Importing necessary libraries/modules

```
import cv2
import json
import numpy as np
import os
from scipy.io import loadmat
import dlib
from tqdm import tqdm
import scipy.io
import pickle

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, Ad
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_s
from sklearn.base import clone

from skimage.feature import hog
from skimage import data, color, exposure, io
from skimage.io import imread
from skimage.transform import resize
from skimage.filters import gabor
from skimage.feature import local_binary_pattern
from skimage.color import rgb2gray

from imblearn.over_sampling import SMOTE

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropou
from tensorflow.keras.optimizers import Adam

import optuna


%matplotlib inline
```

Gabor Features

Extracing Gabor features using the function gabor in scikit-image.

```
subjectIds = ['SN001', 'SN002', 'SN003', 'SN004', 'SN00
```

```
# Define Gabor filter parameters
orientations = [0, np.pi / 4, np.pi / 2, 3 * np.pi / 4]
frequencies = [0.2, 0.4, 0.6, 0.8]

def extract_gabor_features(image_path):
    image = imread(image_path, as_gray=True)
    gabor_responses = []
    for theta in orientations:
        for frequency in frequencies:
            # Apply Gabor filter
            filtered, _ = gabor(image, frequency=freque
            # Extract magnitude of the response and fla
            gabor_responses.extend(np.abs(filtered).fla

    # Compute histogram of Gabor responses
    hist, _ = np.histogram(gabor_responses, bins=20)
    hist = hist.astype(float) / hist.sum()  # Normalize

    return hist

# Calculate Gabor features for each cropped image
gabor_features = {}
for subject_id in tqdm(subjectIds, desc="Extracting Gab
    cropped_dir = os.path.join("croppedImg", subject_id
    cropped_images = [f for f in os.listdir(cropped_dir
    gabor_features[subject_id] = [extract_gabor_feature
```

```
Extracting Gabor features: 100%|██████████| 27/27
[12:04:19<00:00, 1609.61s/it]
```

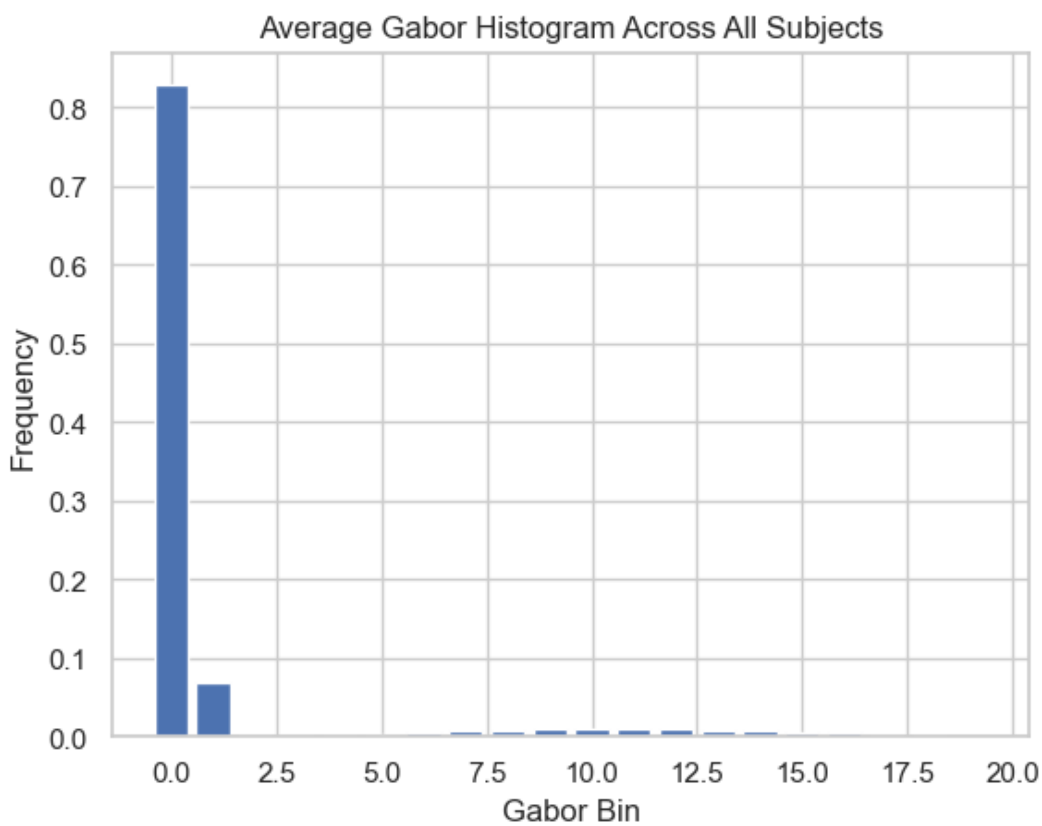
```
# Save the Gabor features to a file
with open('gabor_features.pkl', 'wb') as f:
    pickle.dump(gabor_features, f)
```

```
# To load the features back from the file
with open('gabor_features.pkl', 'rb') as f:
    gabor_features = pickle.load(f)
```

```
len(gabor_features) # Finding the number of features
```

Plotting Average Gabors across all subjects

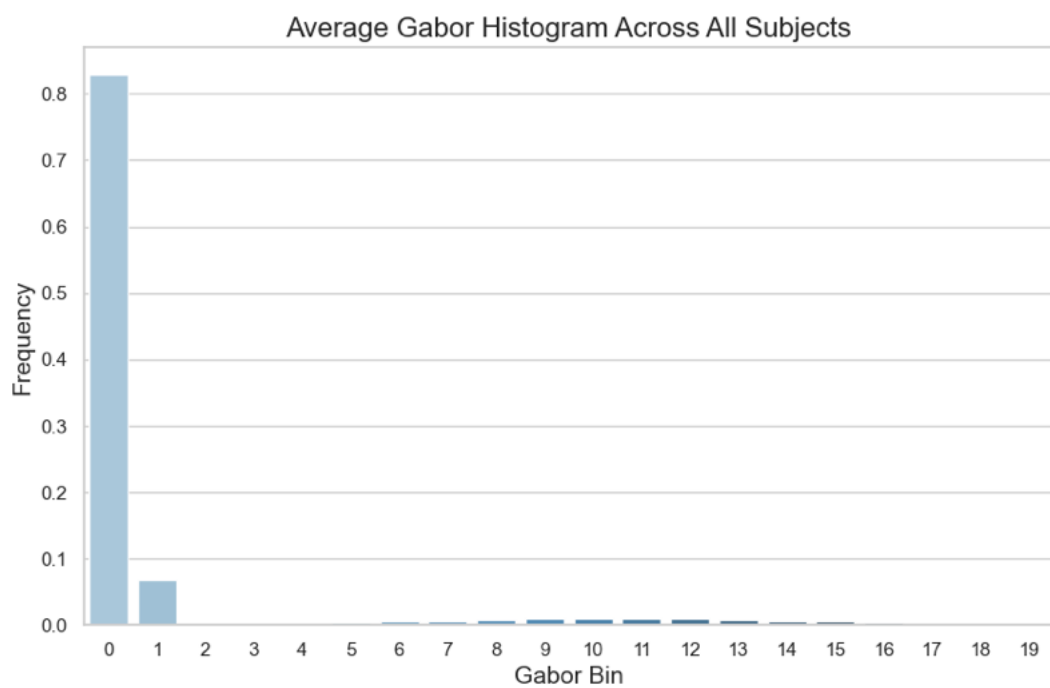
```
all_histograms = []
for subject_id in subjectIds:
    for hist in gabor_features[subject_id]:
        all_histograms.append(hist)

# Calculate the average histogram
average_histogram = np.mean(all_histograms, axis=0)

# Plot the average histogram
sns.set(style="whitegrid")
plt.bar(range(len(average_histogram)), average_histogra
plt.title('Average Gabor Histogram Across All Subjects'
plt.xlabel('Gabor Bin')
plt.ylabel('Frequency')
plt.show()
```




```
# Set the style of seaborn
sns.set(style="whitegrid")
```

```
# Set the figure size
plt.figure(figsize=(10, 6))

# Plot the average histogram with seaborn color palette
sns.barplot(x=list(range(len(average_histogram))), y=av

# Add titles and labels with a larger font size
plt.title('Average Gabor Histogram Across All Subjects'
plt.xlabel('Gabor Bin', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()
```

```
/var/folders/_t/k4q8g4jd7kq8f8kh6qvvgs_c0000gn/T/ipykernel_3287
0/1524029027.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and
will be removed in v0.14.0. Assign the `x` variable to `hue`
and set `legend=False` for the same effect.

  sns.barplot(x=list(range(len(average_histogram))),
y=average_histogram, palette="Blues_d")
```



