Dataprocessing+train+inference7

January 21, 2024

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

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

def read_and_concatenate_files_with_labels_and_user(folder_paths,
     column_names):
    # Initialize an empty DataFrame
    result_df = pd.DataFrame(columns=column_names + ['Label',
                    'User'])

    # Iterate through folder paths
    for folder_label, folder_path in zip(['Reading',
                     'Speaking', 'Watching'],
                     folder_paths):
        # Initialize an empty list to store DataFrames
        dfs = []

        # Get a sorted list of files in the folder
        files_to_process = sorted([file_name for
    file_name in os.listdir(folder_path)
    if file_name.endswith('.csv')])

        # Iterate through sorted files in the folder
        for file_name in files_to_process:
            file_path = os.path.join(folder_path, file_name)

            # Read the CSV file without column names and concatenate rows
            df = pd.read_csv(file_path, header=None,
                             names=column_names)

            # Extract numerical user information from
            #the file name using regular expression
            user_match = re.search(r'(\d+)', file_name)
            user_info = int(user_match.group(1)) if \
            user_match else None
```

```
            # Add 'Label' and 'User' columns
            df['Label'] = folder_label
            df['User'] = user_info
            dfs.append(df)

            # Print statement for debugging
            #print(f"Processed file: {file_name},
            #User: {user_info}, Label: {folder_label}")

        # Concatenate the list of DataFrames vertically
        result_df = pd.concat([result_df, pd.concat(dfs,
                    ignore_index=True)], ignore_index=True)

    return result_df

# Example usage:
folder_paths = ['Data/Reading',
    'Data/Speaking', 'Data/Watching']
column_names = ['EEG1', 'EEG2', 'Acc_X', 'Acc_Y', 'Acc_Z']

result_dataframe = read_and_concatenate_files_with_labels_and_user(folder_paths,
                 column_names)

# Print the unique values in the "User" column
#print(result_dataframe['User'].unique())

# Display the resulting DataFrame
print(result_dataframe)
```

```
              EEG1        EEG2       Acc_X       Acc_Y       Acc_Z     Label  \
0       842.229919  847.164856 -656.251038  789.063721  136.718964   Reading
1       845.519897  853.744812 -660.157288  792.969971  136.718964   Reading
2       847.164856  858.679748 -656.251038  792.969971  136.718964   Reading
3       843.874939  852.099793 -656.251038  792.969971  140.625214   Reading
4       847.164856  857.034729 -656.251038  792.969971  136.718964   Reading
…              …           …           …           …           …         …
104475  847.164856  857.034729 -703.126099  750.001160  136.718964  Watching
104476  875.129517  837.294983 -703.126099  750.001160  136.718964  Watching
104477  852.099793  837.294983 -703.126099  750.001160  136.718964  Watching
104478  832.360046  870.194580 -707.032349  746.094910  132.812714  Watching
104479  843.874939  843.874939 -703.126099  746.094910  136.718964  Watching

       User
0         1
1         1
2         1
```

```
3         1
4         1
…       …
104475    6
104476    6
104477    6
104478    6
104479    6

[104480 rows x 7 columns]
```

[ ]:
```
User = result_dataframe['User']
```

[ ]:
```
# Separate the data into features (X) and target variable (y)
X = result_dataframe[['EEG1', 'EEG2']]  # Features for EEG1 and EEG2
y = result_dataframe['Label']  # Target variable

# Filter data for User 1
X_test = X[User == 1]
y_test = y[User == 1]

# Filter data for training (excluding User 1)
X_train = X[User != 1]
y_train = y[User != 1]

# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train: " ,y_train.shape)
print("y_test: ", y_test.shape)
```

```
X_train shape: (93537, 2)
X_test shape: (10943, 2)
y_train:  (93537,)
y_test:  (10943,)
```

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

encoder = OneHotEncoder(sparse_output=False)


y_train = encoder.fit_transform(y_train.values.reshape(-1, 1))
y_test = encoder.transform(y_test.values.reshape(-1, 1))
X_train = X_train.values
X_test = X_test.values
```

[ ]:
```python
# Function to create sliding windows
import numpy as np
def create_sliding_windows(data, window_size):
    windows = []
    for i in range(len(data) - window_size + 1):
        windows.append(data[i:i+window_size])
    return np.array(windows)
```

