Presented by: ***Mendoza, Claire Antonette***

CS - FOPI01

Emotion Detector Accuracy Model

Data Gathering

- Data for the project was gathered from a reputable public source (such as Kaggle) that provided labeled images depicting various emotions. The Python library Pandas was used to load and verify the structure of the data, ensuring it was suitable for the modeling process. This initial step laid the groundwork for the next phases.

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/pyt
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dis
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/py
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from ker
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from ke
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from k
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/p
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-package
```

```
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (fr
```

```
import tensorflow as tf
print(tf.__version__)
```

```
2.17.0
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import os
import cv2
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import img_to_array, load_img
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from imblearn.over_sampling import SMOTE
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix , classification_report
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc, roc_auc_score

from IPython.display import clear_output
import warnings
warnings.filterwarnings('ignore')
```

```
np.random.seed(42)
tf.random.set_seed(42)

SEED = 12
IMG_HEIGHT = 48
IMG_WIDTH = 48
BATCH_SIZE = 64
EPOCHS = 30
FINE_TUNING_EPOCHS = 10
```

```
LR = 0.0001
NUM_CLASSES = 7
EARLY_STOPPING_CRITERIA = 3
CLASS_LABELS = ['Anger', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sadness', "Surprise"]
CLASS_LABELS_EMOJIS = ["😈", "🤢", "😱", "😊", "😐", "😔", "😳"]
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d msambare/fer2013 --force
```

```
Dataset URL: https://www.kaggle.com/datasets/msambare/fer2013
License(s): DbCL-1.0
Downloading fer2013.zip to /content
 91% 55.0M/60.3M [00:00<00:00, 81.7MB/s]
100% 60.3M/60.3M [00:00<00:00, 75.9MB/s]
```

```
!unzip /content/fer2013.zip -d /content/fer2013/
```

```
  inflating: /content/fer2013/train/surprise/Training_90031426.jpg
  inflating: /content/fer2013/train/surprise/Training_9003378.jpg
  inflating: /content/fer2013/train/surprise/Training_90128876.jpg
  inflating: /content/fer2013/train/surprise/Training_9013322.jpg
  inflating: /content/fer2013/train/surprise/Training_90166716.jpg
  inflating: /content/fer2013/train/surprise/Training_90167636.jpg
  inflating: /content/fer2013/train/surprise/Training_90187735.jpg
  inflating: /content/fer2013/train/surprise/Training_90190497.jpg
  inflating: /content/fer2013/train/surprise/Training_9023594.jpg
  inflating: /content/fer2013/train/surprise/Training_90241523.jpg
  inflating: /content/fer2013/train/surprise/Training_90255651.jpg
  inflating: /content/fer2013/train/surprise/Training_90260579.jpg
  inflating: /content/fer2013/train/surprise/Training_90340109.jpg
  inflating: /content/fer2013/train/surprise/Training_90347319.jpg
  inflating: /content/fer2013/train/surprise/Training_90460996.jpg
  inflating: /content/fer2013/train/surprise/Training_90519473.jpg
  inflating: /content/fer2013/train/surprise/Training_90544840.jpg
  inflating: /content/fer2013/train/surprise/Training_9059249.jpg
  inflating: /content/fer2013/train/surprise/Training_90625118.jpg
  inflating: /content/fer2013/train/surprise/Training_90678511.jpg
  inflating: /content/fer2013/train/surprise/Training_90690092.jpg
  inflating: /content/fer2013/train/surprise/Training_90709299.jpg
  inflating: /content/fer2013/train/surprise/Training_90712913.jpg
  inflating: /content/fer2013/train/surprise/Training_90736173.jpg
  inflating: /content/fer2013/train/surprise/Training_90738425.jpg
```

```
#finding the exact paths of my train and test datasets
import os
print(os.getcwd())
```

```
/content
```

```
!ls /content/fer2013/
!ls /content/fer2013/train/
!ls /content/fer2013/test/
```

```
test  train
angry  disgust  fear  happy  neutral  sad  surprise
angry  disgust  fear  happy  neutral  sad  surprise
```