Assigning Each subject its features using the JSON folder.

```
combined_features = []
labels = []
```

```
        for subject_id in subjectIds:
            json_file = os.path.join('json', f'{subject_id}.jso
            with open(json_file, 'r') as file:
                data = json.load(file)
                for i, frame_data in enumerate(data):
                    combined_features.append(gabor_features[sub

                    # Extract labels (AU intensities) with a de
                    au_intensities = [frame_data.get(f'au{j}',
                    labels.append(au_intensities)
```

Machine Learning Aspect

First, doing SVM using scikit-learn on the data where the target variable are the labels (AU labels).

```
        X = np.array(combined_features)
        y = np.array(labels)

        # Remove AUs with only one class
        var_threshold = 0.0  # Threshold for variance
        filtered_indices = [i for i in range(y.shape[1]) if np.
        y_filtered = y[:, filtered_indices]

        # Split the data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X,

        # Train a multi-output SVM classifier
        svm = SVC(kernel='linear', decision_function_shape='ovo
        multi_target_svm = MultiOutputClassifier(svm, n_jobs=-1
        multi_target_svm.fit(X_train, y_train)

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

        # Calculate accuracy for each AU
        accuracies = [accuracy_score(y_test[:, i], y_pred[:, i]
        average_accuracy = np.mean(accuracies)
        print(f'Average Accuracy: {average_accuracy}')
        for i, acc in enumerate(accuracies):
            print(f'Accuracy for AU{filtered_indices[i]+1}: {ac
```

```
Average Accuracy: 0.7931919526517565
```

```
Accuracy for AU1: 0.8008153904956046
Accuracy for AU2: 0.9550515989298
Accuracy for AU4: 0.7518155179003695
Accuracy for AU5: 0.9921518664798064
Accuracy for AU6: 0.882150592432157
Accuracy for AU9: 0.8602879347687603
Accuracy for AU12: 0.8383997961523761
Accuracy for AU17: 0.9937826474710154
Accuracy for AU20: 0.9942158236718053
Accuracy for AU25: 0.3347432793986495
Accuracy for AU26: 0.32169703146897694
```

```
#Calculating Precision, Recall, F1-score
precision = [precision_score(y_test[:, i], y_pred[:, i]
recall = [recall_score(y_test[:, i], y_pred[:, i], aver
f1 = [f1_score(y_test[:, i], y_pred[:, i], average='mac


average_recall = np.mean(recall)
print(f'Average Recall: {average_recall}')
for i, rec in enumerate(recall):
    print(f'Recall for AU{filtered_indices[i]+1}: {rec}

average_f1 = np.mean(f1)
print(f'Average F1: {average_f1}')
for i, f1 in enumerate(f1):
    print(f'F1 for AU{filtered_indices[i]+1}: {f1}')

average_precision = np.mean(precision)
print(f'Average Precision: {average_precision}')
for i, prec in enumerate(precision):
    print(f'Precision for AU{filtered_indices[i]+1}: {p
```

```
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
```

```
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

Average Recall: 0.23787878787878788
Recall for AU1: 0.16666666666666666
Recall for AU2: 0.16666666666666666
Recall for AU4: 0.16666666666666666
Recall for AU5: 0.3333333333333333
Recall for AU6: 0.25
Recall for AU9: 0.16666666666666666
Recall for AU12: 0.25
Recall for AU17: 0.3333333333333333
Recall for AU20: 0.3333333333333333
Recall for AU25: 0.2
Recall for AU26: 0.25
Average F1: 0.20812160900484783
F1 for AU1: 0.14823199807564416
F1 for AU2: 0.16283484955121688
F1 for AU4: 0.14305454545454546
F1 for AU5: 0.3320201580926556
F1 for AU6: 0.2343464428349015
F1 for AU9: 0.15414960209295
F1 for AU12: 0.22802433885901202
F1 for AU17: 0.3322938765772904
F1 for AU20: 0.332366507800621
F1 for AU25: 0.10031690275285403
F1 for AU26: 0.12169847696163485
Average Precision: 0.193838396707552
Precision for AU1: 0.13346923174926742
Precision for AU2: 0.1591752664883
Precision for AU4: 0.12530258631672825
Precision for AU5: 0.3307172888266021
Precision for AU6: 0.22053764810803925
Precision for AU9: 0.14338132246146004
Precision for AU12: 0.20959994903809404
```

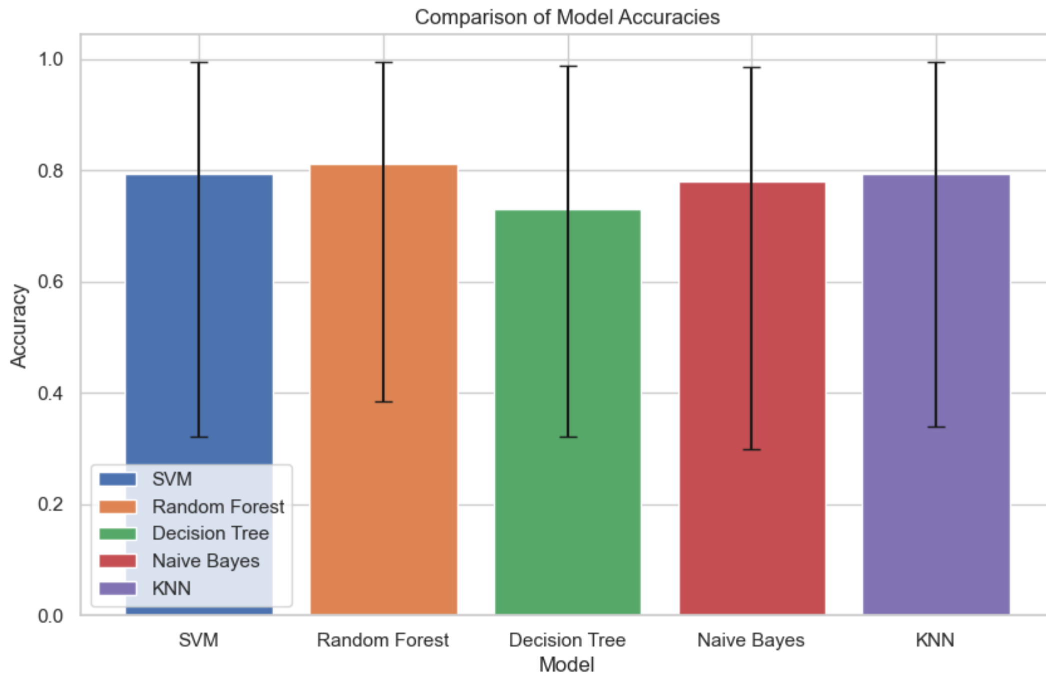
```
Precision for AU17: 0.33126088249033847
Precision for AU20: 0.3314052745572684
Precision for AU25: 0.0669486558797299
Precision for AU26: 0.08042425786724423
```

Comparing Various ML Classifiers

To find the best classifer on the dataset, we ran multiple classifers using MultiOutputClassifer in scikit-learn. Then, the accuracy of each classifer was plot using matplotlib libary.

```
X_train, X_test, y_train, y_test = train_test_split(X,

# Define the models to compare
models = {
    'SVM': MultiOutputClassifier(SVC(kernel='linear', d
    'Random Forest': MultiOutputClassifier(RandomForest
    'Decision Tree': MultiOutputClassifier(DecisionTree
    'Naive Bayes': MultiOutputClassifier(GaussianNB()),
    'KNN': MultiOutputClassifier(KNeighborsClassifier()
}

# Initialize a dictionary to store the accuracies for e
model_accuracies = {model_name: [] for model_name in mo

# Train each model and calculate accuracies
for model_name, model in models.items():
    print(f'Training {model_name}...')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracies = [accuracy_score(y_test[:, i], y_pred[:
    model_accuracies[model_name] = accuracies

# Calculate minimum, maximum, and average accuracies fo
model_stats = {model_name: {'min': min(acc), 'max': max
               for model_name, acc in model_accuracies.

# Plot the results
plt.figure(figsize=(10, 6))
for model_name, stats in model_stats.items():
    plt.bar(model_name, stats['avg'], yerr=[[stats['avg
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.legend()
plt.show()
```

```
Training SVM...
Training Random Forest...
Training Decision Tree...
Training Naive Bayes...
Training KNN...
```

