REAL-TIMESENTIMENTANAYLSIS

AIML Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

BY

<K.VURVIK><1602-22-737-146>

<PREETHAM><1602-22-737-188>

Under the guidance of Mrs. Satyadevi

Department of InformationTechnology

Vasavi Collegeof Engineering (Autonomous)

(Affiliated toOsmania University)

Ibrahimbagh, Hyderabad-31 2020

DECLARATION BY THE CANDIDATE

We, **K.Vurvik** and **Preetham**, bearing hall ticket numbers, **1602-22-737-146** and **1602-22-737-188**, hereby declare that the project report entitled **" real-time sentimentanalysis"** Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of Bachelor of Engineering in Information Technology

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

K.Vurvik<1602-22-737-146>

Preetham<1602-22-737-188>

(Faculty In-Charge) **(Head,DeptOfIt)**

AIM AND PRIORITY OF THE PROJECT:

The aim of this project is to develop a real-time sentiment analysis system that processes the FER-2013 dataset to identify and classify facial emotions accurately. The focus is on analyzing raw data, preprocessing it effectively, and building a foundation for real-time emotion recognition.

INTRODUCTION:

Sentiment analysis plays a vital role in human-computer interaction, providing systems with the ability to understand and respond to human emotions. The FER-2013 dataset is widely used for facial expression recognition tasks, offering labeled data for emotions such as happiness, sadness, anger, and more. This project explores the preprocessing steps required to transform raw FER-2013 data into a usable format for real-time sentiment analysis applications.

By analyzing and cleaning the dataset, we aim to lay the groundwork for training a machine learning model capable of classifying emotions accurately.

OBJECTIVES:

- To preprocess the FER-2013 dataset by handling missing values, removing unnecessary columns, and structuring the data for machine learning applications.
- To ensure the data is suitable for real-time sentiment analysis by optimizing size and format.
- To identify key challenges in emotion classification and prepare the dataset for further training using supervised learning techniques.

- To evaluate the effectiveness of the preprocessing steps through initial exploratory analysis.

CONCLUSION:

The analysis and preprocessing of the FER-2013 dataset form the cornerstone of this project. By addressing data quality issues and structuring the dataset effectively, we ensure a solid foundation for training a machine learning model for real-time sentiment analysis. This step is critical for achieving robust performance in classifying facial emotions, paving the way for applications in areas such as mental health monitoring, customer service, and human-computer interaction.

Architecture and Technology Used:

The project is designed using a modular architecture for real-time sentiment analysis, comprising the following components:

1. **Data Processing Layer**:
    - Cleans and preprocesses the FER-2013 dataset for efficient emotion classification.
    - Features like pixel intensities are extracted from the dataset for analysis.
2. **Model Training Layer**:
    - A machine learning model (e.g., logistic regression, SVM, or neural networks) or a pre-trained deep learning model (e.g., CNNs) is used to classify emotions based on facial features.
3. **Real-Time Processing Layer**:
    - Captures real-time inputs from camera feeds or pre-saved images.
    - Converts these inputs into a format consistent with the model's requirements.
4. **Prediction and Visualization Layer**:
    - Processes model outputs to classify emotions.
    - Displays results in real-time with appropriate visualizations or text outputs.

Technologies Used:

- **Programming Language**: Python

- **Data Processing**: Pandas, NumPy
- **Machine Learning/Deep Learning**: TensorFlow or Scikit-Learn
- **Visualization**: Matplotlib, Seaborn
- **Real-Time Input**: OpenCV for capturing and processing real-time images.

Additional Dependencies:

To execute the project, the following dependencies are used:

1. **Core Libraries**:
    - pandas: For data manipulation and cleaning.
    - numpy: For numerical computations.
    - matplotlib: For data visualization.
    - seaborn: For enhanced visualizations.
2. **Machine Learning/Deep Learning**:
    - tensorflow or keras: For building and training deep learning models.
    - scikit-learn: For basic machine learning models and metrics.
3. **Real-Time Input**:
    - opencv-python: For real-time image and video feed processing.
4. **Dataset Handling**:
    - FER-2013 dataset (fer2013.csv).

REAL-TIMESENTIMENT ANALYSIS(FER-2013) :

```
import pandas as pd
import numpy as np
df=pd.read_csv('fer2013.csv')
df
```

```
       emotion                                             pixels
Usage
0            0  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
Training
1            0  151 150 147 155 148 133 111 140 170 174 182 15...
Training
2            2  231 212 156 164 174 138 161 173 182 200 106 38...
Training
3            4  24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
Training
4            6  4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...
Training
...        ...                                                ...
...
35882        6  50 36 17 22 23 29 33 39 34 37 37 37 39 43 48 5...
PrivateTest
35883        3  178 174 172 173 181 188 191 194 196 199 200 20...
PrivateTest
35884        0  17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...
PrivateTest
35885        3  30 28 28 29 31 30 42 68 79 81 77 67 67 71 63 6...
PrivateTest
35886        2  19 13 14 12 13 16 21 33 50 57 71 84 97 108 122...
PrivateTest

[35887 rows x 3 columns]
```

```
df.isna().sum()
```

```
emotion    0
pixels     0
Usage      0
dtype: int64
```

```
df.drop('Usage',axis=1,inplace=True)
```

```
df
```

```
       emotion                                             pixels
0            0  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1            0  151 150 147 155 148 133 111 140 170 174 182 15...
2            2  231 212 156 164 174 138 161 173 182 200 106 38...
3            4  24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4            6  4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...
```

```
...         ...                                                ...
35882         6   50 36 17 22 23 29 33 39 34 37 37 37 39 43 48 5...
35883         3   178 174 172 173 181 188 191 194 196 199 200 20...
35884         0   17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...
35885         3   30 28 28 29 31 30 42 68 79 81 77 67 67 71 63 6...
35886         2   19 13 14 12 13 16 21 33 50 57 71 84 97 108 122...

[35887 rows x 2 columns]
```

```
df.head()
```

```
   emotion                                              pixels
0        0  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1        0  151 150 147 155 148 133 111 140 170 174 182 15...
2        2  231 212 156 164 174 138 161 173 182 200 106 38...
3        4  24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4        6  4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...
```