```
train_directory = '/content/fer2013/train'
test_directory = '/content/fer2013/test'
```

```
!ls /content/fer2013/
```

```
test  train
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
preprocess_fun = tf.keras.applications.densenet.preprocess_input

#PRE-PROCESSING
```

```
train_datagen = ImageDataGenerator(horizontal_flip=True,
                                   rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   validation_split=0.2
                                  )
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(directory=train_directory,
                                                    target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                    batch_size=BATCH_SIZE,
                                                    shuffle=True,
                                                    color_mode="rgb",
                                                    class_mode="categorical",
                                                    subset="training",
                                                    seed=SEED
                                                   )

validation_generator = train_datagen.flow_from_directory(directory=train_directory,
                                                         target_size=(IMG_HEIGHT, IMG_WIDTH)
                                                         batch_size=BATCH_SIZE,
                                                         shuffle=True,
                                                         color_mode="rgb",
                                                         class_mode="categorical",
                                                         subset="validation",
                                                         seed=SEED
                                                        )

test_generator = test_datagen.flow_from_directory(directory=test_directory,
                                                  target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                   batch_size=BATCH_SIZE,
                                                   shuffle=False,
                                                   color_mode="rgb",
                                                   class_mode="categorical",
                                                   seed=SEED
                                                 )
```

```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

Cleansing Data

- The data cleansing process included several critical steps: resizing all images to a uniform input shape for model consistency, normalizing pixel values for improved model training, and addressing any missing or corrupt data entries. Python tools like NumPy and Pandas were

employed to detect and manage these data quality issues. Additionally, the data was checked for duplicates and inconsistencies to maintain the dataset's integrity.

```
# Define function to remove corrupt images
def remove_corrupt_images(directory):
    for folder in os.listdir(directory):
        folder_path = os.path.join(directory, folder)
        if os.path.isdir(folder_path):
            for filename in os.listdir(folder_path):
                filepath = os.path.join(folder_path, filename)
                try:
                    img = cv2.imread(filepath)
                    if img is None or not isinstance(img, np.ndarray):
                        print(f"Removing corrupt image: {filepath}")
                        os.remove(filepath)
                except Exception as e:
                    print(f"Error loading image {filepath}: {e}")
                    os.remove(filepath)

train_directory = '/content/fer2013/train'
test_directory = '/content/fer2013/test'

# Clean the training and testing datasets
remove_corrupt_images(train_directory)
remove_corrupt_images(test_directory)

print("Data cleaning complete.")

# Further data cleansing steps:
#  (resizing)
def preprocess_images(directory, target_size=(48, 48)):
    for folder in os.listdir(directory):
        folder_path = os.path.join(directory, folder)
        if os.path.isdir(folder_path):
            for filename in os.listdir(folder_path):
                filepath = os.path.join(folder_path, filename)
                try:
                    img = cv2.imread(filepath)
                    img_resized = cv2.resize(img, target_size)
                    cv2.imwrite(filepath, img_resized)
                except Exception as e:
                    print(f"Error processing image {filepath}: {e}")

# Preprocess images in both training and testing directories
preprocess_images(train_directory)
preprocess_images(test_directory)

print("Image resizing complete.")
```

```
Data cleaning complete.
Image resizing complete.
```

Exploratory Data Analysis

- The EDA phase involved examining the dataset's distribution across different emotional categories using visual plots such as histograms and bar charts (created with Matplotlib and Seaborn). Descriptive statistics were used to summarize the data properties, highlighting any class imbalances or distribution patterns. Correlation analysis was conducted to identify relationships between features, providing insights into which variables were significant for emotion differentiation. Feature extraction methods were applied, and the findings guided the feature engineering process.

General Problem Statement

- The primary challenge of this project is to develop a robust and accurate emotion detection model that can classify images into various emotional categories. The main problem we aim to address is: What are the key processes and techniques that contribute to achieving high accuracy in emotion classification using machine learning? The goal is to identify the best practices for data gathering, preprocessing, feature selection, model training, and evaluation to ensure reliable emotion detection results.