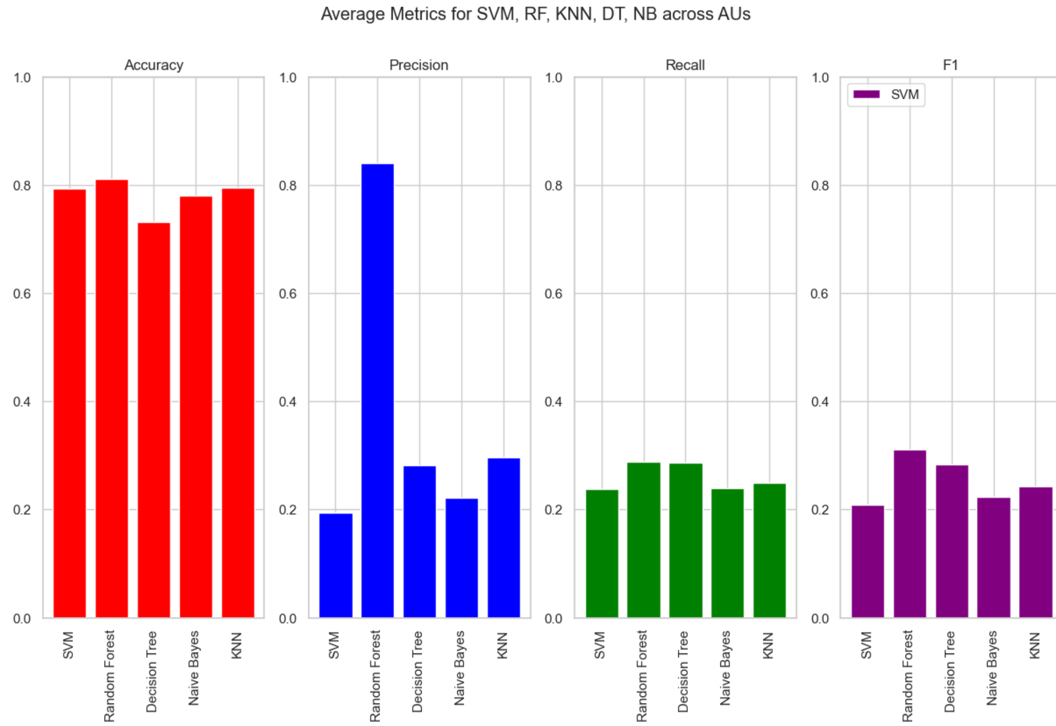


To get better comparison, we even compared other evaluation metrics like precision, recall, and F1 Score. Further plotted them for better understanding and comparison.

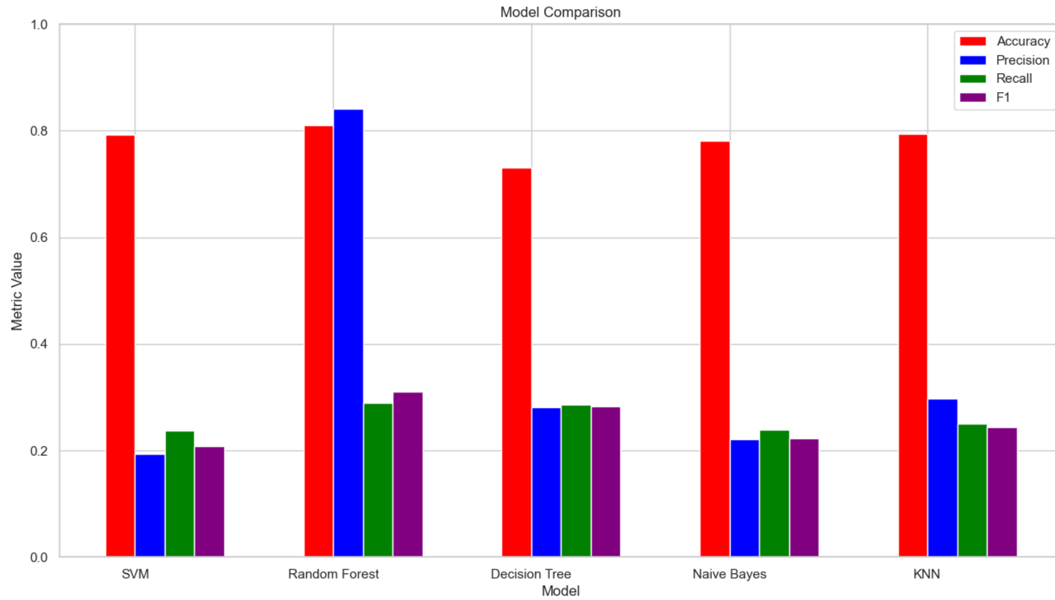
```
from sklearn.metrics import precision_score, recall_sco

# Initialize a dictionary to store the metrics for each
model_metrics = {
    model_name: {'Accuracy': [], 'Precision': [], 'Reca
    for model_name in models.keys()
}

# Calculate metrics for each model
for model_name, model in models.items():
    print(f'Training and evaluating {model_name}...')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Calculate metrics for each AU and average them
    accuracies = [accuracy_score(y_test[:, i], y_pred[:
    precisions = [precision_score(y_test[:, i], y_pred[
    recalls = [recall_score(y_test[:, i], y_pred[:, i],
    f1_scores = [f1_score(y_test[:, i], y_pred[:, i], a
```

```
            model_metrics[model_name]['Accuracy'] = np.mean(acc
            model_metrics[model_name]['Precision'] = np.mean(pr
            model_metrics[model_name]['Recall'] = np.mean(recal
            model_metrics[model_name]['F1'] = np.mean(f1_scores

        # Define the metrics
        metrics = ['Accuracy', 'Precision', 'Recall', 'F1']
        # Define colors for each model
        colors = ['red', 'blue', 'green', 'purple', 'orange']

        # Plot the results
        plt.figure(figsize=(15, 8))

        for i, metric in enumerate(metrics):
            plt.subplot(1, len(metrics), i+1)
            plt.bar(models.keys(), [model_metrics[model_name][m
            plt.title(metric)
            plt.ylim(0, 1)
            plt.xticks(rotation=90)


        plt.suptitle('Average Metrics for SVM, RF, KNN, DT, NB
        plt.legend(models.keys(), loc='upper left')
        plt.show()
```

```
Training and evaluating SVM...
Training and evaluating Random Forest...
Training and evaluating Decision Tree...
Training and evaluating Naive Bayes...
Training and evaluating KNN...
```




```python
import matplotlib.pyplot as plt

# Define the metrics and colors for each model
metrics = ['Accuracy', 'Precision', 'Recall', 'F1']
colors = ['red', 'blue', 'green', 'purple', 'orange']

# Plot the results without error bars
plt.figure(figsize=(15, 8))
bar_width = 0.15  # Width of the bars
for i, metric in enumerate(metrics):
    # Calculate positions for each bar
    positions = np.arange(len(models)) + i * bar_width

    # Extract mean for the metric
    means = [model_metrics[model_name][metric] for mode
    plt.bar(positions, means, bar_width, color=colors[i

plt.xticks(np.arange(len(models)) + bar_width / 2, mode
plt.title('Model Comparison')
plt.legend()
plt.ylabel('Metric Value')
plt.xlabel('Model')
plt.ylim(0, 1)
plt.show()
```