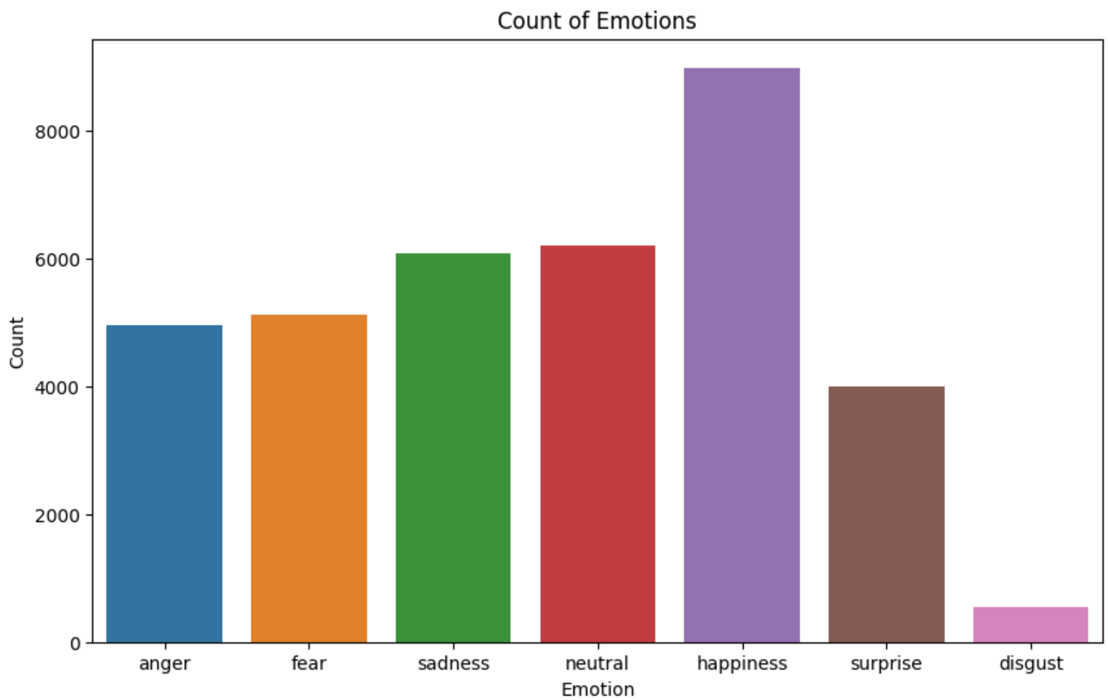
ADDING EMOTION_MAPPING TO THE FER2013 DATA

```
emotion_mapping = {
    0: 'anger',
    1: 'disgust',
    2: 'fear',
    3: 'happiness',
    4: 'sadness',
    5: 'surprise',
    6: 'neutral'
}
df['emotion_label'] = df['emotion'].map(emotion_mapping)
df
```

```
       emotion                                              pixels  \
0            0  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1            0  151 150 147 155 148 133 111 140 170 174 182 15...
2            2  231 212 156 164 174 138 161 173 182 200 106 38...
3            4  24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4            6  4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...
...        ...                                                ...
35882        6  50 36 17 22 23 29 33 39 34 37 37 37 39 43 48 5...
35883        3  178 174 172 173 181 188 191 194 196 199 200 20...
35884        0  17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...
35885        3  30 28 28 29 31 30 42 68 79 81 77 67 67 71 63 6...
35886        2  19 13 14 12 13 16 21 33 50 57 71 84 97 108 122...

      emotion_label
0             anger
1             anger
2              fear
3           sadness
4           neutral
```

```
...               ...
35882         neutral
35883       happiness
35884           anger
35885       happiness
35886            fear

[35887 rows x 3 columns]
```

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='emotion_label')
plt.title('Count of Emotions')
plt.xlabel('Emotion')
plt.ylabel('Count')
plt.show()
```



RESHAPING THE DATA,AND APPLYING MIN MAX SCALER

```
from sklearn.preprocessing import MinMaxScaler
pixels = df['pixels'].apply(lambda x: np.array(x.split(), dtype='int32'))
images = np.vstack(pixels).reshape(-1, 48, 48)
scaler = MinMaxScaler()
images_scaled = scaler.fit_transform(images.reshape(-1, 2304)).reshape(-1,
48, 48)
fig, axes = plt.subplots(1, 5, figsize=(15, 3))
for i, ax in enumerate(axes):
    ax.imshow(images_scaled[i], cmap='gray')
```

```
    ax.set_title(f'Image {i+1}')
    ax.axis('off')
plt.show()
```


```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['emotion_label'])
```

```
import numpy as np
x = np.array(images_scaled).astype('float32')
y = np.array(y).astype('float32')
```