[ ]:
```python
window_size=500
X_train_windows = create_sliding_windows(X_train, window_size=window_size)
X_test_windows = create_sliding_windows(X_test, window_size=window_size)
y_train_windows = create_sliding_windows(y_train, window_size=window_size)
y_test_windows = create_sliding_windows(y_test, window_size=window_size)
y_train_flat = y_train_windows[:, -1, :]
y_test_flat = y_test_windows[:, -1, :]

mean_values = X_train.mean(axis=(0, 1))
std_values = X_train.std(axis=(0, 1))
# Compute min and max values from the training data
min_values = X_train.min(axis=(0, 1))
max_values = X_train.max(axis=(0, 1))
X_train_windows = (X_train_windows - min_values) / (max_values - min_values)
X_test_windows = (X_test_windows - min_values) / (max_values - min_values)
```

[ ]:
```python
import tensorflow as tf
from tensorflow import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, \
Flatten, Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras import regularizers
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization


# Assuming 'X' is your input data of shape
#(1044, 100, 6) and 'y' is your corresponding labels
train_steps_per_epoch = len(X_train_windows)
val_steps_per_epoch  = len(X_test_windows)
# Define the CNN model

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, \
MaxPooling1D, Flatten, Dense, Dropout, LSTM
from tensorflow.keras import regularizers
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.callbacks import LearningRateScheduler
```

```
from tensorflow.keras.callbacks import EarlyStopping

tf.random.set_seed(42)




model = Sequential([
    Conv1D(filters=32, kernel_size=3, activation='relu',
            input_shape=(X_train_windows.shape[1], X_train_windows.shape[2])),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(128, activation='relu'),
    BatchNormalization(),
   Dense(128, activation='tanh'),
    Dropout(0.5),
    Dense(32, activation = 'tanh', kernel_regularizer=regularizers.l2(0.01)),
    Dense(3, activation='softmax')
])








early_stopping = EarlyStopping(monitor='val_loss',
                     patience=50, restore_best_weights=True)


# Compile the model

model.compile(optimizer=optimizers.Adam(learning_rate=0.00001),
              loss=tf.keras.losses.CategoricalCrossentropy()
,
              metrics=['accuracy'])

# Train the model

history = model.fit(X_train_windows, y_train_flat,
                    epochs=50, batch_size=64,
                    validation_data=(X_test_windows, y_test_flat),
                    callbacks=[early_stopping]
                  )
```

```python
#   steps_per_epoch=train_steps_per_epoch,

#  validation_steps=val_steps_per_epoch



# Accessing the history of training
training_accuracy = history.history.get('accuracy') \
    or history.history.get('acc')
training_loss = history.history['loss']
validation_accuracy = history.history.get('val_accuracy')\
or history.history.get('val_acc')
validation_loss = history.history.get('val_loss') \
or history.history.get('validation_loss')

# Plotting accuracy
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
if training_accuracy:
    plt.plot(training_accuracy, label='Training Accuracy')
if validation_accuracy:
    plt.plot(validation_accuracy, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

# Plotting loss
plt.subplot(1, 2, 2)
plt.plot(training_loss, label='Training Loss')
if validation_loss:
    plt.plot(validation_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

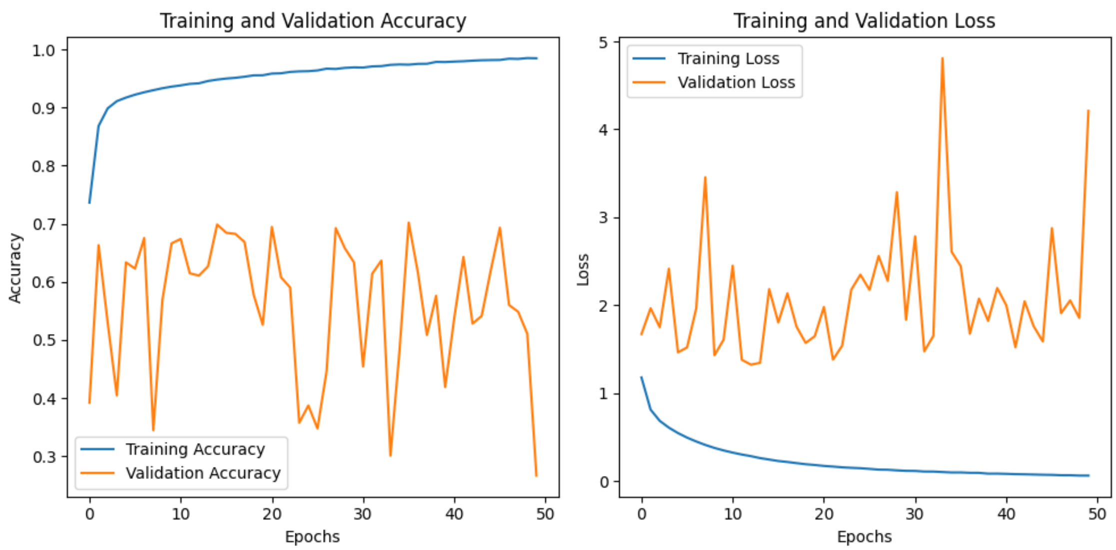
```
2024-01-21 01:28:45.870140: I tensorflow/core/util/port.cc:113] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-01-21 01:28:45.878271: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
```