Running Random Forest Classifier

In the above plots, we saw RF is doing the best amongst all. So we evaluated for all AUs.

```
# Train a multi-output Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=100
multi_target_rf = MultiOutputClassifier(random_forest,
multi_target_rf.fit(X_train, y_train)

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

# Calculate accuracy for each AU
accuracies = [accuracy_score(y_test[:, i], y_pred[:, i]
average_accuracy = np.mean(accuracies)
print(f'Average Accuracy: {average_accuracy}')
for i, acc in enumerate(accuracies):
    print(f'Accuracy for AU{filtered_indices[i]+1}: {ac
```

```
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/joblib/externals/loky/process_executor.py:752:
UserWarning: A worker stopped while some jobs were given to the
executor. This can be caused by a too short worker timeout or
by a memory leak.
  warnings.warn(

Average Accuracy: 0.81104946779555
Accuracy for AU1: 0.810243343101032
Accuracy for AU2: 0.9571410370747866
```

```
Accuracy for AU4: 0.7644031086762645
Accuracy for AU5: 0.9926869664925468
Accuracy for AU6: 0.8883424640081539
Accuracy for AU9: 0.8674735635112754
Accuracy for AU12: 0.8457383106128169
Accuracy for AU17: 0.9940884189068672
Accuracy for AU20: 0.9946235189196075
Accuracy for AU25: 0.4226525672060135
Accuracy for AU26: 0.38415084724168685
```

```
#Calculating Precision, Recall, F1-score
precision = [precision_score(y_test[:, i], y_pred[:, i]
recall = [recall_score(y_test[:, i], y_pred[:, i], aver
f1 = [f1_score(y_test[:, i], y_pred[:, i], average='mac

average_recall = np.mean(recall)
print(f'Average Recall: {average_recall}')
for i, rec in enumerate(recall):
    print(f'Recall for AU{filtered_indices[i]+1}: {rec}

average_f1 = np.mean(f1)
print(f'Average F1: {average_f1}')
for i, f1 in enumerate(f1):
    print(f'F1 for AU{filtered_indices[i]+1}: {f1}')

average_precision = np.mean(precision)
print(f'Average Precision: {average_precision}')
for i, prec in enumerate(precision):
    print(f'Precision for AU{filtered_indices[i]+1}: {p
```

```
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/metrics/_classification.py:1509:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

Average Recall: 0.2887257336955567
Recall for AU1: 0.20783152663856
Recall for AU2: 0.21128126747641893
Recall for AU4: 0.21075824158549636
Recall for AU5: 0.373015873015873
Recall for AU6: 0.2910614292358913
Recall for AU9: 0.21115081433060542
Recall for AU12: 0.28856505567788543
Recall for AU17: 0.3508771929824561
```

```
Recall for AU20: 0.3877002413868773
Recall for AU25: 0.2869794743492667
Recall for AU26: 0.35676195397179306
Average F1: 0.3100196273662155
F1 for AU1: 0.2274636704938664
F1 for AU2: 0.2473034648764436
F1 for AU4: 0.22790869461527286
F1 for AU5: 0.40685683875254125
F1 for AU6: 0.3129014330493045
F1 for AU9: 0.23915057199049908
F1 for AU12: 0.3022880226461355
F1 for AU17: 0.3656781791378003
F1 for AU20: 0.432725304368095
F1 for AU25: 0.29786772125634364
F1 for AU26: 0.35007199984206855
Average Precision: 0.8360692823073361
Precision for AU1: 0.9345128361687803
Precision for AU2: 0.984719582888773
Precision for AU4: 0.9085415339319883
Precision for AU5: 0.9975610170643824
Precision for AU6: 0.941889550855297
Precision for AU9: 0.9537131233806484
Precision for AU12: 0.8635350992607932
Precision for AU17: 0.6646955369204496
Precision for AU20: 0.9982071086866009
Precision for AU25: 0.5797789378108416
Precision for AU26: 0.36960777841214165
```

Boosting

Next, we did boosting using the AdaBoostClassifier in the scikit-learn over RF as it was doing the best job. Further evaluation was also done.

```
# Train a multi-output AdaBoost classifier with RandomF
ada_boost = AdaBoostClassifier(RandomForestClassifier(n
multi_target_ada = MultiOutputClassifier(ada_boost, n_j
multi_target_ada.fit(X_train, y_train)

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

# Calculate accuracy, precision, recall, and F1 score f
accuracies = [accuracy_score(y_test[:, i], y_pred[:, i]
precisions = [precision_score(y_test[:, i], y_pred[:, i
recalls = [recall_score(y_test[:, i], y_pred[:, i], ave
f1_scores = [f1_score(y_test[:, i], y_pred[:, i], avera

# Calculate and print average metrics
```

```python
        average_accuracy = np.mean(accuracies)
        average_precision = np.mean(precisions)
        average_recall = np.mean(recalls)
        average_f1_score = np.mean(f1_scores)

        print(f'Average Accuracy: {average_accuracy}')
        print(f'Average Precision: {average_precision}')
        print(f'Average Recall: {average_recall}')
        print(f'Average F1 Score: {average_f1_score}')

        for i, (acc, prec, rec, f1) in enumerate(zip(accuracies
            print(f'AU{filtered_indices[i]+1}: Accuracy={acc},
```

```
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
```

```
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/joblib/externals/loky/process_executor.py:752:
UserWarning: A worker stopped while some jobs were given to the
executor. This can be caused by a too short worker timeout or
by a memory leak.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/Users/aayushgupta/miniconda3/envs/Soft_Vul/lib/python3.12/site
-packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
```

Bagging Classifier

```
# Train a multi-output AdaBoost classifier with RandomF
ada_boost = AdaBoostClassifier(BaggingClassifier(n_esti
multi_target_ada = MultiOutputClassifier(ada_boost, n_j
multi_target_ada.fit(X_train, y_train)
```

```python
    # Make predictions on the test set
    y_pred = multi_target_ada.predict(X_test)

    # Calculate accuracy, precision, recall, and F1 score f
    accuracies = [accuracy_score(y_test[:, i], y_pred[:, i]
    precisions = [precision_score(y_test[:, i], y_pred[:, i
    recalls = [recall_score(y_test[:, i], y_pred[:, i], ave
    f1_scores = [f1_score(y_test[:, i], y_pred[:, i], avera

    # Calculate and print average metrics
    average_accuracy = np.mean(accuracies)
    average_precision = np.mean(precisions)
    average_recall = np.mean(recalls)
    average_f1_score = np.mean(f1_scores)

    print(f'Average Accuracy: {average_accuracy}')
    print(f'Average Precision: {average_precision}')
    print(f'Average Recall: {average_recall}')
    print(f'Average F1 Score: {average_f1_score}')

    for i, (acc, prec, rec, f1) in enumerate(zip(accuracies
        print(f'AU{filtered_indices[i]+1}: Accuracy={acc},
```

```
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
```

```
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(
/users/aayush/latest/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519:
FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm
to circumvent this warning.
  warnings.warn(

Average Accuracy: 0.9658479920345169
Average Precision: 0.915027904269168
Average Recall: 0.774536602904949
Average F1 Score: 0.8238689305552155
AU1: Accuracy=0.9528708927978758, Precision=0.9191379335675495,
Recall=0.7797899816519326, F1 Score=0.8397201273249246
AU2: Accuracy=0.9857285097909061, Precision=0.9493271953538639,
Recall=0.7182744227381058, F1 Score=0.8063590686328582
AU4: Accuracy=0.9780949220046465, Precision=0.936412528432676,
Recall=0.824809791987304, F1 Score=0.8645768524398263
AU5: Accuracy=0.994357782940590 7, Precision=0.6647765176784523,
```

```
Recall=0.5057471264367815, F1 Score=0.5596582989959809
AU6: Accuracy=0.9581812147361434, Precision=0.9352591342907706,
Recall=0.8779744942797302, F1 Score=0.9047258471146493
AU9: Accuracy=0.9880517756388981, Precision=0.979719056384548,
Recall=0.6961560356530082, F1 Score=0.7771586405881696
AU12: Accuracy=0.9349485562562231,
Precision=0.9172899721411275, Recall=0.8680176681053431, F1
Score=0.8907461876571166
AU17: Accuracy=0.9960172585462994,
Precision=0.9711066488174921, Recall=0.6886651752423565, F1
Score=0.7709346969847416
AU25: Accuracy=0.9266511782276801,
Precision=0.9298623257893348, Recall=0.838891000717164, F1
Score=0.8776476162441776
AU26: Accuracy=0.9435778294059077, Precision=0.947387730235865,
Recall=0.9470403322377642, F1 Score=0.9471619695697099
```