3 Data Science Questions for Exploratory Data Analysis (EDA)

1. What is the distribution of different emotional categories in the dataset, and are there significant imbalances?

Understanding the distribution of emotional categories helps identify potential class imbalances that could affect training and model performance. Analyzing this will guide decisions on whether to apply data augmentation or rebalancing techniques.

2. What are the most impactful features that differentiate one emotional category from another?

Identifying key features or visual characteristics that separate emotions provides insight into the dataset's structure and helps refine feature selection for model training. This ensures the model learns from the most relevant information.

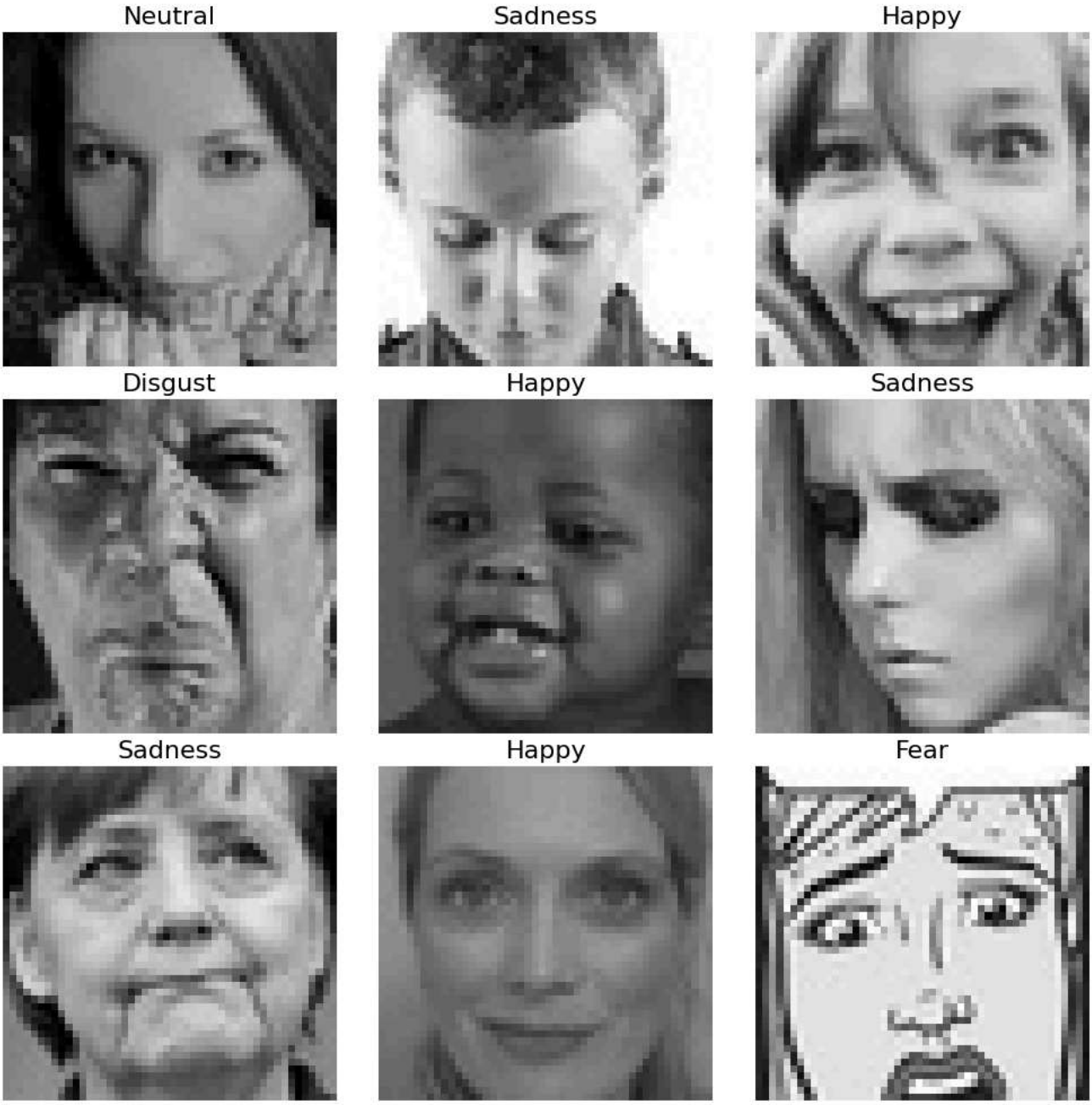
3. How do the correlations between different features influence the model's performance in classifying emotions?

