model_v4.0

April 16, 2024

[ ]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,␣
↪Dropout, BatchNormalization
from tensorflow.keras.callbacks import ReduceLROnPlateau
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
```

[ ]:
```python
csv_path = 'D:\\Semester 7\\FYP\\preprocessing\\output_labels.csv'
df = pd.read_csv(csv_path)

# Extract file paths and class labels
file_paths = df['Path'].values
class_labels = df['Class'].values

# Load ECG data from file paths
ecg_data = []
for path in file_paths:
    # Load ECG data from CSV file
    ecg_df = pd.read_csv(path)
    # Assuming your ECG data is in columns I, II, III, AVR, AVL, AVF, V1, V2,␣
↪V3, V4, V5, V6
    ecg_values = ecg_df[['I', 'II', 'III', 'AVR', 'AVL',
                         'AVF', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6']].values
    ecg_data.append(ecg_values)

X = np.array(ecg_data)
y = np.array(class_labels)
```

[ ]:
```python
print(X.shape)
```

```
(5000, 5000, 12)
```

[ ]:
```python
print(y.shape)
```

```
(5000,)
```

[ ]:
```
print(y)
```

```
['NORM' 'NORM' 'NORM' … 'HYP' 'HYP' 'HYP']
```

[ ]:
```
# Assuming you have already loaded and preprocessed your data into X and y

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```
print(y_train)
```

```
['CD' 'HYP' 'CD' … 'CD' 'STTC' 'NORM']
```

[ ]:
```
print(y_test)
```

```
['STTC' 'CD' 'STTC' 'STTC' 'NORM' 'NORM' 'NORM' 'MI' 'CD' 'NORM' 'MI'
 'NORM' 'HYP' 'NORM' 'HYP' 'STTC' 'HYP' 'MI' 'CD' 'NORM' 'STTC' 'HYP'
 'NORM' 'STTC' 'HYP' 'MI' 'STTC' 'NORM' 'HYP' 'MI' 'STTC' 'NORM' 'HYP'
 'HYP' 'NORM' 'STTC' 'STTC' 'CD' 'HYP' 'STTC' 'STTC' 'HYP' 'NORM' 'NORM'
 'STTC' 'CD' 'STTC' 'HYP' 'HYP' 'HYP' 'HYP' 'HYP' 'HYP' 'STTC' 'MI' 'MI'
 'NORM' 'NORM' 'NORM' 'MI' 'MI' 'HYP' 'CD' 'HYP' 'STTC' 'HYP' 'HYP' 'MI'
 'CD' 'HYP' 'MI' 'NORM' 'NORM' 'STTC' 'NORM' 'MI' 'NORM' 'MI' 'CD' 'CD'
 'MI' 'MI' 'CD' 'MI' 'MI' 'NORM' 'NORM' 'NORM' 'HYP' 'HYP' 'STTC' 'NORM'
 'NORM' 'NORM' 'STTC' 'HYP' 'CD' 'STTC' 'NORM' 'NORM' 'CD' 'NORM' 'STTC'
 'HYP' 'NORM' 'MI' 'CD' 'NORM' 'CD' 'NORM' 'CD' 'MI' 'STTC' 'STTC' 'MI'
 'STTC' 'HYP' 'MI' 'NORM' 'CD' 'MI' 'CD' 'HYP' 'NORM' 'CD' 'HYP' 'HYP'
 'HYP' 'STTC' 'STTC' 'HYP' 'CD' 'HYP' 'CD' 'STTC' 'STTC' 'CD' 'STTC'
 'NORM' 'NORM' 'HYP' 'HYP' 'HYP' 'HYP' 'CD' 'MI' 'HYP' 'HYP' 'STTC' 'MI'
 'MI' 'STTC' 'HYP' 'HYP' 'HYP' 'NORM' 'STTC' 'HYP' 'CD' 'MI' 'STTC' 'CD'
 'HYP' 'NORM' 'NORM' 'STTC' 'NORM' 'NORM' 'STTC' 'MI' 'STTC' 'CD' 'MI'
 'MI' 'STTC' 'MI' 'STTC' 'CD' 'NORM' 'STTC' 'HYP' 'MI' 'MI' 'MI' 'STTC'
 'CD' 'HYP' 'STTC' 'MI' 'MI' 'NORM' 'STTC' 'CD' 'HYP' 'CD' 'HYP' 'NORM'
 'CD' 'CD' 'HYP' 'CD' 'NORM' 'CD' 'CD' 'HYP' 'MI' 'HYP' 'STTC' 'CD' 'NORM'
 'CD' 'MI' 'HYP' 'MI' 'CD' 'STTC' 'HYP' 'MI' 'HYP' 'MI' 'NORM' 'MI' 'HYP'
 'STTC' 'CD' 'HYP' 'STTC' 'MI' 'STTC' 'MI' 'CD' 'STTC' 'NORM' 'HYP' 'NORM'
 'STTC' 'CD' 'CD' 'STTC' 'HYP' 'HYP' 'NORM' 'HYP' 'HYP' 'STTC' 'NORM'
 'HYP' 'CD' 'STTC' 'HYP' 'HYP' 'CD' 'MI' 'STTC' 'MI' 'NORM' 'CD' 'MI'
 'NORM' 'HYP' 'STTC' 'MI' 'MI' 'HYP' 'NORM' 'HYP' 'MI' 'STTC' 'STTC' 'CD'
 'STTC' 'CD' 'HYP' 'NORM' 'MI' 'CD' 'NORM' 'HYP' 'STTC' 'NORM' 'HYP' 'HYP'
 'MI' 'NORM' 'NORM' 'CD' 'NORM' 'HYP' 'CD' 'MI' 'NORM' 'MI' 'STTC' 'CD'
 'MI' 'NORM' 'MI' 'CD' 'HYP' 'NORM' 'HYP' 'STTC' 'STTC' 'NORM' 'CD' 'HYP'
 'HYP' 'MI' 'HYP' 'MI' 'CD' 'STTC' 'MI' 'HYP' 'NORM' 'STTC' 'CD' 'STTC'
 'HYP' 'STTC' 'HYP' 'HYP' 'MI' 'HYP' 'CD' 'NORM' 'STTC' 'STTC' 'STTC' 'CD'
 'STTC' 'NORM' 'CD' 'MI' 'STTC' 'STTC' 'NORM' 'CD' 'CD' 'CD' 'CD' 'STTC'
 'NORM' 'MI' 'STTC' 'STTC' 'HYP' 'CD' 'CD' 'MI' 'CD' 'STTC' 'STTC' 'HYP'
 'HYP' 'STTC' 'MI' 'NORM' 'MI' 'STTC' 'NORM' 'HYP' 'CD' 'MI' 'NORM' 'MI'
```