SMOTE boosting

```
class MultiOutputAdaBoost:
    def __init__(self, base_estimator=None, n_estimator
        self.base_estimator = base_estimator if base_es
        self.n_estimators = n_estimators
        self.estimators_ = []

    def fit(self, X, Y):
        self.estimators_ = []
        smote = SMOTE()  # Initialize SMOTE
        for i in range(Y.shape[1]):
            X_resampled, y_resampled = smote.fit_resamp
            model = AdaBoostClassifier(base_estimator=c
            model.fit(X_resampled, y_resampled)
            self.estimators_.append(model)
        return self

    def predict(self, X):
        predictions = []
        for model in self.estimators_:
            predictions.append(model.predict(X).reshape
        return np.hstack(predictions)

# Initialize the multi-output AdaBoost classifier with
multi_output_adaboost = MultiOutputAdaBoost(base_estima

# Train the classifier
multi_output_adaboost.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = multi_output_adaboost.predict(X_test)

# Calculate accuracy for each AU and average accuracy
accuracies = [accuracy_score(y_test[:, i], y_pred[:, i]
average_accuracy = np.mean(accuracies)

# Print the results
print(f'Average Accuracy: {average_accuracy}')
for i, acc in enumerate(accuracies):
    print(f'Accuracy for AU{filtered_indices[i]+1}: {ac
```

```
Average Accuracy: 0.7459012731740006
Accuracy for AU1: 0.6563144963144963
Accuracy for AU2: 0.9292383292383293
Accuracy for AU4: 0.6165110565110565
Accuracy for AU5: 0.9875184275184276
Accuracy for AU6: 0.8112039312039312
Accuracy for AU9: 0.7728746928746929
Accuracy for AU12: 0.753022113022113
Accuracy for AU17: 0.9897788697788698
Accuracy for AU20: 0.9908599508599508
Accuracy for AU25: 0.36324324324324325
Accuracy for AU26: 0.33434889434889437
```

```
# Calculate precision, recall, and F1-score for each AU
precision = [precision_score(y_test[:, i], y_pred[:, i]
recall = [recall_score(y_test[:, i], y_pred[:, i], aver
f1 = [f1_score(y_test[:, i], y_pred[:, i], average='mac

average_recall = np.mean(recall)
print(f'Average Recall: {average_recall}')
for i, rec in enumerate(recall):
    print(f'Recall for AU{filtered_indices[i]+1}: {rec}

average_f1 = np.mean(f1)
print(f'Average F1: {average_f1}')
for i, f1 in enumerate(f1):
    print(f'F1 for AU{filtered_indices[i]+1}: {f1}')

average_precision = np.mean(precision)
print(f'Average Precision: {average_precision}')
for i, prec in enumerate(precision):
    print(f'Precision for AU{filtered_indices[i]+1}: {p
```

```
Average Recall: 0.2574286326531309
```

```
Recall for AU1: 0.17424702386930757
Recall for AU2: 0.17559957862512435
Recall for AU4: 0.18098681787299856
Recall for AU5: 0.33178140993891364
Recall for AU6: 0.2512745428044791
Recall for AU9: 0.17772054267978113
Recall for AU12: 0.2862386790175889
Recall for AU17: 0.3318833415719229
Recall for AU20: 0.33211450406825443
Recall for AU25: 0.2678658311928507
Recall for AU26: 0.32200268754321876
Average F1: 0.25649593121850567
F1 for AU1: 0.1729720665659573
F1 for AU2: 0.1770466704872019
F1 for AU4: 0.18096488691066084
F1 for AU5: 0.3312400072524683
F1 for AU6: 0.2502982434462411
F1 for AU9: 0.17803680437571814
F1 for AU12: 0.2858999352798279
F1 for AU17: 0.3316210609503112
F1 for AU20: 0.3318029981405605
F1 for AU25: 0.2608647037499556
F1 for AU26: 0.3207078662446595
Average Precision: 0.25688486834141105
Precision for AU1: 0.17229283631525513
Precision for AU2: 0.1801074636153341
Precision for AU4: 0.18227732370210117
Precision for AU5: 0.3307003686150605
Precision for AU6: 0.2513394759053566
Precision for AU9: 0.17964493549615632
Precision for AU12: 0.28682985342196105
Precision for AU17: 0.33135919455137697
Precision for AU20: 0.3314920760176235
Precision for AU25: 0.2594667677093767
Precision for AU26: 0.32022325640591975
```

RNN

Two LSTM Layers:

1. 50 units and returns sequences. Here, the output is the full sequence to the next layer which is useful for maintaining temporal information throughout the network.
2. Also 50 units but no return sequence.

Two Dropout Layers

Positioned after each LSTM layer, with a dropout rate of 0.2. This will prevent overfitting by randomly setting a fraction of the input units to 0 at each update during training time. ##### One Dense Layer A fully connected layer that outputs predictions for each AU. It uses a sigmoid activation function because your model likely predicts the presence or absence of each AU as a binary classification task. The sigmoid function is ideal here because it squashes its input to a range between 0 and 1, representing a probability.

Total Layers and Hidden Layers:

Total Layers: 5 (2 LSTM, 2 Dropout, 1 Dense)

Hidden Layers: 4 (2 LSTM, 2 Dropout)

```
        # Create RNN model
        model = Sequential([
            LSTM(50, input_shape=(1, X_train.shape[2]), return_
            Dropout(0.2),
            LSTM(50),
            Dropout(0.2),
            Dense(y_filtered.shape[1], activation='sigmoid')
        ])

        # Compile the model
        model.compile(loss='binary_crossentropy', optimizer='ad

        # Fit the model on training data
        model.fit(X_train, y_train, epochs=50, batch_size=64, v

        # Evaluate the model
        scores = model.evaluate(X_test, y_test, verbose=0)
        print(f'RNN Model Accuracy: {scores[1]}')
```

```
2024-05-02 19:37:37.925603: I
tensorflow/core/platform/cpu_feature_guard.cc:210] This
TensorFlow binary is optimized to use available CPU
instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other
operations, rebuild TensorFlow with the appropriate compiler
flags.
2024-05-02 19:37:43.875680: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT
Warning: Could not find TensorRT
2024-05-02 19:37:57.228091: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:2251] 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...
/users/aayush/latest/lib/python3.10/site-
packages/keras/src/layers/rnn/rnn.py:204: 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__(**kwargs)

Epoch 1/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.1834 -
loss: -1.3062 - val_accuracy: 0.1814 - val_loss: -5.5130
Epoch 2/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1824 -
loss: -6.6794 - val_accuracy: 0.1814 - val_loss: -10.1278
Epoch 3/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1831 -
loss: -11.1855 - val_accuracy: 0.1814 - val_loss: -14.6815
Epoch 4/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1836 -
loss: -15.8497 - val_accuracy: 0.1814 - val_loss: -19.2196
Epoch 5/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1846 -
loss: -20.2840 - val_accuracy: 0.1814 - val_loss: -23.7636
Epoch 6/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1834 -
loss: -24.5989 - val_accuracy: 0.1814 - val_loss: -28.2993
Epoch 7/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1826 -
loss: -29.3429 - val_accuracy: 0.1814 - val_loss: -32.8307
Epoch 8/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1822 -
loss: -33.6589 - val_accuracy: 0.1814 - val_loss: -37.3618
Epoch 9/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1818 -
loss: -38.2699 - val_accuracy: 0.1814 - val_loss: -41.8904
Epoch 10/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1821 -
loss: -42.9610 - val_accuracy: 0.1814 - val_loss: -46.4164
Epoch 11/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1839 -
loss: -47.4379 - val_accuracy: 0.1814 - val_loss: -50.9456
Epoch 12/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1841 -
loss: -51.5549 - val_accuracy: 0.1814 - val_loss: -55.4904
Epoch 13/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1811 -
```

```
loss: -56.2330 - val_accuracy: 0.1814 - val_loss: -60.0247
Epoch 14/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1859 -
loss: -61.1369 - val_accuracy: 0.1814 - val_loss: -64.5654
Epoch 15/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1840 -
loss: -65.8710 - val_accuracy: 0.1814 - val_loss: -69.0931
Epoch 16/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1860 -
loss: -70.3406 - val_accuracy: 0.1814 - val_loss: -73.6437
Epoch 17/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1824 -
loss: -74.0897 - val_accuracy: 0.1814 - val_loss: -78.1758
Epoch 18/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1842 -
loss: -78.2093 - val_accuracy: 0.1814 - val_loss: -82.7019
Epoch 19/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1835 -
loss: -84.4398 - val_accuracy: 0.1814 - val_loss: -87.2283
Epoch 20/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1810 -
loss: -86.7808 - val_accuracy: 0.1814 - val_loss: -91.7757
Epoch 21/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1819 -
loss: -92.4597 - val_accuracy: 0.1814 - val_loss: -96.3009
Epoch 22/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1839 -
loss: -97.4591 - val_accuracy: 0.1814 - val_loss: -100.8302
Epoch 23/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1837 -
loss: -101.5628 - val_accuracy: 0.1814 - val_loss: -105.3730
Epoch 24/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1827 -
loss: -106.1807 - val_accuracy: 0.1814 - val_loss: -109.9016
Epoch 25/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1829 -
loss: -110.1793 - val_accuracy: 0.1814 - val_loss: -114.4463
Epoch 26/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1835 -
loss: -116.1108 - val_accuracy: 0.1814 - val_loss: -118.9762
Epoch 27/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1828 -
loss: -119.1027 - val_accuracy: 0.1814 - val_loss: -123.5217
Epoch 28/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1815 -
loss: -125.2012 - val_accuracy: 0.1814 - val_loss: -128.0527
Epoch 29/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1843 -
```

```
loss: -128.5376 - val_accuracy: 0.1814 - val_loss: -132.5943
Epoch 30/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1830 -
loss: -132.4716 - val_accuracy: 0.1814 - val_loss: -137.1406
Epoch 31/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1866 -
loss: -137.7289 - val_accuracy: 0.1814 - val_loss: -141.6828
Epoch 32/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1821 -
loss: -140.5846 - val_accuracy: 0.1814 - val_loss: -146.2289
Epoch 33/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1829 -
loss: -147.7092 - val_accuracy: 0.1814 - val_loss: -150.7576
Epoch 34/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1836 -
loss: -151.6160 - val_accuracy: 0.1814 - val_loss: -155.2947
Epoch 35/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1843 -
loss: -155.6775 - val_accuracy: 0.1814 - val_loss: -159.8228
Epoch 36/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1852 -
loss: -160.0126 - val_accuracy: 0.1814 - val_loss: -164.3685
Epoch 37/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1822 -
loss: -165.8757 - val_accuracy: 0.1814 - val_loss: -168.8991
Epoch 38/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1820 -
loss: -168.7384 - val_accuracy: 0.1814 - val_loss: -173.4362
Epoch 39/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1834 -
loss: -174.9553 - val_accuracy: 0.1814 - val_loss: -177.9665
Epoch 40/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1825 -
loss: -180.0646 - val_accuracy: 0.1814 - val_loss: -182.5078
Epoch 41/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1833 -
loss: -182.5726 - val_accuracy: 0.1814 - val_loss: -187.0570
Epoch 42/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1833 -
loss: -189.7874 - val_accuracy: 0.1814 - val_loss: -191.5839
Epoch 43/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1806 -
loss: -191.7772 - val_accuracy: 0.1814 - val_loss: -196.1272
Epoch 44/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1825 -
loss: -197.4665 - val_accuracy: 0.1814 - val_loss: -200.6645
Epoch 45/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1820 -
```

```
loss: -199.9168 - val_accuracy: 0.1814 - val_loss: -205.2029
Epoch 46/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1837 -
loss: -208.3030 - val_accuracy: 0.1814 - val_loss: -209.7292
Epoch 47/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1837 -
loss: -208.3297 - val_accuracy: 0.1814 - val_loss: -214.2806
Epoch 48/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1827 -
loss: -213.8220 - val_accuracy: 0.1814 - val_loss: -218.8164
Epoch 49/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1808 -
loss: -218.8820 - val_accuracy: 0.1814 - val_loss: -223.3537
Epoch 50/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1845 -
loss: -222.7314 - val_accuracy: 0.1814 - val_loss: -227.8895
RNN Model Accuracy: 0.1813659369945526
```