Investigating feature correlations can reveal dependencies that may help or hinder model learning. This analysis assists in feature engineering, enabling more effective model training and improved

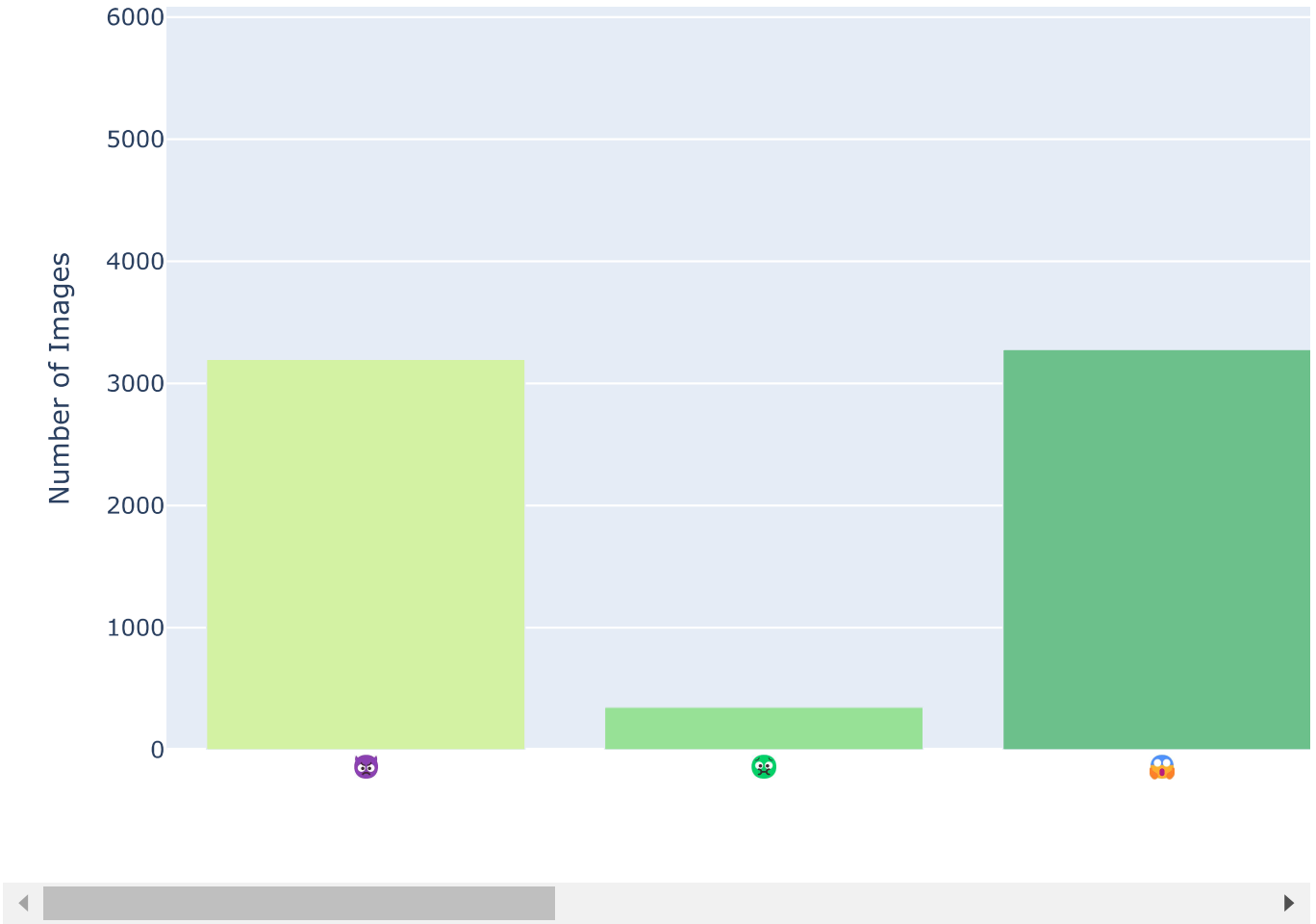
performance.

```
# Helper Functions
def display_one_image(image, title, subplot, color):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image)
    plt.title(title, fontsize=16)

def display_nine_images(images, titles, title_colors=None):
    subplot = 331
    plt.figure(figsize=(10,10))
    for i in range(9):
        color = 'black' if title_colors is None else title_colors[i]
        display_one_image(images[i], titles[i], 331+i, color)
    plt.tight_layout()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    plt.show()

def image_title(label, prediction):
  # Both prediction (probabilities) and label (one-hot) are arrays with one item per class.
    class_idx = np.argmax(label, axis=-1)
    prediction_idx = np.argmax(prediction, axis=-1)
    if class_idx == prediction_idx:
        return f'{CLASS_LABELS[prediction_idx]} [correct]', 'black'
    else:
        return f'{CLASS_LABELS[prediction_idx]} [incorrect, should be {CLASS_LABELS[class_ic

def get_titles(images, labels, model):
    predictions = model.predict(images)
    titles, colors = [], []
    for label, prediction in zip(classes, predictions):
        title, color = image_title(label, prediction)
        titles.append(title)
        colors.append(color)
    return titles, colors



# Display a few images with labels
img_datagen = ImageDataGenerator(rescale = 1./255)

train_datagen = ImageDataGenerator(horizontal_flip=True,
                                   rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   rescale=1./255,
                                   validation_split=0.25
                                  )

img_generator = img_datagen.flow_from_directory(directory = train_directory,
```

```
        target_size = (IMG_HEIGHT, IMG_WIDTH),
        batch_size = BATCH_SIZE,
        shuffle  = True ,
        color_mode = "rgb",
        class_mode = "categorical",
        seed = 24
    )
clear_output()

images, classes = next(img_generator)
class_idxs = np.argmax(classes, axis=-1)
labels = [CLASS_LABELS[idx] for idx in class_idxs]
display_nine_images(images, labels)
```