```
y
```

```
array([0., 0., 2., ..., 0., 3., 2.], dtype=float32)
```

```
from sklearn.model_selection import
train_test_split,cross_val_score,GridSearchCV
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

```
((28709, 48, 48), (28709,), (7178, 48, 48), (7178,))
```

```
X_train
```

```
array([[[0.01568628, 0.01568628, 0.01568628, ..., 0.11372549,
         0.1254902 , 0.12156863],
        [0.03921569, 0.03529412, 0.02352941, ..., 0.4862745 ,
         0.5294118 , 0.5294118 ],
        [0.03529412, 0.03529412, 0.02745098, ..., 0.46666667,
         0.5019608 , 0.5137255 ],
        ...,
        [0.45882353, 0.46666667, 0.49803922, ..., 0.02745098,
         0.01568628, 0.01568628],
        [0.05882353, 0.04705882, 0.08235294, ..., 0.01176471,
         0.00784314, 0.01176471],
        [0.00392157, 0.00392157, 0.        , ..., 0.01176471,
         0.01176471, 0.01176471]],

       [[0.11764706, 0.1254902 , 0.16078432, ..., 0.7137255 ,
         0.63529414, 0.6117647 ],
        [0.09411765, 0.14509805, 0.12156863, ..., 0.70980394,
         0.627451  , 0.5921569 ],
```

```
 [0.11372549, 0.12156863, 0.08235294, ..., 0.73333335,
  0.627451  , 0.58431375],
 ...,
 [0.6745098 , 0.68235296, 0.4745098 , ..., 0.49411765,
  0.43529412, 0.4627451 ],
 [0.6666667 , 0.6666667 , 0.39215687, ..., 0.46666667,
  0.46666667, 0.44705883],
 [0.68235296, 0.6392157 , 0.36862746, ..., 0.42352942,
  0.45490196, 0.4627451 ]],

[[0.9843137 , 0.9607843 , 0.8117647 , ..., 0.30588236,
  0.3372549 , 0.36862746],
 [0.9764706 , 0.83137256, 0.5529412 , ..., 0.29803923,
  0.32941177, 0.35686275],
 [0.8901961 , 0.62352943, 0.40392157, ..., 0.23529412,
  0.29411766, 0.32941177],
 ...,
 [0.6431373 , 0.64705884, 0.65882355, ..., 0.32156864,
  0.13333334, 0.01568628],
 [0.6745098 , 0.63529414, 0.6431373 , ..., 0.2901961 ,
  0.05098039, 0.02745098],
 [0.67058825, 0.6313726 , 0.6313726 , ..., 0.24313726,
  0.00784314, 0.04313726]],

...,

[[0.38039216, 0.3647059 , 0.23921569, ..., 0.5803922 ,
  0.5176471 , 0.5058824 ],
 [0.41568628, 0.34901962, 0.25882354, ..., 0.627451  ,
  0.5254902 , 0.5176471 ],
 [0.30980393, 0.34509805, 0.27058825, ..., 0.654902  ,
  0.57254905, 0.5137255 ],
 ...,
 [0.09019608, 0.01960784, 0.01960784, ..., 0.2627451 ,
  0.42352942, 0.7294118 ],
 [0.10588235, 0.03921569, 0.01568628, ..., 0.28235295,
  0.4       , 0.72156864],
 [0.05490196, 0.05882353, 0.01568628, ..., 0.32156864,
  0.41568628, 0.63529414]],

[[0.99607843, 0.89411765, 0.84705883, ..., 0.6862745 ,
  0.6392157 , 0.62352943],
 [0.9607843 , 0.84313726, 0.8392157 , ..., 0.6666667 ,
  0.6509804 , 0.6       ],
 [0.90588236, 0.8156863 , 0.81960785, ..., 0.6392157 ,
  0.6784314 , 0.5803922 ],
 ...,
 [0.8862745 , 0.8235294 , 0.8509804 , ..., 0.5764706 ,
  0.75686276, 0.8       ],
```

```
         [0.8392157 , 0.8392157 , 0.827451  , ..., 0.69803923,
          0.7607843 , 0.80784315],
         [0.8627451 , 0.8235294 , 0.85490197, ..., 0.74509805,
          0.85490197, 0.8784314 ]],

        [[0.46666667, 0.39607844, 0.07058824, ..., 0.00392157,
          0.00392157, 0.00784314],
         [0.48235294, 0.21176471, 0.        , ..., 0.00784314,
          0.01960784, 0.00784314],
         [0.36862746, 0.05882353, 0.        , ..., 0.        ,
          0.01960784, 0.00392157],
         ...,
         [0.        , 0.        , 0.        , ..., 0.        ,
          0.        , 0.        ],
         [0.        , 0.        , 0.        , ..., 0.        ,
          0.        , 0.        ],
         [0.        , 0.        , 0.        , ..., 0.        ,
          0.        , 0.        ]]], dtype=float32)
```

