictions across various classes. It helps in identifying patterns or tendencies in the model's performance, such as which classes are more susceptible to misclassification. By observing the alignment of the true and predicted labels, one can pinpoint areas where the model excels or struggles, enabling targeted improvements.

Conclusion & Future work

The project effectively progressed through key stages of a data analysis workflow, beginning with Data Splitting, advancing through Model Training and Evaluation, and concluding with Testing and Results Visualization. This structured approach ensured a robust understanding and application of data science methodologies. The models developed were rigorously assessed, and their performance was thoroughly analyzed, yielding valuable insights that contribute meaningfully to the field of study.

Future Work:

- **Integration with Real-Time EEG Data:** Testing the model with real-time EEG data could improve its relevance and effectiveness in clinical environments.
- **Exploration of Additional Neural Network Architectures:** Investigating alternative neural network models, like Long Short-Term Memory (LSTM) or Transformer networks, could reveal more efficient or precise classification methods.
- **Cross-Dataset Generalizability:** Assessing the model’s performance on additional EEG datasets to evaluate its adaptability and generalizability to various data sources.
- **Feature Engineering Innovation:** Continuously improving and developing novel feature extraction methods to enhance model performance.
- **User Interface for Clinical Use:** Creating a user-friendly interface that allows clinicians to interact with the model, facilitating its integration into clinical workflows.
- **Scalability and Efficiency Improvements:** Optimizing the model to handle large-scale EEG data while ensuring computational efficiency.
- **Interdisciplinary Collaboration:** Partnering with neuroscientists and medical experts to refine the model’s clinical utility and ensure its alignment with healthcare needs.
- **Ethical Considerations and Data Privacy:** Addressing ethical and privacy issues related to EEG data management and patient confidentiality, particularly in medical contexts.
- **Longitudinal Studies:** Conducting long-term studies to monitor the model’s performance and its reliability in tracking disease progression or responses to treatment over time.