Neutral
Sadness
Happy
Disgust
Happy
Sadness
Sadness
Happy
Fear

```
# Data distribution visualization
fig = px.bar(x = CLASS_LABELS_EMOJIS,
             y = [list(train_generator.classes).count(i) for i in np.unique(train_generator.
             color = np.unique(train_generator.classes) ,
             color_continuous_scale="Emrld")
fig.update_xaxes(title="Emotions")
fig.update_yaxes(title = "Number of Images")
fig.update_layout(showlegend = True,
    title = {
        'text': 'Train Data Distribution ',
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
fig.show()
```



Modelling (and Training)

- A Convolutional Neural Network (CNN) was designed and trained using TensorFlow and Keras. The code included callbacks for early stopping to prevent overfitting and learning rate adjustments for efficient training. The model was trained on a split dataset with validation data to track performance metrics such as accuracy and loss during each epoch. The features selected for model training were based on insights from the EDA phase, emphasizing those that contributed most to distinguishing emotions.

```
# Define DenseNet169 feature extractor
def feature_extractor(inputs):
    feature_extractor = tf.keras.applications.DenseNet169(
        input_shape=(IMG_HEIGHT, IMG_WIDTH, 3),
        include_top=False,
        weights='imagenet'
    )(inputs)
    return feature_extractor


# Define classifier layers
def classifier(inputs):
    x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
    x = tf.keras.layers.Dense(2048, activation='relu', kernel_regularizer=tf.keras.regulariz
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(1024, activation='relu', kernel_regularizer=tf.keras.regulariz
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(512, activation='relu', kernel_regularizer=tf.keras.regularize
    x = tf.keras.layers.Dropout(0.4)(x)