```
2024-01-21 01:28:45.903536: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-01-21 01:28:45.903564: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-01-21 01:28:45.904432: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-01-21 01:28:45.909123: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-01-21 01:28:45.909544: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2024-01-21 01:28:46.721799: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
2024-01-21 01:28:47.787074: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-21 01:28:47.787624: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:2256] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are installed
properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices…

Epoch 1/50
1454/1454 [==============================] - 27s 18ms/step - loss: 1.1811 -
accuracy: 0.7365 - val_loss: 1.6740 - val_accuracy: 0.3920
Epoch 2/50
1454/1454 [==============================] - 24s 17ms/step - loss: 0.8154 -
accuracy: 0.8682 - val_loss: 1.9673 - val_accuracy: 0.6632
Epoch 3/50
1454/1454 [==============================] - 25s 17ms/step - loss: 0.6880 -
accuracy: 0.8986 - val_loss: 1.7511 - val_accuracy: 0.5298
Epoch 4/50
1454/1454 [==============================] - 27s 19ms/step - loss: 0.6108 -
accuracy: 0.9110 - val_loss: 2.4171 - val_accuracy: 0.4046
```

```
Epoch 5/50
1454/1454 [==============================] - 32s 22ms/step - loss: 0.5492 -
accuracy: 0.9172 - val_loss: 1.4663 - val_accuracy: 0.6336
Epoch 6/50
1454/1454 [==============================] - 34s 24ms/step - loss: 0.4974 -
accuracy: 0.9222 - val_loss: 1.5225 - val_accuracy: 0.6229
Epoch 7/50
1454/1454 [==============================] - 31s 21ms/step - loss: 0.4534 -
accuracy: 0.9263 - val_loss: 1.9661 - val_accuracy: 0.6755
Epoch 8/50
1454/1454 [==============================] - 32s 22ms/step - loss: 0.4141 -
accuracy: 0.9299 - val_loss: 3.4561 - val_accuracy: 0.3446
Epoch 9/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.3803 -
accuracy: 0.9333 - val_loss: 1.4331 - val_accuracy: 0.5691
Epoch 10/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.3518 -
accuracy: 0.9360 - val_loss: 1.6096 - val_accuracy: 0.6662
Epoch 11/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.3271 -
accuracy: 0.9382 - val_loss: 2.4510 - val_accuracy: 0.6739
Epoch 12/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.3052 -
accuracy: 0.9408 - val_loss: 1.3819 - val_accuracy: 0.6148
Epoch 13/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.2868 -
accuracy: 0.9417 - val_loss: 1.3268 - val_accuracy: 0.6108
Epoch 14/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.2647 -
accuracy: 0.9457 - val_loss: 1.3470 - val_accuracy: 0.6267
Epoch 15/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.2481 -
accuracy: 0.9481 - val_loss: 2.1851 - val_accuracy: 0.6988
Epoch 16/50
1454/1454 [==============================] - 32s 22ms/step - loss: 0.2316 -