```
'NORM' 'NORM' 'HYP' 'CD' 'CD' 'HYP' 'NORM' 'HYP' 'MI' 'STTC' 'STTC' 'MI'
'CD' 'CD' 'CD' 'HYP' 'STTC' 'MI' 'NORM' 'STTC' 'HYP' 'NORM' 'NORM' 'MI'
'NORM' 'HYP' 'MI' 'NORM' 'NORM' 'STTC' 'NORM' 'NORM' 'STTC' 'MI' 'NORM'
'CD' 'STTC' 'STTC' 'STTC' 'MI' 'STTC' 'HYP' 'NORM' 'STTC' 'NORM' 'MI'
'STTC' 'NORM' 'CD' 'HYP' 'NORM' 'HYP' 'MI' 'STTC' 'CD' 'STTC' 'STTC'
'NORM' 'MI' 'HYP' 'STTC' 'MI' 'HYP' 'NORM' 'NORM' 'HYP' 'MI' 'HYP' 'CD'
'MI' 'HYP' 'STTC' 'NORM' 'MI' 'STTC' 'NORM' 'STTC' 'MI' 'NORM' 'NORM'
'CD' 'HYP' 'NORM' 'NORM' 'MI' 'HYP' 'HYP' 'MI' 'HYP' 'STTC' 'CD' 'MI'
'STTC' 'MI' 'NORM' 'NORM' 'CD' 'MI' 'MI' 'CD' 'CD' 'HYP' 'HYP' 'HYP' 'MI'
'MI' 'STTC' 'MI' 'CD' 'STTC' 'MI' 'MI' 'MI' 'NORM' 'CD' 'CD' 'MI' 'CD'
'CD' 'HYP' 'CD' 'CD' 'NORM' 'MI' 'NORM' 'HYP' 'MI' 'MI' 'NORM' 'STTC'
'MI' 'CD' 'NORM' 'STTC' 'STTC' 'CD' 'CD' 'CD' 'NORM' 'NORM' 'STTC' 'NORM'
'NORM' 'HYP' 'STTC' 'HYP' 'CD' 'NORM' 'NORM' 'CD' 'STTC' 'NORM' 'NORM'
'CD' 'CD' 'CD' 'NORM' 'CD' 'MI' 'CD' 'CD' 'MI' 'CD' 'HYP' 'MI' 'STTC'
'STTC' 'NORM' 'HYP' 'HYP' 'CD' 'STTC' 'MI' 'HYP' 'HYP' 'HYP' 'NORM' 'HYP'
'HYP' 'MI' 'STTC' 'HYP' 'STTC' 'MI' 'STTC' 'CD' 'CD' 'CD' 'STTC' 'NORM'
'CD' 'HYP' 'HYP' 'MI' 'CD' 'STTC' 'MI' 'MI' 'NORM' 'HYP' 'CD' 'MI' 'CD'
'CD' 'HYP' 'CD' 'HYP' 'MI' 'NORM' 'HYP' 'NORM' 'HYP' 'STTC' 'STTC' 'CD'
'HYP' 'HYP' 'CD' 'MI' 'CD' 'HYP' 'STTC' 'MI' 'HYP' 'CD' 'NORM' 'CD' 'HYP'
'HYP' 'MI' 'STTC' 'STTC' 'MI' 'STTC' 'NORM' 'HYP' 'NORM' 'HYP' 'HYP'
'STTC' 'NORM' 'MI' 'CD' 'HYP' 'NORM' 'HYP' 'NORM' 'STTC' 'CD' 'MI' 'NORM'
'MI' 'NORM' 'MI' 'STTC' 'STTC' 'MI' 'MI' 'CD' 'HYP' 'HYP' 'MI' 'MI'
'NORM' 'NORM' 'STTC' 'NORM' 'MI' 'MI' 'NORM' 'MI' 'CD' 'HYP' 'HYP' 'STTC'
'STTC' 'HYP' 'STTC' 'HYP' 'STTC' 'MI' 'CD' 'NORM' 'STTC' 'STTC' 'MI'
'HYP' 'MI' 'CD' 'CD' 'CD' 'NORM' 'STTC' 'NORM' 'NORM' 'NORM' 'CD' 'MI'
'MI' 'HYP' 'NORM' 'HYP' 'STTC' 'NORM' 'HYP' 'NORM' 'NORM' 'NORM' 'CD'
'CD' 'STTC' 'CD' 'STTC' 'MI' 'STTC' 'CD' 'CD' 'HYP' 'STTC' 'HYP' 'NORM'
'STTC' 'HYP' 'HYP' 'CD' 'HYP' 'HYP' 'MI' 'HYP' 'STTC' 'NORM' 'NORM' 'CD'
'STTC' 'HYP' 'CD' 'MI' 'HYP' 'HYP' 'HYP' 'MI' 'STTC' 'CD' 'CD' 'STTC'
'NORM' 'STTC' 'CD' 'MI' 'STTC' 'CD' 'MI' 'NORM' 'HYP' 'STTC' 'NORM' 'CD'
'STTC' 'MI' 'HYP' 'NORM' 'MI' 'CD' 'MI' 'HYP' 'HYP' 'STTC' 'CD' 'HYP'
'HYP' 'NORM' 'STTC' 'CD' 'NORM' 'MI' 'MI' 'STTC' 'CD' 'HYP' 'STTC' 'NORM'
'NORM' 'STTC' 'HYP' 'MI' 'MI' 'HYP' 'HYP' 'CD' 'MI' 'STTC' 'MI' 'CD' 'CD'
'CD' 'NORM' 'MI' 'MI' 'CD' 'NORM' 'MI' 'STTC' 'CD' 'MI' 'CD' 'MI' 'NORM'
'STTC' 'STTC' 'CD' 'NORM' 'CD' 'NORM' 'CD' 'MI' 'MI' 'CD' 'MI' 'CD' 'CD'
'STTC' 'CD' 'MI' 'NORM' 'STTC' 'HYP' 'HYP' 'NORM' 'HYP' 'MI' 'NORM' 'CD'
'HYP' 'NORM' 'MI' 'NORM' 'MI' 'NORM' 'HYP' 'STTC' 'MI' 'HYP' 'NORM' 'HYP'
'HYP' 'NORM' 'NORM' 'STTC' 'CD' 'CD' 'HYP' 'STTC' 'HYP' 'STTC' 'NORM'
'HYP' 'CD' 'HYP' 'CD' 'HYP' 'HYP' 'STTC' 'CD' 'STTC' 'CD' 'STTC' 'NORM'
'HYP' 'CD' 'MI' 'HYP' 'NORM' 'MI' 'STTC' 'HYP' 'NORM' 'NORM' 'MI' 'NORM'
'HYP' 'STTC' 'HYP' 'HYP' 'HYP' 'CD' 'NORM' 'MI' 'CD' 'HYP' 'NORM' 'NORM'
'NORM' 'CD' 'MI' 'STTC' 'CD' 'CD' 'HYP' 'CD' 'STTC' 'CD' 'CD' 'STTC'
'HYP' 'CD' 'NORM' 'NORM' 'HYP' 'STTC' 'MI' 'CD' 'MI' 'CD' 'NORM' 'STTC'
'NORM' 'CD' 'STTC' 'NORM' 'NORM' 'HYP' 'HYP' 'STTC' 'CD' 'STTC' 'HYP'
'CD' 'MI' 'HYP' 'MI' 'MI' 'CD' 'MI' 'HYP' 'NORM' 'MI' 'CD' 'CD' 'HYP'
'NORM' 'STTC' 'MI' 'MI' 'HYP' 'STTC' 'HYP' 'STTC' 'MI' 'CD' 'STTC' 'HYP'
'HYP' 'CD' 'NORM' 'HYP' 'STTC' 'CD' 'STTC' 'CD' 'HYP' 'STTC' 'HYP' 'NORM'
'MI' 'NORM' 'HYP' 'NORM' 'NORM' 'CD' 'STTC' 'CD' 'MI' 'NORM' 'HYP' 'CD'
```