    x = tf.keras.layers.Dense(NUM_CLASSES, activation='softmax', name='classification')(x)
    return x


# Define the full model
def final_model(inputs):
    densenet_feature_extractor = feature_extractor(inputs)
    classification_output = classifier(densenet_feature_extractor)
    return classification_output


# Compile the model
def define_compile_model():
    inputs = tf.keras.layers.Input(shape=(IMG_HEIGHT, IMG_WIDTH, 3))
    classification_output = final_model(inputs)
    model = tf.keras.Model(inputs=inputs, outputs=classification_output)
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
```

```
                  metrics=['accuracy'])
    return model


model = define_compile_model()
for layer in model.layers[1].layers[:30]:
    layer.trainable = False

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/dense
51877672/51877672 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "functional"
```

Layer (type)	Output Shape	P
input_layer (InputLayer)	(None, 48, 48, 3)	
densenet169 (Functional)	(None, 1, 1, 1664)	12,6
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1664)	
dense (Dense)	(None, 2048)	3,4
dropout (Dropout)	(None, 2048)	
dense_1 (Dense)	(None, 1024)	2,0
dropout_1 (Dropout)	(None, 1024)	
dense_2 (Dense)	(None, 512)	5
dropout_2 (Dropout)	(None, 512)	
classification (Dense)	(None, 7)	

```
Total params: 18,679,367 (71.26 MB)
```

```
model = define_compile_model()
clear_output()
```

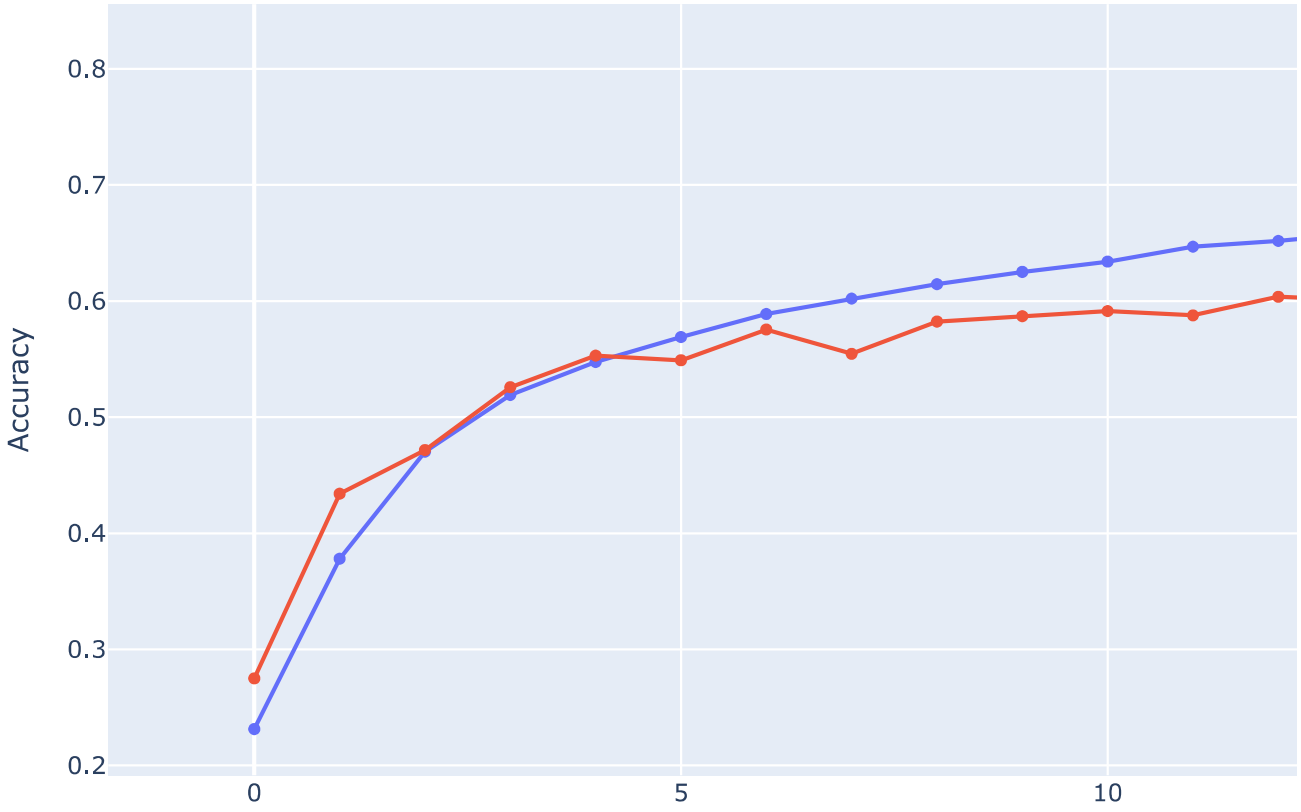
* In this part of the phase, I trained the model with 30 epochs for early stopping to prevent overfitting. I have also added a learning rate reduction callback to assist the model's processing in case it declines in validation accuracy.

```
# Training the model with early stopping
earlyStoppingCallback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
```