accuracy: 0.9500 - val_loss: 1.8051 - val_accuracy: 0.6845
Epoch 17/50
1454/1454 [==============================] - 31s 21ms/step - loss: 0.2206 -
accuracy: 0.9512 - val_loss: 2.1373 - val_accuracy: 0.6826
Epoch 18/50
1454/1454 [==============================] - 33s 23ms/step - loss: 0.2074 -
accuracy: 0.9531 - val_loss: 1.7578 - val_accuracy: 0.6686
Epoch 19/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.1950 -
accuracy: 0.9554 - val_loss: 1.5732 - val_accuracy: 0.5779
Epoch 20/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.1864 -
accuracy: 0.9556 - val_loss: 1.6498 - val_accuracy: 0.5261
```

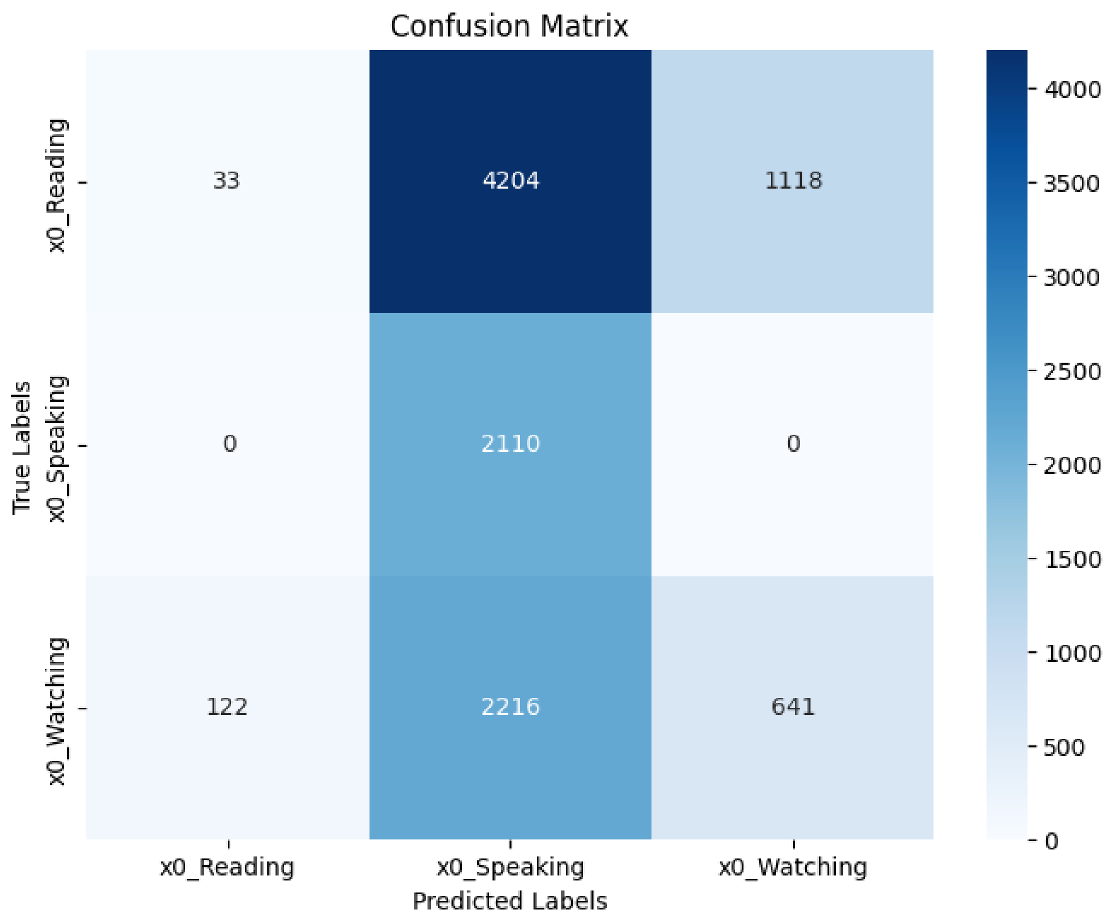
```
Epoch 21/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.1756 -
accuracy: 0.9585 - val_loss: 1.9819 - val_accuracy: 0.6947
Epoch 22/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.1684 -
accuracy: 0.9590 - val_loss: 1.3839 - val_accuracy: 0.6079
Epoch 23/50
1454/1454 [==============================] - 30s 21ms/step - loss: 0.1597 -
accuracy: 0.9613 - val_loss: 1.5414 - val_accuracy: 0.5904
Epoch 24/50
1454/1454 [==============================] - 31s 21ms/step - loss: 0.1541 -
accuracy: 0.9623 - val_loss: 2.1777 - val_accuracy: 0.3575
Epoch 25/50
1454/1454 [==============================] - 30s 20ms/step - loss: 0.1497 -
accuracy: 0.9627 - val_loss: 2.3492 - val_accuracy: 0.3872
Epoch 26/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.1422 -
accuracy: 0.9639 - val_loss: 2.1766 - val_accuracy: 0.3477
Epoch 27/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.1336 -
accuracy: 0.9670 - val_loss: 2.5618 - val_accuracy: 0.4464
Epoch 28/50
1454/1454 [==============================] - 29s 20ms/step - loss: 0.1321 -
accuracy: 0.9665 - val_loss: 2.2780 - val_accuracy: 0.6926
Epoch 29/50
1454/1454 [==============================] - 29s 20ms/step - loss: 0.1252 -
accuracy: 0.9683 - val_loss: 3.2867 - val_accuracy: 0.6579
Epoch 30/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.1200 -
accuracy: 0.9692 - val_loss: 1.8374 - val_accuracy: 0.6336
Epoch 31/50
1454/1454 [==============================] - 27s 19ms/step - loss: 0.1188 -
accuracy: 0.9689 - val_loss: 2.7853 - val_accuracy: 0.4543
Epoch 32/50
1454/1454 [==============================] - 31s 21ms/step - loss: 0.1120 -
accuracy: 0.9709 - val_loss: 1.4779 - val_accuracy: 0.6140
Epoch 33/50
1454/1454 [==============================] - 28s 20ms/step - loss: 0.1114 -
accuracy: 0.9713 - val_loss: 1.6537 - val_accuracy: 0.6366
Epoch 34/50
1454/1454 [==============================] - 29s 20ms/step - loss: 0.1064 -
accuracy: 0.9735 - val_loss: 4.8103 - val_accuracy: 0.3009
Epoch 35/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.1014 -
accuracy: 0.9742 - val_loss: 2.6095 - val_accuracy: 0.4830
Epoch 36/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.1016 -
accuracy: 0.9739 - val_loss: 2.4461 - val_accuracy: 0.7021
```