Hyperparameterization using OPTUNA

```
def objective(trial):
    # Hyperparameters to be tuned
    lstm_units = trial.suggest_categorical('lstm_units'
    dropout_rate = trial.suggest_uniform('dropout_rate'
    learning_rate = trial.suggest_loguniform('learning_

    # Model construction
    model = Sequential([
        LSTM(lstm_units, input_shape=(1, X_train.shape[
        Dropout(dropout_rate),
        LSTM(lstm_units),
        Dropout(dropout_rate),
        Dense(y_filtered.shape[1], activation='sigmoid'
    ])

    # Compile the model
    model.compile(loss='binary_crossentropy', optimizer

    # Fit the model
    history = model.fit(X_train, y_train, epochs=50, ba

    # Objective value to minimize
    return history.history['val_loss'][-1]
```

```
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=10)
```

```
[I 2024-05-02 19:41:48,050] A new study created in memory with
name: no-name-91248581-7dda-4d0d-9113-067ab0c946bb
/tmp/ipykernel_12515/2497591657.py:11: FutureWarning:
suggest_uniform has been deprecated in v3.0.0. This feature
will be removed in v6.0.0. See
https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float instead.
  dropout_rate = trial.suggest_uniform('dropout_rate', 0.1,
0.5)
/tmp/ipykernel_12515/2497591657.py:12: FutureWarning:
suggest_loguniform has been deprecated in v3.0.0. This feature
will be removed in v6.0.0. See
https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float(..., log=True) instead.
  learning_rate = trial.suggest_loguniform('learning_rate', 1e-
5, 1e-2)
/users/aayush/latest/lib/python3.10/site-
packages/keras/src/layers/rnn/rnn.py:204: 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__(**kwargs)
[I 2024-05-02 19:45:48,379] Trial 0 finished with value:
-374.6436462402344 and parameters: {'lstm_units': 100,
'dropout_rate': 0.4474279918651616, 'learning_rate':
0.0008549848391553601}. Best is trial 0 with value:
-374.6436462402344.
[I 2024-05-02 19:49:50,185] Trial 1 finished with value:
-206.6237030029297 and parameters: {'lstm_units': 100,
'dropout_rate': 0.4315748657729971, 'learning_rate':
0.00046941656148111476}. Best is trial 0 with value:
-374.6436462402344.
[I 2024-05-02 19:52:51,922] Trial 2 finished with value:
-380.7765808105469 and parameters: {'lstm_units': 50,
'dropout_rate': 0.10773835915759072, 'learning_rate':
0.0016629719767406053}. Best is trial 2 with value:
-380.7765808105469.
[I 2024-05-02 19:55:53,831] Trial 3 finished with value:
-37.028358459472656 and parameters: {'lstm_units': 50,
'dropout_rate': 0.4302590477025331, 'learning_rate':
0.00016474123643579695}. Best is trial 2 with value:
-380.7765808105469.
[I 2024-05-02 20:01:15,211] Trial 4 finished with value:
-162.86119079589844 and parameters: {'lstm_units': 150,
'dropout_rate': 0.22017502716933604, 'learning_rate':
0.0002414231021603019}. Best is trial 2 with value:
-380.7765808105469.
[I 2024-05-02 20:08:25,895] Trial 5 finished with value:
```

```
-7.759028911590576 and parameters: {'lstm_units': 150,
'dropout_rate': 0.2012875183226059, 'learning_rate':
1.0701800358392484e-05}. Best is trial 2 with value:
-380.7765808105469.
[I 2024-05-02 20:23:36,951] Trial 6 finished with value:
-493.8099670410156 and parameters: {'lstm_units': 150,
'dropout_rate': 0.1608366146230273, 'learning_rate':
0.0007330124021490447}. Best is trial 6 with value:
-493.8099670410156.
[I 2024-05-02 20:32:24,063] Trial 7 finished with value:
-198.52203369140625 and parameters: {'lstm_units': 50,
'dropout_rate': 0.13124903284420647, 'learning_rate':
0.0008669079395207692}. Best is trial 6 with value:
-493.8099670410156.
[I 2024-05-02 20:44:19,695] Trial 8 finished with value:
-16.28822135925293 and parameters: {'lstm_units': 100,
'dropout_rate': 0.26740499247326843, 'learning_rate':
3.5018036485020174e-05}. Best is trial 6 with value:
-493.8099670410156.
[I 2024-05-02 20:53:05,209] Trial 9 finished with value:
-103.61286926269531 and parameters: {'lstm_units': 50,
'dropout_rate': 0.11909455444051478, 'learning_rate':
0.00045091977763594634}. Best is trial 6 with value:
-493.8099670410156.
```

```
print('Best trial:', study.best_trial.params)

# Rebuild and train the model with the best parameters
best_params = study.best_trial.params
model = Sequential([
    LSTM(best_params['lstm_units'], input_shape=(1, X_t
    Dropout(best_params['dropout_rate']),
    LSTM(best_params['lstm_units']),
    Dropout(best_params['dropout_rate']),
    Dense(y_filtered.shape[1], activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer=Ada
model.fit(X_train, y_train, epochs=50, batch_size=64, v

# Predictions
y_pred = model.predict(X_test)
y_pred_rounded = np.round(y_pred)  # Round predictions

# Calculate metrics for each AU
accuracies = [accuracy_score(y_test[:, i], y_pred_round
precisions = [precision_score(y_test[:, i], y_pred_roun
recalls = [recall_score(y_test[:, i], y_pred_rounded[:,
```

```python
        f1_scores = [f1_score(y_test[:, i], y_pred_rounded[:, i

        # Print metrics for each AU
        for i, (acc, prec, rec, f1) in enumerate(zip(accuracies
            print(f'AU{filtered_indices[i]+1}: Accuracy={acc:.4

        # Calculate and print average metrics
        average_accuracy = np.mean(accuracies)
        average_precision = np.mean(precisions)
        average_recall = np.mean(recalls)
        average_f1_score = np.mean(f1_scores)

        print(f'Best Model Average Accuracy: {average_accuracy:
        print(f'Best Model Average Precision: {average_precisio
        print(f'Best Model Average Recall: {average_recall:.4f}
        print(f'Best Model Average F1 Score: {average_f1_score:
```

```
Best trial: {'lstm_units': 150, 'dropout_rate':
0.1608366146230273, 'learning_rate': 0.0007330124021490447}
Epoch 1/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.1821
- loss: -3.2658 - val_accuracy: 0.1814 - val_loss: -11.9670
Epoch 2/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.1832 -
loss: -14.3201 - val_accuracy: 0.1814 - val_loss: -21.9474
Epoch 3/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1824 -
loss: -24.3552 - val_accuracy: 0.1814 - val_loss: -31.7918
Epoch 4/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.1842
- loss: -34.3083 - val_accuracy: 0.1814 - val_loss: -41.6481
Epoch 5/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1823 -
loss: -43.9948 - val_accuracy: 0.1814 - val_loss: -51.4993
Epoch 6/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1837 -
loss: -53.7660 - val_accuracy: 0.1814 - val_loss: -61.3371
Epoch 7/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1847 -
loss: -64.2598 - val_accuracy: 0.1814 - val_loss: -71.1723
Epoch 8/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.1836
- loss: -73.8583 - val_accuracy: 0.1814 - val_loss: -80.9896
Epoch 9/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1854 -
loss: -82.3735 - val_accuracy: 0.1814 - val_loss: -90.8665
Epoch 10/50
```

```
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1834 -
loss: -92.5576 - val_accuracy: 0.1814 - val_loss: -100.6992
Epoch 11/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1828 -
loss: -103.0416 - val_accuracy: 0.1814 - val_loss: -110.5308
Epoch 12/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1837 -
loss: -112.8720 - val_accuracy: 0.1814 - val_loss: -120.3586
Epoch 13/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1842 -
loss: -123.5474 - val_accuracy: 0.1814 - val_loss: -130.1588
Epoch 14/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1823 -