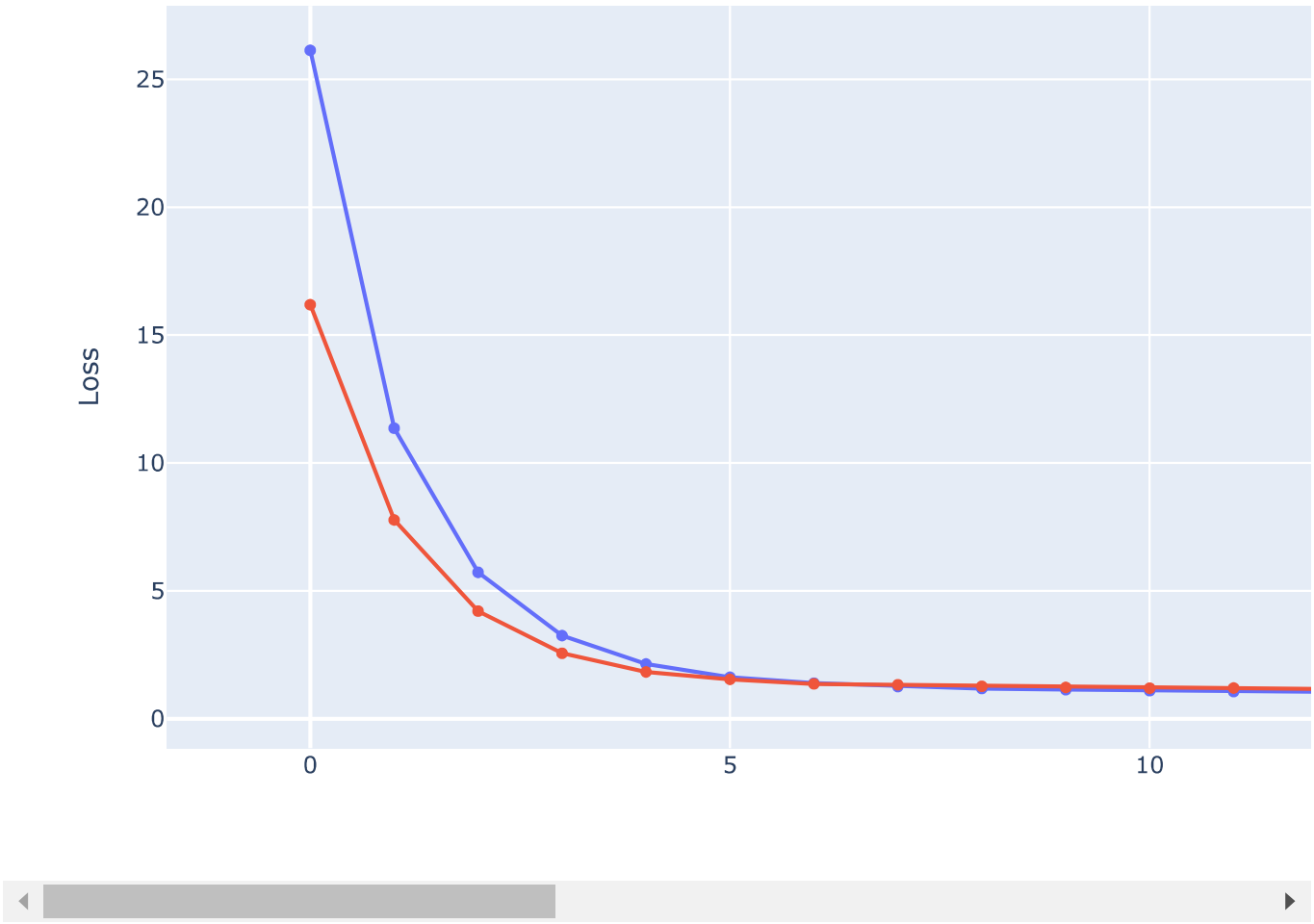
```
                                                patience= 8,
                                                verbose= 1 ,
                                                restore_best_weights=True
                                               )

# Learning rate reduction callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.5,
                              patience=3,
                              min_lr=1e-07,
                              verbose=1)

checkpoint = ModelCheckpoint('best_model.keras',
                             save_best_only=True,
                             monitor='val_loss',
                             mode='min',
                             verbose=1)
history = model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=validation_generator,
    callbacks=[earlyStoppingCallback, reduce_lr, checkpoint]
)

history = pd.DataFrame(history.history)
history = pd.concat([history], ignore_index=True)
```

```
Epoch 23: val_loss improved from 1.12877 to 1.12261, saving model to best_model.keras
359/359 ━━━━━━━━━━━━━━━━━━━━ 1099s 3s/step - accuracy: 0.7785 - loss: 0.6967 - val_ac
Epoch 24/30
359/359 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.7856 - loss: 0.6731
Epoch 24: val_loss did not improve from 1.12261
359/359 ━━━━━━━━━━━━━━━━━━━━ 1060s 3s/step - accuracy: 0.7856 - loss: 0.6731 - val_ac
Epoch 25/30