```
Epoch 37/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.0975 -
accuracy: 0.9752 - val_loss: 1.6773 - val_accuracy: 0.6169
Epoch 38/50
1454/1454 [==============================] - 27s 19ms/step - loss: 0.0971 -
accuracy: 0.9754 - val_loss: 2.0767 - val_accuracy: 0.5084
Epoch 39/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.0881 -
accuracy: 0.9786 - val_loss: 1.8242 - val_accuracy: 0.5760
Epoch 40/50
1454/1454 [==============================] - 27s 19ms/step - loss: 0.0883 -
accuracy: 0.9784 - val_loss: 2.1971 - val_accuracy: 0.4190
Epoch 41/50
1454/1454 [==============================] - 27s 19ms/step - loss: 0.0856 -
accuracy: 0.9791 - val_loss: 2.0003 - val_accuracy: 0.5387
Epoch 42/50
1454/1454 [==============================] - 27s 19ms/step - loss: 0.0822 -
accuracy: 0.9798 - val_loss: 1.5238 - val_accuracy: 0.6430
Epoch 43/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.0806 -
accuracy: 0.9807 - val_loss: 2.0470 - val_accuracy: 0.5282
Epoch 44/50
1454/1454 [==============================] - 32s 22ms/step - loss: 0.0780 -
accuracy: 0.9815 - val_loss: 1.7618 - val_accuracy: 0.5414
Epoch 45/50
1454/1454 [==============================] - 32s 22ms/step - loss: 0.0754 -
accuracy: 0.9818 - val_loss: 1.5902 - val_accuracy: 0.6202
Epoch 46/50
1454/1454 [==============================] - 29s 20ms/step - loss: 0.0738 -
accuracy: 0.9820 - val_loss: 2.8778 - val_accuracy: 0.6932
Epoch 47/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.0698 -
accuracy: 0.9841 - val_loss: 1.9118 - val_accuracy: 0.5602
Epoch 48/50
1454/1454 [==============================] - 27s 19ms/step - loss: 0.0696 -
accuracy: 0.9837 - val_loss: 2.0568 - val_accuracy: 0.5485
Epoch 49/50
1454/1454 [==============================] - 27s 19ms/step - loss: 0.0660 -
accuracy: 0.9852 - val_loss: 1.8575 - val_accuracy: 0.5107
Epoch 50/50
1454/1454 [==============================] - 28s 19ms/step - loss: 0.0656 -
accuracy: 0.9848 - val_loss: 4.2108 - val_accuracy: 0.2666
```



[ ]:
```python
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

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

# Convert one-hot encoded predictions back to labels
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_test_flat, axis=1)

# Create a confusion matrix
cm = confusion_matrix(y_true_labels, y_pred_labels)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=encoder.get_feature_names_out(),
            yticklabels=encoder.get_feature_names_out())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