loss: -132.3140 - val_accuracy: 0.1814 - val_loss: -140.0184
Epoch 15/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1831 -
loss: -141.9195 - val_accuracy: 0.1814 - val_loss: -149.8403
Epoch 16/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1855 -
loss: -150.0950 - val_accuracy: 0.1814 - val_loss: -159.7024
Epoch 17/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1832 -
loss: -162.2418 - val_accuracy: 0.1814 - val_loss: -169.5336
Epoch 18/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1819 -
loss: -169.8916 - val_accuracy: 0.1814 - val_loss: -179.3831
Epoch 19/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1834 -
loss: -181.3383 - val_accuracy: 0.1814 - val_loss: -189.1992
Epoch 20/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1838 -
loss: -191.7570 - val_accuracy: 0.1814 - val_loss: -199.0177
Epoch 21/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1826 -
loss: -200.3775 - val_accuracy: 0.1814 - val_loss: -208.8486
Epoch 22/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1823 -
loss: -211.4456 - val_accuracy: 0.1814 - val_loss: -218.6710
Epoch 23/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1822 -
loss: -218.4956 - val_accuracy: 0.1814 - val_loss: -228.5188
Epoch 24/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1834 -
loss: -228.4209 - val_accuracy: 0.1814 - val_loss: -238.3535
Epoch 25/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1822 -
loss: -238.2001 - val_accuracy: 0.1814 - val_loss: -248.1911
Epoch 26/50
```

```
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1826 -
loss: -246.8616 - val_accuracy: 0.1814 - val_loss: -258.0431
Epoch 27/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1845 -
loss: -257.9693 - val_accuracy: 0.1814 - val_loss: -267.8697
Epoch 28/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1821 -
loss: -267.6242 - val_accuracy: 0.1814 - val_loss: -277.7088
Epoch 29/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1830 -
loss: -275.2669 - val_accuracy: 0.1814 - val_loss: -287.5693
Epoch 30/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1819 -
loss: -287.7704 - val_accuracy: 0.1814 - val_loss: -297.4003
Epoch 31/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1832 -
loss: -295.4328 - val_accuracy: 0.1814 - val_loss: -307.2607
Epoch 32/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1817 -
loss: -309.9949 - val_accuracy: 0.1814 - val_loss: -317.0871
Epoch 33/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1843 -
loss: -322.8765 - val_accuracy: 0.1814 - val_loss: -326.9557
Epoch 34/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1829 -
loss: -328.0951 - val_accuracy: 0.1814 - val_loss: -336.8303
Epoch 35/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1843 -
loss: -336.1337 - val_accuracy: 0.1814 - val_loss: -346.6892
Epoch 36/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1844 -
loss: -346.6607 - val_accuracy: 0.1814 - val_loss: -356.5066
Epoch 37/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1843 -
loss: -356.7512 - val_accuracy: 0.1814 - val_loss: -366.3376
Epoch 38/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.1838 -
loss: -368.4597 - val_accuracy: 0.1814 - val_loss: -376.1436
Epoch 39/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1846 -
loss: -373.6660 - val_accuracy: 0.1814 - val_loss: -385.9837
Epoch 40/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1830 -
loss: -385.9144 - val_accuracy: 0.1814 - val_loss: -395.8244
Epoch 41/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1846 -
loss: -397.9948 - val_accuracy: 0.1814 - val_loss: -405.6595
Epoch 42/50
```

```
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1856 -
loss: -404.0225 - val_accuracy: 0.1814 - val_loss: -415.4825
Epoch 43/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1814 -
loss: -415.8937 - val_accuracy: 0.1814 - val_loss: -425.3173
Epoch 44/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1819 -
loss: -425.2364 - val_accuracy: 0.1814 - val_loss: -435.1375
Epoch 45/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1856 -
loss: -434.7177 - val_accuracy: 0.1814 - val_loss: -444.9690
Epoch 46/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1805 -
loss: -445.9583 - val_accuracy: 0.1814 - val_loss: -454.7777
Epoch 47/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1829 -
loss: -458.7742 - val_accuracy: 0.1814 - val_loss: -464.6208
Epoch 48/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1843 -
loss: -462.1565 - val_accuracy: 0.1814 - val_loss: -474.4689
Epoch 49/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1810 -
loss: -475.8527 - val_accuracy: 0.1814 - val_loss: -484.3018
Epoch 50/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1812 -
loss: -483.9254 - val_accuracy: 0.1814 - val_loss: -494.1577
1181/1181 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
AU1: Accuracy=0.0210, Precision=0.0035, Recall=0.1667, F1
Score=0.0068
AU2: Accuracy=0.9557, Precision=0.1593, Recall=0.1667, F1
Score=0.1629
AU4: Accuracy=0.0248, Precision=0.0041, Recall=0.1667, F1
Score=0.0081
AU5: Accuracy=0.9922, Precision=0.3307, Recall=0.3333, F1
Score=0.3320
AU6: Accuracy=0.8797, Precision=0.2199, Recall=0.2500, F1
Score=0.2340
AU9: Accuracy=0.8563, Precision=0.1427, Recall=0.1667, F1
Score=0.1538
AU12: Accuracy=0.8407, Precision=0.2102, Recall=0.2500, F1
Score=0.2284
AU17: Accuracy=0.9938, Precision=0.3313, Recall=0.3333, F1
Score=0.3323
AU20: Accuracy=0.9940, Precision=0.3313, Recall=0.3333, F1
Score=0.3323
AU25: Accuracy=0.0336, Precision=0.0067, Recall=0.2000, F1
Score=0.0130
AU26: Accuracy=0.2334, Precision=0.0583, Recall=0.2500, F1
```

```
Score=0.0946
Best Model Average Accuracy: 0.6205
Best Model Average Precision: 0.1635
Best Model Average Recall: 0.2379
Best Model Average F1 Score: 0.1726

/users/aayush/latest/lib/python3.10/site-
packages/keras/src/layers/rnn/rnn.py:204: 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__(**kwargs)
```

```
        best_model = Sequential([
            LSTM(best_params['lstm_units'], input_shape=(1, X_t
            Dropout(best_params['dropout_rate']),
            LSTM(best_params['lstm_units']),
            Dropout(best_params['dropout_rate']),
            Dense(y_filtered.shape[1], activation='sigmoid')
        ])

        best_model.compile(loss='binary_crossentropy', optimize
        best_model.fit(X_train, y_train, epochs=50, batch_size=

        # Predictions
        y_pred = best_model.predict(X_test)
        y_pred_rounded = np.round(y_pred)  # Round predictions

        # Calculate metrics for each AU
        accuracies = [accuracy_score(y_test[:, i], y_pred_round
        precisions = [precision_score(y_test[:, i], y_pred_roun
        recalls = [recall_score(y_test[:, i], y_pred_rounded[:,
        f1_scores = [f1_score(y_test[:, i], y_pred_rounded[:, i

        # Print metrics for each AU
        for i, (acc, prec, rec, f1) in enumerate(zip(accuracies
            print(f'AU{filtered_indices[i]+1}: Accuracy={acc:.4

        # Calculate and print average metrics
        average_accuracy = np.mean(accuracies)
        average_precision = np.mean(precisions)
        average_recall = np.mean(recalls)
        average_f1_score = np.mean(f1_scores)

        print(f'Best Model Average Accuracy: {average_accuracy:
        print(f'Best Model Average Precision: {average_precisio
        print(f'Best Model Average Recall: {average_recall:.4f}
        print(f'Best Model Average F1 Score: {average_f1_score:
```

```
Epoch 1/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 26s 15ms/step - accuracy: 0.1797