OUTLIER DETECTION USING QUANTILE AND Z-SCORE

```
df.describe()
```

```
            emotion
count  35887.000000
mean       3.323265
std        1.873819
min        0.000000
25%        2.000000
50%        3.000000
75%        5.000000
max        6.000000
```

```
df[df.emotion>df.emotion.quantile(0.5990)]
```

```
       emotion                                             pixels  \
4            6  4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...
11           6  39 75 78 58 58 45 49 48 103 156 81 45 41 38 49...
12           6  219 213 206 202 209 217 216 215 219 218 223 23...
13           6  148 144 130 129 119 122 129 131 139 153 140 12...
15           5  107 107 109 109 109 109 110 101 123 140 144 14...
...        ...                                                ...
35874        5  43 43 51 73 94 97 102 95 99 107 126 144 154 17...
35875        5  248 251 239 144 102 95 82 77 91 138 153 145 14...
35876        6  29 29 27 31 49 56 29 19 22 20 34 43 55 71 85 9...
35877        6  139 143 145 154 159 168 176 181 190 191 195 19...
35882        6  50 36 17 22 23 29 33 39 34 37 37 37 39 43 48 5...

      emotion_label
4           neutral
11          neutral
```

```
12          neutral
13          neutral
15         surprise
...             ...
35874      surprise
35875      surprise
35876       neutral
35877       neutral
35882       neutral

[10200 rows x 3 columns]
```

```
df[df.emotion<df.emotion.quantile(0.2)]
```

```
       emotion                                             pixels  \
0            0  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1            0  151 150 147 155 148 133 111 140 170 174 182 15...
10           0  30 24 21 23 25 25 49 67 84 103 120 125 130 139...
22           0  123 125 124 142 209 226 234 236 231 232 235 22...
23           0  8 9 14 21 26 32 37 46 52 62 72 70 71 73 76 83 ...
...        ...                                                ...
35845        0  149 141 141 137 143 157 166 123 217 230 220 21...
35849        0  90 82 42 36 33 45 81 112 122 139 155 149 152 1...
35854        0  139 141 141 138 152 164 176 160 137 118 106 95...
35881        0  181 177 176 156 178 144 136 132 122 107 131 16...
35884        0  17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...

      emotion_label
0             anger
1             anger
10            anger
22            anger
23            anger
...             ...
35845         anger
35849         anger
35854         anger
35881         anger
35884         anger

[5500 rows x 3 columns]
```

```
upper_limit=df.emotion.mean()+3*df.emotion.std()
upper_limit
```

```
8.944720969849602
```

```
lower_limit=df.emotion.mean()-3*df.emotion.std()
lower_limit
```

```
-2.298191585950139
```

```
df=df[(df.emotion<upper_limit) & (df.emotion>lower_limit)]
```

```
print(df.describe())
```

```
            emotion
count  35887.000000
mean       3.323265
std        1.873819
min        0.000000
25%        2.000000
50%        3.000000
75%        5.000000
max        6.000000
```

```
from  sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn .model_selection import RandomizedSearchCV
```

```
print(X_train.shape)
```

```
(28709, 48, 48)
```

```
svc = SVC()

# Parameter grid
param_grid_svc = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 0.001, 0.01]
}

# Set up RandomizedSearchCV
random_search_svc = RandomizedSearchCV(
    estimator=svc,
    param_distributions=param_grid_svc,
    n_iter=20,
    scoring='accuracy',
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=1  # Use a single worker
)
# Use only a portion of the training data
X_train_small = X_train[:1000]
y_train_small = y_train[:1000]
X_train_small1=X_train_small.reshape(X_train_small.shape[0], -1)  # Reshape
to (28709, 2304)
```

```
random_search_svc.fit(X_train_small1, y_train_small)

# Best parameters and accuracy
print("Best Parameters for SVC:", random_search_svc.best_params_)
print("Best Score:", random_search_svc.best_score_)
```

```
c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\model_selection\_search.py:296: UserWarning: The total space
of parameters 18 is smaller than n_iter=20. Running 18 iterations. For
exhaustive searches, use GridSearchCV.
  UserWarning,

Fitting 5 folds for each of 18 candidates, totalling 90 fits
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=
0.6s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=
0.6s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=
0.5s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=
0.5s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=
0.5s
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time=
0.6s
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time=
0.7s
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time=
0.7s
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time=
0.6s
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time=
0.6s
[CV] END ..................C=0.1, gamma=0.001, kernel=linear; total time=
0.5s
[CV] END ..................C=0.1, gamma=0.001, kernel=linear; total time=
0.5s
[CV] END ..................C=0.1, gamma=0.001, kernel=linear; total time=
0.5s
[CV] END ..................C=0.1, gamma=0.001, kernel=linear; total time=
0.4s
[CV] END ..................C=0.1, gamma=0.001, kernel=linear; total time=
0.5s
[CV] END .....................C=0.1, gamma=0.001, kernel=rbf; total time=
0.6s
[CV] END .....................C=0.1, gamma=0.001, kernel=rbf; total time=
0.6s
[CV] END .....................C=0.1, gamma=0.001, kernel=rbf; total time=
0.6s
[CV] END .....................C=0.1, gamma=0.001, kernel=rbf; total time=
```

```
0.6s
[CV] END .....................C=0.1, gamma=0.001, kernel=rbf; total time=
0.6s
[CV] END ...................C=0.1, gamma=0.01, kernel=linear; total time=
0.5s
[CV] END ...................C=0.1, gamma=0.01, kernel=linear; total time=
0.5s
[CV] END ...................C=0.1, gamma=0.01, kernel=linear; total time=
0.5s
[CV] END ...................C=0.1, gamma=0.01, kernel=linear; total time=
0.5s
[CV] END ...................C=0.1, gamma=0.01, kernel=linear; total time=
0.6s
[CV] END ......................C=0.1, gamma=0.01, kernel=rbf; total time=
0.8s
[CV] END ......................C=0.1, gamma=0.01, kernel=rbf; total time=
0.8s
[CV] END ......................C=0.1, gamma=0.01, kernel=rbf; total time=
0.9s
[CV] END ......................C=0.1, gamma=0.01, kernel=rbf; total time=
0.8s
[CV] END ......................C=0.1, gamma=0.01, kernel=rbf; total time=
0.9s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=
0.8s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=
0.7s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=
0.7s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=
0.7s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=
0.7s
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time=
0.8s
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time=
0.8s
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time=
0.8s
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time=
0.8s
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time=
0.9s
[CV] END ....................C=1, gamma=0.001, kernel=linear; total time=
0.9s
[CV] END ....................C=1, gamma=0.001, kernel=linear; total time=
0.8s
[CV] END ....................C=1, gamma=0.001, kernel=linear; total time=
0.8s
[CV] END ....................C=1, gamma=0.001, kernel=linear; total time=
```

```
0.8s
[CV] END ....................C=1, gamma=0.001, kernel=linear; total time=
0.7s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=
0.8s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=
0.8s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=
0.7s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=
0.8s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=
0.9s
[CV] END .....................C=1, gamma=0.01, kernel=linear; total time=
0.9s
[CV] END .....................C=1, gamma=0.01, kernel=linear; total time=
0.7s
[CV] END .....................C=1, gamma=0.01, kernel=linear; total time=
0.8s
[CV] END .....................C=1, gamma=0.01, kernel=linear; total time=
0.9s
[CV] END .....................C=1, gamma=0.01, kernel=linear; total time=
0.8s
[CV] END ........................C=1, gamma=0.01, kernel=rbf; total time=
0.8s
[CV] END ........................C=1, gamma=0.01, kernel=rbf; total time=
1.0s
[CV] END ........................C=1, gamma=0.01, kernel=rbf; total time=
1.0s
[CV] END ........................C=1, gamma=0.01, kernel=rbf; total time=