```
 'STTC' 'HYP' 'NORM' 'HYP' 'MI' 'NORM' 'HYP' 'STTC' 'MI' 'HYP' 'HYP' 'MI'
 'CD' 'NORM' 'NORM' 'NORM' 'CD' 'HYP' 'HYP' 'MI' 'MI' 'CD' 'CD' 'STTC'
 'HYP' 'NORM' 'NORM' 'STTC' 'CD' 'STTC' 'NORM' 'NORM' 'HYP' 'MI' 'MI'
 'STTC' 'STTC' 'STTC' 'HYP' 'NORM' 'NORM' 'HYP' 'MI' 'MI' 'STTC' 'HYP'
 'NORM' 'HYP' 'HYP' 'MI' 'CD' 'CD' 'MI' 'CD' 'HYP' 'MI' 'HYP' 'CD' 'CD']
```

[ ]:
```python
# # Initialize LabelEncoder
# label_encoder = LabelEncoder()

# # Fit and transform labels for training data
# y_train_encoded = label_encoder.fit_transform(y_train)

# # Transform labels for test data (using the same encoder from training data)
# y_test_encoded = label_encoder.transform(y_test)
```

[ ]:
```python
from sklearn.preprocessing import LabelEncoder

# Initialize the label encoder
le = LabelEncoder()

# Fit the label encoder and transform the labels
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)
```

[ ]:
```python
print(le.classes_)
```

```
['CD' 'HYP' 'MI' 'NORM' 'STTC']
```

[ ]:
```python
print(y_train)
```

```
[0 1 0 … 0 4 3]
```

[ ]:
```python
print(y_test)
```

```
[4 0 4 4 3 3 3 2 0 3 2 3 1 3 1 4 1 2 0 3 4 1 3 4 1 2 4 3 1 2 4 3 1 1 3 4 4
 0 1 4 4 1 3 3 4 0 4 1 1 1 1 1 1 4 2 2 3 3 3 2 2 1 0 1 4 1 1 2 0 1 2 3 3 4
 3 2 3 2 0 0 2 2 0 2 2 3 3 3 1 1 4 3 3 3 4 1 0 4 3 3 0 3 4 1 3 2 0 3 0 3 0
 2 4 4 2 4 1 2 3 0 2 0 1 3 0 1 1 1 4 4 1 0 1 0 4 4 0 4 3 3 1 1 1 1 0 2 1 1
 4 2 2 4 1 1 1 3 4 1 0 2 4 0 1 3 3 4 3 3 4 2 4 0 2 2 4 2 4 0 3 4 1 2 2 2 4
 0 1 4 2 2 3 4 0 1 0 1 3 0 0 1 0 3 0 0 1 2 1 4 0 3 0 2 1 2 0 4 1 2 1 2 3 2
 1 4 0 1 4 2 4 2 0 4 3 1 3 4 0 0 4 1 1 3 1 1 4 3 1 0 4 1 1 0 2 4 2 3 0 2 3
 1 4 2 2 1 3 1 2 4 4 0 4 0 1 3 2 0 3 1 4 3 1 1 2 3 3 0 3 1 0 2 3 2 4 0 2 3
 2 0 1 3 1 4 4 3 0 1 1 2 1 2 0 4 2 1 3 4 0 4 1 4 1 1 2 1 0 3 4 4 4 0 4 3 0
 2 4 4 3 0 0 0 0 4 3 2 4 4 1 0 0 2 0 4 4 1 1 4 2 3 2 4 3 1 0 2 3 2 3 3 1 0
 0 1 3 1 2 4 4 2 0 0 0 1 4 2 3 4 1 3 3 2 3 1 2 3 3 4 3 3 4 2 3 0 4 4 4 2 4
 1 3 4 3 2 4 3 0 1 3 1 2 4 0 4 4 3 2 1 4 2 1 3 3 1 2 1 0 2 1 4 3 2 4 3 4 2
 3 3 0 1 3 3 2 1 1 2 1 4 0 2 4 2 3 3 0 2 2 0 0 1 1 1 2 2 4 2 0 4 2 2 2 3 0
 0 2 0 0 1 0 0 3 2 3 1 2 2 3 4 2 0 3 4 4 0 0 0 3 3 4 3 3 1 4 1 0 3 3 0 4 3
 3 0 0 0 3 0 2 0 0 2 0 1 2 4 4 3 1 1 0 4 2 1 1 1 3 1 1 2 4 1 4 2 4 0 0 0 4
```