- loss: -3.2435 - val_accuracy: 0.1814 - val_loss: -11.8997
Epoch 2/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 20s 15ms/step - accuracy: 0.1818
- loss: -14.4171 - val_accuracy: 0.1814 - val_loss: -21.8251
Epoch 3/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1830 -
loss: -23.9206 - val_accuracy: 0.1814 - val_loss: -31.7045
Epoch 4/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1827 -
loss: -34.0440 - val_accuracy: 0.1814 - val_loss: -41.5334
Epoch 5/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1835 -
loss: -44.0277 - val_accuracy: 0.1814 - val_loss: -51.3828
Epoch 6/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1848 -
loss: -53.2217 - val_accuracy: 0.1814 - val_loss: -61.2267
Epoch 7/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1844 -
loss: -63.0427 - val_accuracy: 0.1814 - val_loss: -71.0619
Epoch 8/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1836 -
loss: -73.8753 - val_accuracy: 0.1814 - val_loss: -80.8650
Epoch 9/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1827 -
loss: -82.0001 - val_accuracy: 0.1814 - val_loss: -90.7328
Epoch 10/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1824 -
loss: -92.9158 - val_accuracy: 0.1814 - val_loss: -100.5478
Epoch 11/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1834 -
loss: -102.0853 - val_accuracy: 0.1814 - val_loss: -110.3893
Epoch 12/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1836 -
loss: -112.3596 - val_accuracy: 0.1814 - val_loss: -120.2159
Epoch 13/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1828 -
loss: -121.3852 - val_accuracy: 0.1814 - val_loss: -130.0584
Epoch 14/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1828 -
loss: -132.7272 - val_accuracy: 0.1814 - val_loss: -139.8809
Epoch 15/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1829 -
loss: -143.7853 - val_accuracy: 0.1814 - val_loss: -149.7126
Epoch 16/50
```

```
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1834 -
loss: -152.4643 - val_accuracy: 0.1814 - val_loss: -159.5703
Epoch 17/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1820 -
loss: -160.4477 - val_accuracy: 0.1814 - val_loss: -169.4258
Epoch 18/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1834 -
loss: -169.5609 - val_accuracy: 0.1814 - val_loss: -179.3057
Epoch 19/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1826 -
loss: -181.2196 - val_accuracy: 0.1814 - val_loss: -189.1246
Epoch 20/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1820 -
loss: -192.0972 - val_accuracy: 0.1814 - val_loss: -198.9630
Epoch 21/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1855 -
loss: -201.6947 - val_accuracy: 0.1814 - val_loss: -208.8063
Epoch 22/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1826 -
loss: -210.3319 - val_accuracy: 0.1814 - val_loss: -218.6736
Epoch 23/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1826 -
loss: -220.1535 - val_accuracy: 0.1814 - val_loss: -228.5101
Epoch 24/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1826 -
loss: -228.6290 - val_accuracy: 0.1814 - val_loss: -238.3201
Epoch 25/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1827 -
loss: -240.0834 - val_accuracy: 0.1814 - val_loss: -248.1510
Epoch 26/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1834 -
loss: -250.9578 - val_accuracy: 0.1814 - val_loss: -257.9806
Epoch 27/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1821 -
loss: -258.4809 - val_accuracy: 0.1814 - val_loss: -267.8338
Epoch 28/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1841 -
loss: -270.2542 - val_accuracy: 0.1814 - val_loss: -277.6533
Epoch 29/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1831 -
loss: -277.1691 - val_accuracy: 0.1814 - val_loss: -287.5250
Epoch 30/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1832 -
loss: -287.3944 - val_accuracy: 0.1814 - val_loss: -297.3580
Epoch 31/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1814 -
loss: -295.0448 - val_accuracy: 0.1814 - val_loss: -307.1913
Epoch 32/50
```

```
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1850 -
loss: -313.8519 - val_accuracy: 0.1814 - val_loss: -316.9832
Epoch 33/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1818 -
loss: -312.3350 - val_accuracy: 0.1814 - val_loss: -326.8353
Epoch 34/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1827 -
loss: -331.4785 - val_accuracy: 0.1814 - val_loss: -336.5907
Epoch 35/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1845 -
loss: -336.3485 - val_accuracy: 0.1814 - val_loss: -346.4266
Epoch 36/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1836 -
loss: -350.1833 - val_accuracy: 0.1814 - val_loss: -356.2438
Epoch 37/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1812 -
loss: -356.2502 - val_accuracy: 0.1814 - val_loss: -366.0988
Epoch 38/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1831 -
loss: -367.3306 - val_accuracy: 0.1814 - val_loss: -375.9098
Epoch 39/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1842 -
loss: -376.1510 - val_accuracy: 0.1814 - val_loss: -385.7501
Epoch 40/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1830 -
loss: -383.0804 - val_accuracy: 0.1814 - val_loss: -395.5809
Epoch 41/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1831 -
loss: -396.0658 - val_accuracy: 0.1814 - val_loss: -405.3993
Epoch 42/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1818 -
loss: -408.5817 - val_accuracy: 0.1814 - val_loss: -415.2400
Epoch 43/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1814 -
loss: -414.5608 - val_accuracy: 0.1814 - val_loss: -425.1046
Epoch 44/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1825 -
loss: -431.9555 - val_accuracy: 0.1814 - val_loss: -434.9210
Epoch 45/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.1828 -
loss: -431.5617 - val_accuracy: 0.1814 - val_loss: -444.7786
Epoch 46/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1823 -
loss: -447.8996 - val_accuracy: 0.1814 - val_loss: -454.6001
Epoch 47/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.1823 -
loss: -459.7343 - val_accuracy: 0.1814 - val_loss: -464.4043
Epoch 48/50
```

```
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1859 -
loss: -470.9236 - val_accuracy: 0.1814 - val_loss: -474.2212
Epoch 49/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1813 -
loss: -474.4066 - val_accuracy: 0.1814 - val_loss: -484.0891
Epoch 50/50
1378/1378 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1851 -
loss: -478.5876 - val_accuracy: 0.1814 - val_loss: -493.9116
1181/1181 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
AU1: Accuracy=0.0210, Precision=0.0035, Recall=0.1667, F1
Score=0.0068
AU2: Accuracy=0.9557, Precision=0.1593, Recall=0.1667, F1
Score=0.1629
AU4: Accuracy=0.0248, Precision=0.0041, Recall=0.1667, F1
Score=0.0081
AU5: Accuracy=0.9922, Precision=0.3307, Recall=0.3333, F1
Score=0.3320
AU6: Accuracy=0.8797, Precision=0.2199, Recall=0.2500, F1
Score=0.2340
AU9: Accuracy=0.8563, Precision=0.1427, Recall=0.1667, F1
Score=0.1538
AU12: Accuracy=0.8407, Precision=0.2102, Recall=0.2500, F1
Score=0.2284
AU17: Accuracy=0.9938, Precision=0.3313, Recall=0.3333, F1
Score=0.3323
AU20: Accuracy=0.9940, Precision=0.3313, Recall=0.3333, F1
Score=0.3323
AU25: Accuracy=0.0336, Precision=0.0067, Recall=0.2000, F1
Score=0.0130
AU26: Accuracy=0.2334, Precision=0.0583, Recall=0.2500, F1
Score=0.0946
Best Model Average Accuracy: 0.6205
Best Model Average Precision: 0.1635
Best Model Average Recall: 0.2379
Best Model Average F1 Score: 0.1726
```