0.9s
[CV] END ........................C=1, gamma=0.01, kernel=rbf; total time=
1.0s
[CV] END ...................C=10, gamma=scale, kernel=linear; total time=
1.0s
[CV] END ...................C=10, gamma=scale, kernel=linear; total time=
0.9s
[CV] END ...................C=10, gamma=scale, kernel=linear; total time=
0.8s
[CV] END ...................C=10, gamma=scale, kernel=linear; total time=
0.7s
[CV] END ...................C=10, gamma=scale, kernel=linear; total time=
0.7s
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=
0.9s
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=
0.9s
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=
0.9s
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=
```

```
0.9s
[CV] END .......................C=10, gamma=scale, kernel=rbf; total time=
0.9s
[CV] END ...................C=10, gamma=0.001, kernel=linear; total time=
0.7s
[CV] END ...................C=10, gamma=0.001, kernel=linear; total time=
0.7s
[CV] END ...................C=10, gamma=0.001, kernel=linear; total time=
0.7s
[CV] END ...................C=10, gamma=0.001, kernel=linear; total time=
0.7s
[CV] END ...................C=10, gamma=0.001, kernel=linear; total time=
0.7s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=
0.7s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=
0.7s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=
0.8s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=
0.8s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=
0.9s
[CV] END ....................C=10, gamma=0.01, kernel=linear; total time=
0.9s
[CV] END ....................C=10, gamma=0.01, kernel=linear; total time=
0.9s
[CV] END ....................C=10, gamma=0.01, kernel=linear; total time=
1.0s
[CV] END ....................C=10, gamma=0.01, kernel=linear; total time=
1.4s
[CV] END ....................C=10, gamma=0.01, kernel=linear; total time=
1.8s
[CV] END .......................C=10, gamma=0.01, kernel=rbf; total time=
2.9s
[CV] END .......................C=10, gamma=0.01, kernel=rbf; total time=
3.0s
[CV] END .......................C=10, gamma=0.01, kernel=rbf; total time=
3.1s
[CV] END .......................C=10, gamma=0.01, kernel=rbf; total time=
2.9s
[CV] END .......................C=10, gamma=0.01, kernel=rbf; total time=
3.0s
Best Parameters for SVC: {'kernel': 'rbf', 'gamma': 0.001, 'C': 10}
Best Score: 0.331
```

```
gbc = GradientBoostingClassifier()

# Parameter grid for GradientBoostingClassifier
param_grid_gbc = {
```

```
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.5],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'subsample': [0.8, 1.0]
}

# Set up RandomizedSearchCV
random_search_gbc = RandomizedSearchCV(
    estimator=gbc,
    param_distributions=param_grid_gbc,
    n_iter=20,
    scoring='accuracy',
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=1  # Use a single worker
)

# Use only a portion of the training data
X_train_small = X_train[:1000]
y_train_small = y_train[:1000]
X_train_small1 = X_train_small.reshape(X_train_small.shape[0], -1)  # Reshape
to (1000, 2304)

# Fit the RandomizedSearchCV with the data
random_search_gbc.fit(X_train_small1, y_train_small)

# Best parameters and accuracy
print("Best Parameters for GradientBoosting:",
random_search_gbc.best_params_)
print("Best Score for GradientBoosting:", random_search_gbc.best_score_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] END learning_rate=0.5, max_depth=7, min_samples_split=10,
n_estimators=100, subsample=0.8; total time=13.1min
[CV] END learning_rate=0.5, max_depth=7, min_samples_split=10,
n_estimators=100, subsample=0.8; total time=12.2min
[CV] END learning_rate=0.5, max_depth=7, min_samples_split=10,
n_estimators=100, subsample=0.8; total time=13.2min
[CV] END learning_rate=0.5, max_depth=7, min_samples_split=10,
n_estimators=100, subsample=0.8; total time=12.5min

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_23792\2329957545.py in <module>
     28
     29 # Fit the RandomizedSearchCV with the data
---> 30 random_search_gbc.fit(X_train_small1, y_train_small)
     31
```

```
     32 # Best parameters and accuracy

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\model_selection\_search.py in fit(self, X, y, groups,
**fit_params)
    889                 return results
    890
--> 891             self._run_search(evaluate_candidates)
    892
    893             # multimetric is determined here because in the case of a
callable

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\model_selection\_search.py in _run_search(self,
evaluate_candidates)
   1766         evaluate_candidates(
   1767             ParameterSampler(
-> 1768                 self.param_distributions, self.n_iter,
random_state=self.random_state
   1769             )
   1770         )

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\model_selection\_search.py in
evaluate_candidates(candidate_params, cv, more_results)
    849                     )
    850                     for (cand_idx, parameters), (split_idx, (train,
test)) in product(
--> 851                         enumerate(candidate_params),
enumerate(cv.split(X, y, groups))
    852                     )
    853                 )

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\joblib\parallel.py in __call__(self, iterable)
   1861             output = self._get_sequential_output(iterable)
   1862             next(output)
-> 1863             return output if self.return_generator else list(output)
   1864
   1865         # Let's create an ID that uniquely identifies the current
call. If the

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\joblib\parallel.py in _get_sequential_output(self, iterable)
   1790                 self.n_dispatched_batches += 1
   1791                 self.n_dispatched_tasks += 1
-> 1792                 res = func(*args, **kwargs)
   1793                 self.n_completed_tasks += 1
   1794                 self.print_progress()
```

```
c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\utils\fixes.py in __call__(self, *args, **kwargs)
    214     def __call__(self, *args, **kwargs):
    215         with config_context(**self.config):
--> 216             return self.function(*args, **kwargs)
    217 
    218 

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\model_selection\_validation.py in _fit_and_score(estimator,
X, y, scorer, train, test, verbose, parameters, fit_params,
return_train_score, return_parameters, return_n_test_samples, return_times,
return_estimator, split_progress, candidate_progress, error_score)
    678             estimator.fit(X_train, **fit_params)
    679         else:
--> 680             estimator.fit(X_train, y_train, **fit_params)
    681 
    682     except Exception:

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\ensemble\_gb.py in fit(self, X, y, sample_weight, monitor)
    594             sample_weight_val,
    595             begin_at_stage,
--> 596             monitor,
    597         )
    598 

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\ensemble\_gb.py in _fit_stages(self, X, y, raw_predictions,
sample_weight, random_state, X_val, y_val, sample_weight_val, begin_at_stage,
monitor)
    670                 random_state,
    671                 X_csc,
--> 672                 X_csr,
    673             )
    674 

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\ensemble\_gb.py in _fit_stage(self, i, X, y,
raw_predictions, sample_weight, sample_mask, random_state, X_csc, X_csr)
    244 
    245             X = X_csr if X_csr is not None else X
--> 246             tree.fit(X, residual, sample_weight=sample_weight,
check_input=False)
    247 
    248             # update tree leaves

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
```

```
packages\sklearn\tree\_classes.py in fit(self, X, y, sample_weight,
check_input, X_idx_sorted)
   1318                 sample_weight=sample_weight,
   1319                 check_input=check_input,
-> 1320                 X_idx_sorted=X_idx_sorted,
   1321             )
   1322             return self

c:\Users\91739\AppData\Local\Programs\Python\Python37\lib\site-
packages\sklearn\tree\_classes.py in fit(self, X, y, sample_weight,
check_input, X_idx_sorted)
    418                 )
    419
--> 420             builder.build(self.tree_, X, y, sample_weight)
    421
    422             if self.n_outputs_ == 1 and is_classifier(self):

KeyboardInterrupt:
```