```
3 0 1 1 2 0 4 2 2 3 1 0 2 0 0 1 0 1 2 3 1 3 1 4 4 0 1 1 0 2 0 1 4 2 1 0 3
0 1 1 2 4 4 2 4 3 1 3 1 1 4 3 2 0 1 3 1 3 4 0 2 3 2 3 2 4 4 2 2 0 1 1 2 2
3 3 4 3 2 2 3 2 0 1 1 4 4 1 4 1 4 2 0 3 4 4 2 1 2 0 0 0 3 4 3 3 3 0 2 2 1
3 1 4 3 1 3 3 3 0 0 4 0 4 2 4 0 0 1 4 1 3 4 1 1 0 1 1 2 1 4 3 3 0 4 1 0 2
1 1 1 2 4 0 0 4 3 4 0 2 4 0 2 3 1 4 3 0 4 2 1 3 2 0 2 1 1 4 0 1 1 3 4 0 3
2 2 4 0 1 4 3 3 4 1 2 2 1 1 0 2 4 2 0 0 0 3 2 2 0 3 2 4 0 2 0 2 3 4 4 0 3
0 3 0 2 2 0 2 0 0 4 0 2 3 4 1 1 3 1 2 3 0 1 3 2 3 2 3 1 4 2 1 3 1 1 3 3 4
0 0 1 4 1 4 3 1 0 1 0 1 1 4 0 4 0 4 3 1 0 2 1 3 2 4 1 3 3 2 3 1 4 1 1 1 0
3 2 0 1 3 3 3 0 2 4 0 0 1 0 4 0 0 4 1 0 3 3 1 4 2 0 2 0 3 4 3 0 4 3 3 1 1
4 0 4 1 0 2 1 2 2 0 2 1 3 2 0 0 1 3 4 2 2 1 4 1 4 2 0 4 1 1 0 3 1 4 0 4 0
1 4 1 3 2 3 1 3 3 0 4 0 2 3 1 0 4 1 3 1 2 3 1 4 2 1 1 2 0 3 3 3 0 1 1 2 2
0 0 4 1 3 3 4 0 4 3 3 1 2 2 4 4 4 1 3 3 1 2 2 4 1 3 1 1 2 0 0 2 0 1 2 1 0
0]
```

[ ]:
```
print(y_train)
```

```
[0 1 0 … 0 4 3]
```

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

# Assuming X_train and X_test are already defined and contain your data

# Reshape data back to 2 dimensions for MinMaxScaler
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform on flattened training data
X_train_scaled = scaler.fit_transform(X_train_flat)

# Transform flattened test data (using the same scaler from training data)
X_test_scaled = scaler.transform(X_test_flat)

# Reshape data for Conv1D model
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train.
 ↪shape[1], X_train.shape[2], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], X_test.
 ↪shape[1], X_test.shape[2], 1)
```

[ ]:
```
total_elements = X_train.size
print(total_elements)
```

```
240000000
```