# Print classification report
print("Classification Report:\n", classification_report(y_true_labels,␣
 ↪y_pred_labels))
```

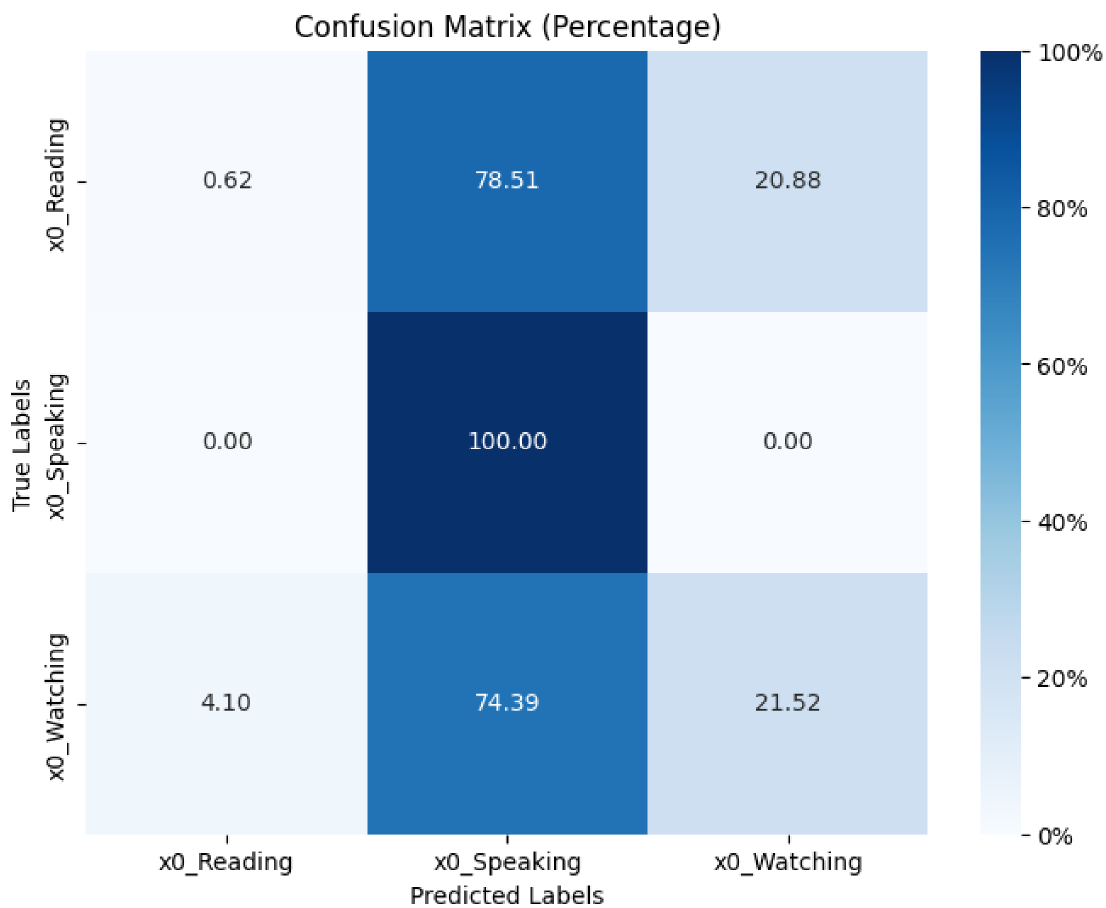
```
327/327 [==============================] - 1s 2ms/step
```




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

           0       0.21      0.01      0.01      5355
           1       0.25      1.00      0.40      2110
           2       0.36      0.22      0.27      2979

    accuracy                           0.27     10444
   macro avg       0.27      0.41      0.23     10444
weighted avg       0.26      0.27      0.16     10444
```

[ ]:
```
from matplotlib.ticker import FuncFormatter

# Calculate confusion matrix percentages
cm_percentage = (cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]) * 100

# Plot the confusion matrix with percentages
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cm_percentage, annot=True, fmt='.2f', cmap='Blues',
            xticklabels=encoder.get_feature_names_out(),
            yticklabels=encoder.get_feature_names_out(),
            cbar_kws={'format': FuncFormatter(lambda x, _: f'{x:.0f}%')})  #␣
 ↪Use FuncFormatter for direct formatting
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix (Percentage)')
plt.show()

# Print classification report
print("Classification Report:\n", classification_report(y_true_labels,␣
 ↪y_pred_labels))
```




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

0	0.21	0.01	0.01	5355
1	0.25	1.00	0.40	2110
2	0.36	0.22	0.27	2979
accuracy			0.27	10444
macro avg	0.27	0.41	0.23	10444
weighted avg	0.26	0.27	0.16	10444