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam

# Model architecture
model = Sequential([
    # First Convolutional Block
    Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Second Convolutional Block
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Third Convolutional Block
    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.4),

    # Fully Connected Layers
    Flatten(),
    Dense(128, activation='relu'),
    BatchNormalization(),
```

```python
    Dropout(0.5),
    Dense(7, activation='softmax')  # 7 classes for the emotions
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Model Summary
model.summary()

# Data reshaping and normalization
X_train = X_train[..., np.newaxis]  # Add channel dimension
X_test = X_test[..., np.newaxis]

# Training the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=25,
    batch_size=64,
    verbose=1
)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_15 (Conv2D)          (None, 46, 46, 32)        320       
                                                                 
 batch_normalization_20 (Bat  (None, 46, 46, 32)       128       
 chNormalization)                                                
                                                                 
 max_pooling2d_15 (MaxPoolin  (None, 23, 23, 32)       0         
 g2D)                                                            
                                                                 
 dropout_20 (Dropout)        (None, 23, 23, 32)        0         
                                                                 
 conv2d_16 (Conv2D)          (None, 21, 21, 64)        18496     
                                                                 
 batch_normalization_21 (Bat  (None, 21, 21, 64)       256       
 chNormalization)                                                
                                                                 
 max_pooling2d_16 (MaxPoolin  (None, 10, 10, 64)       0         
```

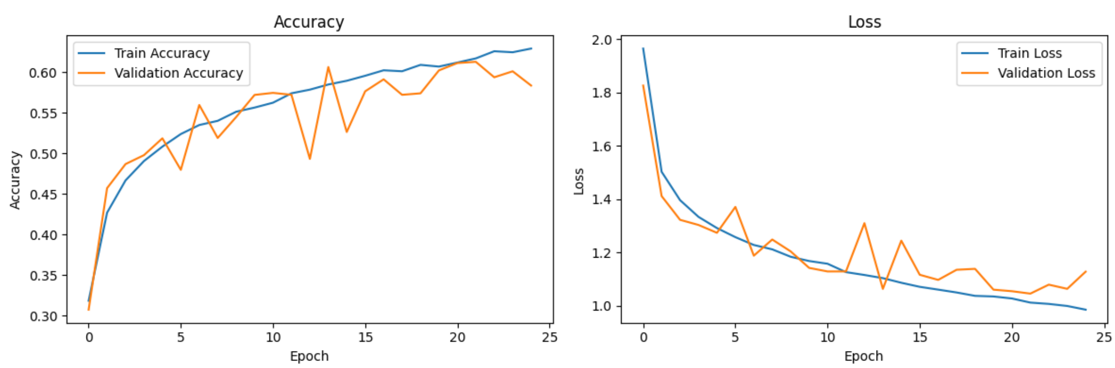
```
 g2D)

 dropout_21 (Dropout)        (None, 10, 10, 64)        0

 conv2d_17 (Conv2D)          (None, 8, 8, 128)         73856

 batch_normalization_22 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 max_pooling2d_17 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 dropout_22 (Dropout)        (None, 4, 4, 128)         0

 flatten_5 (Flatten)         (None, 2048)              0

 dense_10 (Dense)            (None, 128)               262272

 batch_normalization_23 (Bat  (None, 128)              512
 chNormalization)

 dropout_23 (Dropout)        (None, 128)               0

 dense_11 (Dense)            (None, 7)                 903

=================================================================
Total params: 357,255
Trainable params: 356,551
Non-trainable params: 704
_________________________________________________________________
Epoch 1/25
449/449 [==============================] - 155s 343ms/step - loss: 1.9642 -
accuracy: 0.3185 - val_loss: 1.8257 - val_accuracy: 0.3075
Epoch 2/25
449/449 [==============================] - 243s 542ms/step - loss: 1.5027 -
accuracy: 0.4270 - val_loss: 1.4116 - val_accuracy: 0.4572
Epoch 3/25
449/449 [==============================] - 351s 782ms/step - loss: 1.3964 -
accuracy: 0.4668 - val_loss: 1.3227 - val_accuracy: 0.4869
Epoch 4/25
449/449 [==============================] - 105s 234ms/step - loss: 1.3338 -
accuracy: 0.4906 - val_loss: 1.3034 - val_accuracy: 0.4979
Epoch 5/25
449/449 [==============================] - 94s 209ms/step - loss: 1.2918 -
accuracy: 0.5084 - val_loss: 1.2741 - val_accuracy: 0.5187
Epoch 6/25
449/449 [==============================] - 82s 182ms/step - loss: 1.2579 -
accuracy: 0.5239 - val_loss: 1.3711 - val_accuracy: 0.4798
Epoch 7/25
```