[ ]:
```
total_elements_test = X_test.size
print(total_elements_test)
```

60000000

```
[ ]: num_features = X_train.shape[1]
     print(num_features)
```

5000

```
[ ]: num_samples = X_train.shape[0]
     print(num_samples)
```

4000

```
[ ]: print(X_test.shape)
```

(1000, 5000, 12)

```
[ ]: print(X_train.shape)
```

(4000, 5000, 12)

```
[ ]: # Reshape data for Conv1D model
     X_train_reshaped = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.
      ↪shape[2], 1)
     X_test_reshaped = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.
      ↪shape[2], 1)
```

```
[ ]: # Model architecture
     model = Sequential()
     model.add(Conv1D(filters=32, kernel_size=5, activation='relu',␣
      ↪input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
     model.add(BatchNormalization())
     model.add(MaxPooling1D(pool_size=2))
     model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
     model.add(BatchNormalization())
     model.add(MaxPooling1D(pool_size=2))
     model.add(Flatten())
     model.add(Dense(128, activation='relu'))
     model.add(Dropout(0.5))
     model.add(Dense(len(np.unique(y_train)), activation='softmax'))
```

```
[ ]: # Compile the model with a lower learning rate and Adam optimizer
     optimizer = Adam(learning_rate=0.0001)
     model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer,␣
      ↪metrics=['accuracy'])

     # Learning rate scheduler
     reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,␣
      ↪min_lr=0.00001)
```

```
# Train the model
history = model.fit(X_train_reshaped, y_train, epochs=20, batch_size=32,
                    validation_data=(X_test_reshaped, y_test),␣
 ↪callbacks=[reduce_lr])

# Model evaluation
accuracy = model.evaluate(X_test_reshaped, y_test)[1]
print(f"Test Accuracy: {accuracy}")

# Additional evaluation metrics
y_pred = model.predict(X_test_reshaped)
y_pred_classes = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_classes))
```

```
Epoch 1/20
125/125              38s 263ms/step -
accuracy: 0.2645 - loss: 3.2561 - val_accuracy: 0.2020 - val_loss: 7.2245 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125              33s 268ms/step -
accuracy: 0.5821 - loss: 1.4657 - val_accuracy: 0.2100 - val_loss: 12.9873 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125              33s 267ms/step -
accuracy: 0.7178 - loss: 0.8243 - val_accuracy: 0.2420 - val_loss: 11.0156 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125              31s 250ms/step -
accuracy: 0.7922 - loss: 0.5749 - val_accuracy: 0.3190 - val_loss: 5.4977 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125              33s 260ms/step -
accuracy: 0.8465 - loss: 0.4518 - val_accuracy: 0.3840 - val_loss: 3.2556 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125              32s 256ms/step -
accuracy: 0.8830 - loss: 0.3447 - val_accuracy: 0.4140 - val_loss: 2.9118 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125              33s 261ms/step -
accuracy: 0.8951 - loss: 0.3084 - val_accuracy: 0.4100 - val_loss: 2.8027 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125              33s 260ms/step -
accuracy: 0.9076 - loss: 0.2526 - val_accuracy: 0.4270 - val_loss: 2.8502 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125              32s 258ms/step -
```

```
accuracy: 0.9296 - loss: 0.2089 - val_accuracy: 0.4090 - val_loss: 2.9207 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125            32s 254ms/step -
accuracy: 0.9416 - loss: 0.1780 - val_accuracy: 0.3970 - val_loss: 2.9550 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125            32s 258ms/step -
accuracy: 0.9338 - loss: 0.1853 - val_accuracy: 0.4100 - val_loss: 2.9196 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125            32s 257ms/step -
accuracy: 0.9477 - loss: 0.1671 - val_accuracy: 0.3990 - val_loss: 3.1181 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125            33s 265ms/step -
accuracy: 0.9551 - loss: 0.1298 - val_accuracy: 0.3990 - val_loss: 3.1113 -
learning_rate: 1.0000e-05
Epoch 14/20
125/125            32s 254ms/step -
accuracy: 0.9561 - loss: 0.1373 - val_accuracy: 0.4030 - val_loss: 3.0626 -
learning_rate: 1.0000e-05
Epoch 15/20
125/125            32s 253ms/step -
accuracy: 0.9514 - loss: 0.1353 - val_accuracy: 0.4050 - val_loss: 3.0212 -
learning_rate: 1.0000e-05
Epoch 16/20
125/125            32s 257ms/step -
accuracy: 0.9611 - loss: 0.1147 - val_accuracy: 0.4000 - val_loss: 3.0252 -
learning_rate: 1.0000e-05
Epoch 17/20
125/125            32s 255ms/step -
accuracy: 0.9730 - loss: 0.0915 - val_accuracy: 0.3990 - val_loss: 3.0954 -
learning_rate: 1.0000e-05
Epoch 18/20
125/125            32s 258ms/step -
accuracy: 0.9730 - loss: 0.0961 - val_accuracy: 0.4050 - val_loss: 3.0617 -
learning_rate: 1.0000e-05
Epoch 19/20
125/125            32s 256ms/step -
accuracy: 0.9703 - loss: 0.1036 - val_accuracy: 0.4030 - val_loss: 3.0910 -
learning_rate: 1.0000e-05
Epoch 20/20
125/125            32s 254ms/step -
accuracy: 0.9742 - loss: 0.0899 - val_accuracy: 0.4000 - val_loss: 3.0354 -
learning_rate: 1.0000e-05
32/32            1s 41ms/step -
accuracy: 0.4060 - loss: 2.9876
```

```
Test Accuracy: 0.4000000059604645
32/32          2s 43ms/step
              precision    recall  f1-score   support

           0       0.40      0.36      0.38       196
           1       0.49      0.38      0.43       224
           2       0.24      0.22      0.23       186
           3       0.45      0.68      0.54       202
           4       0.38      0.35      0.37       192

    accuracy                           0.40      1000
   macro avg       0.39      0.40      0.39      1000
weighted avg       0.40      0.40      0.39      1000
```

[ ]:
```
print(model.input_shape)
```

```
(None, 5000, 1)
```

[ ]:
```
type(X_train_reshaped)
```

[ ]:
```
numpy.ndarray
```

[ ]:
```
type(X_test_reshaped)
```

[ ]:
```
numpy.ndarray
```

[ ]:
```
type(y_train)
```

[ ]:
```
numpy.ndarray
```

[ ]:
```
type(y_test)
```

[ ]:
```
numpy.ndarray
```

[ ]:
```
y_test
```

[ ]:
```
array(['STTC', 'CD', 'STTC', 'STTC', 'NORM', 'NORM', 'NORM', 'MI', 'CD',
       'NORM', 'MI', 'NORM', 'HYP', 'NORM', 'HYP', 'STTC', 'HYP', 'MI',
       'CD', 'NORM', 'STTC', 'HYP', 'NORM', 'STTC', 'HYP', 'MI', 'STTC',
       'NORM', 'HYP', 'MI', 'STTC', 'NORM', 'HYP', 'HYP', 'NORM', 'STTC',
       'STTC', 'CD', 'HYP', 'STTC', 'STTC', 'HYP', 'NORM', 'NORM', 'STTC',
       'CD', 'STTC', 'HYP', 'HYP', 'HYP', 'HYP', 'HYP', 'HYP', 'STTC',
       'MI', 'MI', 'NORM', 'NORM', 'NORM', 'MI', 'MI', 'HYP', 'CD', 'HYP',
       'STTC', 'HYP', 'HYP', 'MI', 'CD', 'HYP', 'MI', 'NORM', 'NORM',
       'STTC', 'NORM', 'MI', 'NORM', 'MI', 'CD', 'CD', 'MI', 'MI', 'CD',
       'MI', 'MI', 'NORM', 'NORM', 'NORM', 'HYP', 'HYP', 'STTC', 'NORM',
       'NORM', 'NORM', 'STTC', 'HYP', 'CD', 'STTC', 'NORM', 'NORM', 'CD',
       'NORM', 'STTC', 'HYP', 'NORM', 'MI', 'CD', 'NORM', 'CD', 'NORM',
```