```
449/449 [==============================] - 85s 189ms/step - loss: 1.2284 -
accuracy: 0.5351 - val_loss: 1.1884 - val_accuracy: 0.5598
Epoch 8/25
449/449 [==============================] - 116s 259ms/step - loss: 1.2121 -
accuracy: 0.5402 - val_loss: 1.2489 - val_accuracy: 0.5191
Epoch 9/25
449/449 [==============================] - 105s 235ms/step - loss: 1.1843 -
accuracy: 0.5514 - val_loss: 1.2044 - val_accuracy: 0.5449
Epoch 10/25
449/449 [==============================] - 114s 253ms/step - loss: 1.1684 -
accuracy: 0.5566 - val_loss: 1.1427 - val_accuracy: 0.5722
Epoch 11/25
449/449 [==============================] - 255s 568ms/step - loss: 1.1581 -
accuracy: 0.5626 - val_loss: 1.1293 - val_accuracy: 0.5747
Epoch 12/25
449/449 [==============================] - 406s 905ms/step - loss: 1.1268 -
accuracy: 0.5741 - val_loss: 1.1295 - val_accuracy: 0.5723
Epoch 13/25
449/449 [==============================] - 195s 435ms/step - loss: 1.1161 -
accuracy: 0.5787 - val_loss: 1.3102 - val_accuracy: 0.4933
Epoch 14/25
449/449 [==============================] - 187s 417ms/step - loss: 1.1043 -
accuracy: 0.5850 - val_loss: 1.0639 - val_accuracy: 0.6066
Epoch 15/25
449/449 [==============================] - 106s 236ms/step - loss: 1.0869 -
accuracy: 0.5896 - val_loss: 1.2446 - val_accuracy: 0.5265
Epoch 16/25
449/449 [==============================] - 148s 330ms/step - loss: 1.0718 -
accuracy: 0.5958 - val_loss: 1.1169 - val_accuracy: 0.5766
Epoch 17/25
449/449 [==============================] - 114s 255ms/step - loss: 1.0612 -
accuracy: 0.6025 - val_loss: 1.0977 - val_accuracy: 0.5914
Epoch 18/25
449/449 [==============================] - 84s 187ms/step - loss: 1.0506 -
accuracy: 0.6013 - val_loss: 1.1357 - val_accuracy: 0.5723
Epoch 19/25
449/449 [==============================] - 102s 227ms/step - loss: 1.0379 -
accuracy: 0.6092 - val_loss: 1.1391 - val_accuracy: 0.5741
Epoch 20/25
449/449 [==============================] - 115s 257ms/step - loss: 1.0356 -
accuracy: 0.6071 - val_loss: 1.0610 - val_accuracy: 0.6024
Epoch 21/25
449/449 [==============================] - 86s 192ms/step - loss: 1.0280 -
accuracy: 0.6121 - val_loss: 1.0553 - val_accuracy: 0.6115
Epoch 22/25
449/449 [==============================] - 85s 189ms/step - loss: 1.0127 -
accuracy: 0.6172 - val_loss: 1.0463 - val_accuracy: 0.6130
Epoch 23/25
449/449 [==============================] - 86s 192ms/step - loss: 1.0077 -
accuracy: 0.6260 - val_loss: 1.0799 - val_accuracy: 0.5939
```

```
Epoch 24/25
449/449 [==============================] - 241s 537ms/step - loss: 1.0000 -
accuracy: 0.6248 - val_loss: 1.0642 - val_accuracy: 0.6013
Epoch 25/25
449/449 [==============================] - 84s 187ms/step - loss: 0.9861 -
accuracy: 0.6293 - val_loss: 1.1285 - val_accuracy: 0.5837
225/225 - 5s - loss: 1.1285 - accuracy: 0.5837 - 5s/epoch - 23ms/step
Test Accuracy: 0.58
```

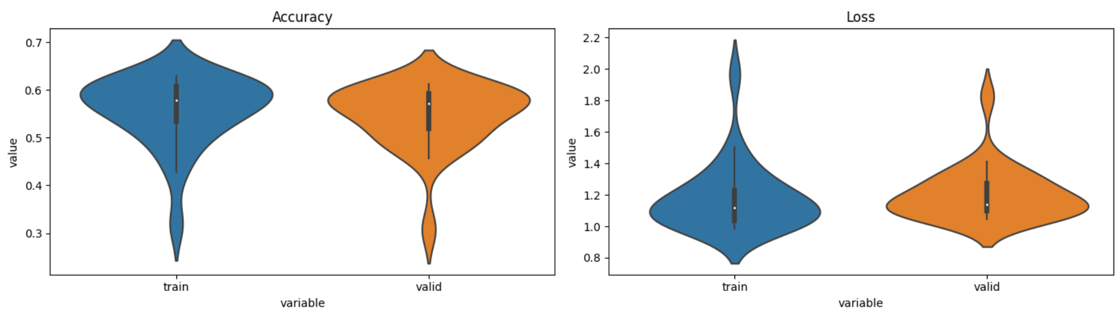
VISUALIZATION

```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy/loss
plt.figure(figsize=(12, 4))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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




```python
df_accu = pd.DataFrame({'train': history.history['accuracy'], 'valid':
history.history['val_accuracy']})
```

```
df_loss = pd.DataFrame({'train': history.history['loss'], 'valid':
history.history['val_loss']})

fig = plt.figure(0, (14, 4))
ax = plt.subplot(1, 2, 1)
sns.violinplot(x="variable", y="value", data=pd.melt(df_accu),
showfliers=False)
plt.title('Accuracy')
plt.tight_layout()

ax = plt.subplot(1, 2, 2)
sns.violinplot(x="variable", y="value", data=pd.melt(df_loss),
showfliers=False)
plt.title('Loss')
plt.tight_layout()

plt.savefig('performance_dist.png')
plt.show()
```




```
pip install scikit-plot

Requirement already satisfied: scikit-plot in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages
(0.3.7)
Requirement already satisfied: matplotlib>=1.4.0 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
scikit-plot) (3.5.3)
Requirement already satisfied: scikit-learn>=0.18 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
scikit-plot) (1.0.2)
Requirement already satisfied: scipy>=0.9 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
scikit-plot) (1.7.3)
Requirement already satisfied: joblib>=0.10 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
scikit-plot) (1.3.2)
Requirement already satisfied: cycler>=0.10 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
matplotlib>=1.4.0->scikit-plot) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
```

```
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
matplotlib>=1.4.0->scikit-plot) (4.38.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
matplotlib>=1.4.0->scikit-plot) (1.4.5)
Requirement already satisfied: numpy>=1.17 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
matplotlib>=1.4.0->scikit-plot) (1.21.6)
Requirement already satisfied: packaging>=20.0 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
matplotlib>=1.4.0->scikit-plot) (23.2)
Requirement already satisfied: pillow>=6.2.0 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
matplotlib>=1.4.0->scikit-plot) (9.5.0)
Requirement already satisfied: pyparsing>=2.2.1 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
matplotlib>=1.4.0->scikit-plot) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
matplotlib>=1.4.0->scikit-plot) (2.9.0.post0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
scikit-learn>=0.18->scikit-plot) (3.1.0)
Requirement already satisfied: typing-extensions in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
kiwisolver>=1.0.1->matplotlib>=1.4.0->scikit-plot) (4.7.1)
Requirement already satisfied: six>=1.5 in
c:\users\91739\appdata\local\programs\python\python37\lib\site-packages (from
python-dateutil>=2.7->matplotlib>=1.4.0->scikit-plot) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

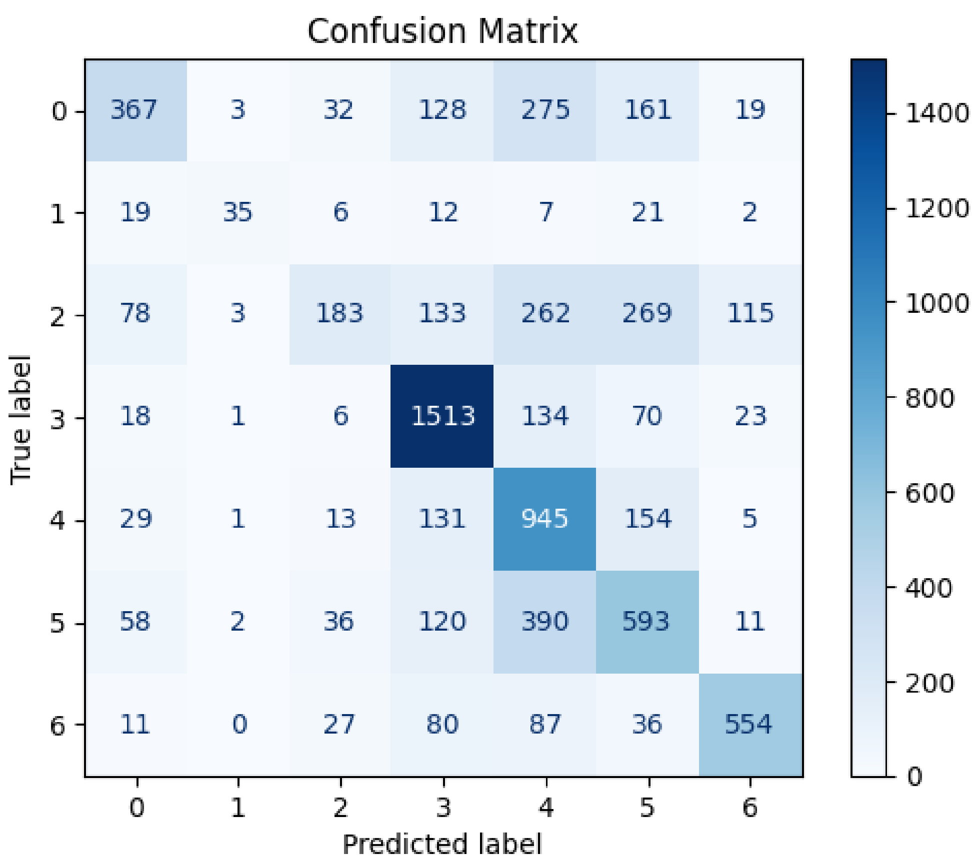
```python
print(f"y_test shape: {y_test.shape}")
```

```
y_test shape: (7178,)
```

```python
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report
import matplotlib.pyplot as plt

# Predict labels for the test set
yhat_valid = np.argmax(model.predict(X_test), axis=1)  # Ensure predictions
are label-encoded

# Compute confusion matrix
cm = confusion_matrix(y_test, yhat_valid)

# Display confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
```

```
plt.title("Confusion Matrix")
plt.show()

# Count misclassifications
total_wrong = np.sum(y_test != yhat_valid)
print(f"Total wrong validation predictions: {total_wrong}\n")

# Print classification report
print("Classification Report:\n")
print(classification_report(y_test, yhat_valid))
```

```
225/225 [==============================] - 3s 14ms/step
```




```
Total wrong validation predictions: 2988

Classification Report:

              precision    recall  f1-score   support

         0.0       0.63      0.37      0.47       985
         1.0       0.78      0.34      0.48       102
         2.0       0.60      0.18      0.27      1043
         3.0       0.71      0.86      0.78      1765
```

```
         4.0       0.45      0.74      0.56      1278
         5.0       0.45      0.49      0.47      1210
         6.0       0.76      0.70      0.73       795

    accuracy                           0.58      7178
   macro avg       0.63      0.52      0.54      7178
weighted avg       0.60      0.58      0.56      7178
```

```python
# Save the trained model
model.save("sentiment_analysis_model.h5")
print("Model saved successfully.")
```

```
Model saved successfully.
```

```python
from tensorflow.keras.models import load_model

# Load the saved model
deployed_model = load_model("sentiment_analysis_model.h5")
print("Model loaded for deployment.")
```

```
Model loaded for deployment.
```

0: 'anger', 1: 'disgust', 2: 'fear', 3: 'happiness', 4: 'sadness', 5: 'surprise', 6: 'neutral'

```python
import numpy as np

# Example function for prediction
def predict_emotion(image_path, model):
    from tensorflow.keras.preprocessing import image
    img = image.load_img(image_path, target_size=(48, 48),
color_mode='grayscale')
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Normalize

    predictions = model.predict(img_array)
    class_index = np.argmax(predictions)
    return class_index, predictions

# Predict emotion for a sample image
class_index, predictions = predict_emotion("Screenshot 2024-09-16
113843.png", deployed_model)
print(f"Predicted class index: {class_index}")
print(f"Class probabilities: {predictions}")
```

```
1/1 [==============================] - 1s 627ms/step
Predicted class index: 3
Class probabilities: [[1.0956363e-02 7.9436453e-05 7.5662122e-03 7.8519464e-
01 1.3591272e-01
  5.5589076e-02 4.7016051e-03]]
```