'CD', 'MI', 'STTC', 'STTC', 'MI', 'STTC', 'HYP', 'MI', 'NORM',
'CD', 'MI', 'CD', 'HYP', 'NORM', 'CD', 'HYP', 'HYP', 'HYP', 'STTC',
'STTC', 'HYP', 'CD', 'HYP', 'CD', 'STTC', 'STTC', 'CD', 'STTC',
'NORM', 'NORM', 'HYP', 'HYP', 'HYP', 'HYP', 'CD', 'MI', 'HYP',
'HYP', 'STTC', 'MI', 'MI', 'STTC', 'HYP', 'HYP', 'HYP', 'NORM',
'STTC', 'HYP', 'CD', 'MI', 'STTC', 'CD', 'HYP', 'NORM', 'NORM',
'STTC', 'NORM', 'NORM', 'STTC', 'MI', 'STTC', 'CD', 'MI', 'MI',
'STTC', 'MI', 'STTC', 'CD', 'NORM', 'STTC', 'HYP', 'MI', 'MI',
'MI', 'STTC', 'CD', 'HYP', 'STTC', 'MI', 'MI', 'NORM', 'STTC',
'CD', 'HYP', 'CD', 'HYP', 'NORM', 'CD', 'CD', 'HYP', 'CD', 'NORM',
'CD', 'CD', 'HYP', 'MI', 'HYP', 'STTC', 'CD', 'NORM', 'CD', 'MI',
'HYP', 'MI', 'CD', 'STTC', 'HYP', 'MI', 'HYP', 'MI', 'NORM', 'MI',
'HYP', 'STTC', 'CD', 'HYP', 'STTC', 'MI', 'STTC', 'MI', 'CD',
'STTC', 'NORM', 'HYP', 'NORM', 'STTC', 'CD', 'CD', 'STTC', 'HYP',
'HYP', 'NORM', 'HYP', 'HYP', 'STTC', 'NORM', 'HYP', 'CD', 'STTC',
'HYP', 'HYP', 'CD', 'MI', 'STTC', 'MI', 'NORM', 'CD', 'MI', 'NORM',
'HYP', 'STTC', 'MI', 'MI', 'HYP', 'NORM', 'HYP', 'MI', 'STTC',
'STTC', 'CD', 'STTC', 'CD', 'HYP', 'NORM', 'MI', 'CD', 'NORM',
'HYP', 'STTC', 'NORM', 'HYP', 'HYP', 'MI', 'NORM', 'NORM', 'CD',
'NORM', 'HYP', 'CD', 'MI', 'NORM', 'MI', 'STTC', 'CD', 'MI',
'NORM', 'MI', 'CD', 'HYP', 'NORM', 'HYP', 'STTC', 'STTC', 'NORM',
'CD', 'HYP', 'HYP', 'MI', 'HYP', 'MI', 'CD', 'STTC', 'MI', 'HYP',
'NORM', 'STTC', 'CD', 'STTC', 'HYP', 'STTC', 'HYP', 'HYP', 'MI',
'HYP', 'CD', 'NORM', 'STTC', 'STTC', 'STTC', 'CD', 'STTC', 'NORM',
'CD', 'MI', 'STTC', 'STTC', 'NORM', 'CD', 'CD', 'CD', 'CD', 'STTC',
'NORM', 'MI', 'STTC', 'STTC', 'HYP', 'CD', 'CD', 'MI', 'CD',
'STTC', 'STTC', 'HYP', 'HYP', 'STTC', 'MI', 'NORM', 'MI', 'STTC',
'NORM', 'HYP', 'CD', 'MI', 'NORM', 'MI', 'NORM', 'NORM', 'HYP',
'CD', 'CD', 'HYP', 'NORM', 'HYP', 'MI', 'STTC', 'STTC', 'MI', 'CD',
'CD', 'CD', 'HYP', 'STTC', 'MI', 'NORM', 'STTC', 'HYP', 'NORM',
'NORM', 'MI', 'NORM', 'HYP', 'MI', 'NORM', 'NORM', 'STTC', 'NORM',
'NORM', 'STTC', 'MI', 'NORM', 'CD', 'STTC', 'STTC', 'STTC', 'MI',
'STTC', 'HYP', 'NORM', 'STTC', 'NORM', 'MI', 'STTC', 'NORM', 'CD',
'HYP', 'NORM', 'HYP', 'MI', 'STTC', 'CD', 'STTC', 'STTC', 'NORM',
'MI', 'HYP', 'STTC', 'MI', 'HYP', 'NORM', 'NORM', 'HYP', 'MI',
'HYP', 'CD', 'MI', 'HYP', 'STTC', 'NORM', 'MI', 'STTC', 'NORM',
'STTC', 'MI', 'NORM', 'NORM', 'CD', 'HYP', 'NORM', 'NORM', 'MI',
'HYP', 'HYP', 'MI', 'HYP', 'STTC', 'CD', 'MI', 'STTC', 'MI',
'NORM', 'NORM', 'CD', 'MI', 'MI', 'CD', 'CD', 'HYP', 'HYP', 'HYP',
'MI', 'MI', 'STTC', 'MI', 'CD', 'STTC', 'MI', 'MI', 'MI', 'NORM',
'CD', 'CD', 'MI', 'CD', 'CD', 'HYP', 'CD', 'CD', 'NORM', 'MI',
'NORM', 'HYP', 'MI', 'MI', 'NORM', 'STTC', 'MI', 'CD', 'NORM',
'STTC', 'STTC', 'CD', 'CD', 'CD', 'NORM', 'NORM', 'STTC', 'NORM',
'NORM', 'HYP', 'STTC', 'HYP', 'CD', 'NORM', 'NORM', 'CD', 'STTC',
'NORM', 'NORM', 'CD', 'CD', 'CD', 'NORM', 'CD', 'MI', 'CD', 'CD',
'MI', 'CD', 'HYP', 'MI', 'STTC', 'STTC', 'NORM', 'HYP', 'HYP',
'CD', 'STTC', 'MI', 'HYP', 'HYP', 'HYP', 'NORM', 'HYP', 'HYP',

'MI', 'STTC', 'HYP', 'STTC', 'MI', 'STTC', 'CD', 'CD', 'CD',
'STTC', 'NORM', 'CD', 'HYP', 'HYP', 'MI', 'CD', 'STTC', 'MI', 'MI',
'NORM', 'HYP', 'CD', 'MI', 'CD', 'CD', 'HYP', 'CD', 'HYP', 'MI',
'NORM', 'HYP', 'NORM', 'HYP', 'STTC', 'STTC', 'CD', 'HYP', 'HYP',
'CD', 'MI', 'CD', 'HYP', 'STTC', 'MI', 'HYP', 'CD', 'NORM', 'CD',
'HYP', 'HYP', 'MI', 'STTC', 'STTC', 'MI', 'STTC', 'NORM', 'HYP',
'NORM', 'HYP', 'HYP', 'STTC', 'NORM', 'MI', 'CD', 'HYP', 'NORM',
'HYP', 'NORM', 'STTC', 'CD', 'MI', 'NORM', 'MI', 'NORM', 'MI',
'STTC', 'STTC', 'MI', 'MI', 'CD', 'HYP', 'HYP', 'MI', 'MI', 'NORM',
'NORM', 'STTC', 'NORM', 'MI', 'MI', 'NORM', 'MI', 'CD', 'HYP',
'HYP', 'STTC', 'STTC', 'HYP', 'STTC', 'HYP', 'STTC', 'MI', 'CD',
'NORM', 'STTC', 'STTC', 'MI', 'HYP', 'MI', 'CD', 'CD', 'CD',
'NORM', 'STTC', 'NORM', 'NORM', 'NORM', 'CD', 'MI', 'MI', 'HYP',
'NORM', 'HYP', 'STTC', 'NORM', 'HYP', 'NORM', 'NORM', 'NORM', 'CD',
'CD', 'STTC', 'CD', 'STTC', 'MI', 'STTC', 'CD', 'CD', 'HYP',
'STTC', 'HYP', 'NORM', 'STTC', 'HYP', 'HYP', 'CD', 'HYP', 'HYP',
'MI', 'HYP', 'STTC', 'NORM', 'NORM', 'CD', 'STTC', 'HYP', 'CD',
'MI', 'HYP', 'HYP', 'HYP', 'MI', 'STTC', 'CD', 'CD', 'STTC',
'NORM', 'STTC', 'CD', 'MI', 'STTC', 'CD', 'MI', 'NORM', 'HYP',
'STTC', 'NORM', 'CD', 'STTC', 'MI', 'HYP', 'NORM', 'MI', 'CD',
'MI', 'HYP', 'HYP', 'STTC', 'CD', 'HYP', 'HYP', 'NORM', 'STTC',
'CD', 'NORM', 'MI', 'MI', 'STTC', 'CD', 'HYP', 'STTC', 'NORM',
'NORM', 'STTC', 'HYP', 'MI', 'MI', 'HYP', 'HYP', 'CD', 'MI',
'STTC', 'MI', 'CD', 'CD', 'CD', 'NORM', 'MI', 'MI', 'CD', 'NORM',
'MI', 'STTC', 'CD', 'MI', 'CD', 'MI', 'NORM', 'STTC', 'STTC', 'CD',
'NORM', 'CD', 'NORM', 'CD', 'MI', 'MI', 'CD', 'MI', 'CD', 'CD',
'STTC', 'CD', 'MI', 'NORM', 'STTC', 'HYP', 'HYP', 'NORM', 'HYP',
'MI', 'NORM', 'CD', 'HYP', 'NORM', 'MI', 'NORM', 'MI', 'NORM',
'HYP', 'STTC', 'MI', 'HYP', 'NORM', 'HYP', 'HYP', 'NORM', 'NORM',
'STTC', 'CD', 'CD', 'HYP', 'STTC', 'HYP', 'STTC', 'NORM', 'HYP',
'CD', 'HYP', 'CD', 'HYP', 'HYP', 'STTC', 'CD', 'STTC', 'CD',
'STTC', 'NORM', 'HYP', 'CD', 'MI', 'HYP', 'NORM', 'MI', 'STTC',
'HYP', 'NORM', 'NORM', 'MI', 'NORM', 'HYP', 'STTC', 'HYP', 'HYP',
'HYP', 'CD', 'NORM', 'MI', 'CD', 'HYP', 'NORM', 'NORM', 'NORM',
'CD', 'MI', 'STTC', 'CD', 'CD', 'HYP', 'CD', 'STTC', 'CD', 'CD',
'STTC', 'HYP', 'CD', 'NORM', 'NORM', 'HYP', 'STTC', 'MI', 'CD',
'MI', 'CD', 'NORM', 'STTC', 'NORM', 'CD', 'STTC', 'NORM', 'NORM',
'HYP', 'HYP', 'STTC', 'CD', 'STTC', 'HYP', 'CD', 'MI', 'HYP', 'MI',
'MI', 'CD', 'MI', 'HYP', 'NORM', 'MI', 'CD', 'CD', 'HYP', 'NORM',
'STTC', 'MI', 'MI', 'HYP', 'STTC', 'HYP', 'STTC', 'MI', 'CD',
'STTC', 'HYP', 'HYP', 'CD', 'NORM', 'HYP', 'STTC', 'CD', 'STTC',
'CD', 'HYP', 'STTC', 'HYP', 'NORM', 'MI', 'NORM', 'HYP', 'NORM',
'NORM', 'CD', 'STTC', 'CD', 'MI', 'NORM', 'HYP', 'CD', 'STTC',
'HYP', 'NORM', 'HYP', 'MI', 'NORM', 'HYP', 'STTC', 'MI', 'HYP',
'HYP', 'MI', 'CD', 'NORM', 'NORM', 'NORM', 'CD', 'HYP', 'HYP',
'MI', 'MI', 'CD', 'CD', 'STTC', 'HYP', 'NORM', 'NORM', 'STTC',
'CD', 'STTC', 'NORM', 'NORM', 'HYP', 'MI', 'MI', 'STTC', 'STTC',

```
'STTC', 'HYP', 'NORM', 'NORM', 'HYP', 'MI', 'MI', 'STTC', 'HYP',
'NORM', 'HYP', 'HYP', 'MI', 'CD', 'CD', 'MI', 'CD', 'HYP', 'MI',
'HYP', 'CD', 'CD'], dtype=object)
```

[ ]:
```
print(X_train_reshaped.dtype)
print(y_train.dtype)
print(X_test_reshaped.dtype)
print(y_test.dtype)
```

```
float64
float64
float64
float64
```

[ ]:
```
print("Shape of y_train:", y_train.shape)
print("Unique values in y_train:", np.unique(y_train))
```

```
Shape of y_train: (4000, 5)
Unique values in y_train: [0. 1.]
```

[ ]:
```
model.summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
conv1d_16 (Conv1D)	(None, 4996, 32)	1,952
batch_normalization_16 (BatchNormalization)	(None, 4996, 32)	128
max_pooling1d_16 (MaxPooling1D)	(None, 2498, 32)	0
conv1d_17 (Conv1D)	(None, 2496, 64)	6,208
batch_normalization_17 (BatchNormalization)	(None, 2496, 64)	256
max_pooling1d_17 (MaxPooling1D)	(None, 1248, 64)	0
flatten_8 (Flatten)	(None, 79872)	0
dense_16 (Dense)	(None, 128)	10,223,744
dropout_8 (Dropout)	(None, 128)	0
dense_17 (Dense)	(None, 5)	645

Total params: 10,232,933 (39.04 MB)

Trainable params: 10,232,741 (39.03 MB)

Non-trainable params: 192 (768.00 B)

[ ]:

```

```

[ ]:

```
print(len(np.unique(y_train)))
```

5

[ ]:

```
print(len(np.unique(y_train)))
```

5

[ ]:

```
from keras import regularizers

# Model architecture
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=5, activation='relu',␣
 ↪input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2]),␣
 ↪kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4)))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu',␣
 ↪kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4)))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4)))
model.add(Dropout(0.7))
model.add(Dense(len(np.unique(y_train)), activation='softmax',␣
 ↪kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4)))
```

```
c:\Program Files\Python312\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(
```

[ ]:

```python
# Compile the model with a lower learning rate and Adam optimizer
optimizer = Adam(learning_rate=0.0001)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer,␣
↪metrics=['accuracy'])

# Learning rate scheduler
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,␣
↪min_lr=0.00001)

# Train the model
history = model.fit(X_train_reshaped, y_train, epochs=20, batch_size=32,
                    validation_data=(X_test_reshaped, y_test),␣
↪callbacks=[reduce_lr])

# Model evaluation
accuracy = model.evaluate(X_test_reshaped, y_test)[1]
print(f"Test Accuracy: {accuracy}")

# Additional evaluation metrics
y_pred = model.predict(X_test_reshaped)
y_pred_classes = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_classes))
```

```
Epoch 1/20
125/125              60s 416ms/step -
accuracy: 0.2364 - loss: 4.5677 - val_accuracy: 0.2230 - val_loss: 3.9713 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125              51s 404ms/step -
accuracy: 0.4309 - loss: 2.9646 - val_accuracy: 0.2450 - val_loss: 6.8186 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125              52s 412ms/step -
accuracy: 0.5225 - loss: 2.0574 - val_accuracy: 0.2810 - val_loss: 5.3512 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125              53s 426ms/step -
accuracy: 0.5818 - loss: 1.7225 - val_accuracy: 0.3340 - val_loss: 3.9937 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125              50s 399ms/step -
accuracy: 0.6367 - loss: 1.5491 - val_accuracy: 0.3600 - val_loss: 2.9510 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125              50s 400ms/step -
accuracy: 0.6718 - loss: 1.4128 - val_accuracy: 0.3860 - val_loss: 2.5592 -
learning_rate: 1.0000e-04
```

```
Epoch 7/20
125/125             51s 411ms/step -
accuracy: 0.6957 - loss: 1.3185 - val_accuracy: 0.3810 - val_loss: 2.4557 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125             53s 421ms/step -
accuracy: 0.7274 - loss: 1.2368 - val_accuracy: 0.3740 - val_loss: 2.4488 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125             55s 440ms/step -
accuracy: 0.7378 - loss: 1.1692 - val_accuracy: 0.3860 - val_loss: 2.5113 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125             57s 454ms/step -
accuracy: 0.7506 - loss: 1.1584 - val_accuracy: 0.3780 - val_loss: 2.5382 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125             55s 440ms/step -
accuracy: 0.7779 - loss: 1.0608 - val_accuracy: 0.3970 - val_loss: 2.4981 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125             56s 449ms/step -
accuracy: 0.7880 - loss: 1.0614 - val_accuracy: 0.3880 - val_loss: 2.6252 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125             64s 511ms/step -
accuracy: 0.7904 - loss: 1.0632 - val_accuracy: 0.3980 - val_loss: 2.5503 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125             61s 491ms/step -
accuracy: 0.8199 - loss: 0.9900 - val_accuracy: 0.3930 - val_loss: 2.5347 -
learning_rate: 1.0000e-05
Epoch 15/20
125/125             58s 467ms/step -
accuracy: 0.8098 - loss: 0.9955 - val_accuracy: 0.3970 - val_loss: 2.5286 -
learning_rate: 1.0000e-05
Epoch 16/20
125/125             63s 500ms/step -
accuracy: 0.8267 - loss: 0.9600 - val_accuracy: 0.3960 - val_loss: 2.5184 -
learning_rate: 1.0000e-05
Epoch 17/20
125/125             59s 468ms/step -
accuracy: 0.8247 - loss: 0.9407 - val_accuracy: 0.3960 - val_loss: 2.5431 -
learning_rate: 1.0000e-05
Epoch 18/20
125/125             59s 471ms/step -
accuracy: 0.8280 - loss: 0.9163 - val_accuracy: 0.3990 - val_loss: 2.5554 -
learning_rate: 1.0000e-05
```

```
Epoch 19/20
125/125              60s 476ms/step -
accuracy: 0.8471 - loss: 0.8891 - val_accuracy: 0.3990 - val_loss: 2.5312 -
learning_rate: 1.0000e-05
Epoch 20/20
125/125              66s 526ms/step -
accuracy: 0.8394 - loss: 0.9026 - val_accuracy: 0.3990 - val_loss: 2.5649 -
learning_rate: 1.0000e-05
32/32              4s 113ms/step -
accuracy: 0.3941 - loss: 2.6535
Test Accuracy: 0.39899998903274536
32/32              3s 95ms/step
              precision    recall  f1-score   support

           0       0.44      0.34      0.38       196
           1       0.51      0.45      0.48       224
           2       0.25      0.29      0.27       186
           3       0.46      0.60      0.52       202
           4       0.32      0.30      0.31       192

    accuracy                           0.40      1000
   macro avg       0.40      0.39      0.39      1000
weighted avg       0.40      0.40      0.40      1000
```

[ ]:
```
model.summary()
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
conv1d_20 (Conv1D)	(None, 4996, 32)	1,952
batch_normalization_20 (BatchNormalization)	(None, 4996, 32)	128
max_pooling1d_20 (MaxPooling1D)	(None, 2498, 32)	0
conv1d_21 (Conv1D)	(None, 2496, 64)	6,208
batch_normalization_21 (BatchNormalization)	(None, 2496, 64)	256
max_pooling1d_21 (MaxPooling1D)	(None, 1248, 64)	0
flatten_10 (Flatten)	(None, 79872)	0

dense_20 (Dense)	(None, 128)	10,223,744
dropout_10 (Dropout)	(None, 128)	0
dense_21 (Dense)	(None, 5)	645

Total params: 30,698,417 (117.11 MB)

Trainable params: 10,232,741 (39.03 MB)

Non-trainable params: 192 (768.00 B)

Optimizer params: 20,465,484 (78.07 MB)