359/359 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.7899 - loss: 0.6706
Epoch 25: val_loss did not improve from 1.12261
359/359 ━━━━━━━━━━━━━━━━━━━━ 1055s 3s/step - accuracy: 0.7899 - loss: 0.6706 - val_ac
Epoch 26/30
359/359 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.7995 - loss: 0.6377
Epoch 26: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.

Epoch 26: val_loss did not improve from 1.12261
359/359 ━━━━━━━━━━━━━━━━━━━━ 1127s 3s/step - accuracy: 0.7995 - loss: 0.6378 - val_ac
Epoch 27/30
359/359 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8021 - loss: 0.6285
Epoch 27: val_loss did not improve from 1.12261
359/359 ━━━━━━━━━━━━━━━━━━━━ 1070s 3s/step - accuracy: 0.8021 - loss: 0.6285 - val_ac
Epoch 28/30
359/359 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8086 - loss: 0.6139
Epoch 28: val_loss did not improve from 1.12261
359/359 ━━━━━━━━━━━━━━━━━━━━ 1103s 3s/step - accuracy: 0.8086 - loss: 0.6139 - val_ac
Epoch 29/30
359/359 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8164 - loss: 0.5970
Epoch 29: ReduceLROnPlateau reducing learning rate to 6.24999984211172e-06.

Epoch 29: val_loss did not improve from 1.12261
359/359 ━━━━━━━━━━━━━━━━━━━━ 1090s 3s/step - accuracy: 0.8164 - loss: 0.5970 - val_ac
```

```
#training plot (accuracy vs number of epochs)
x = px.line(data_frame= history , y= ["accuracy" , "val_accuracy"] ,markers = True )
x.update_xaxes(title="Number of Epochs")
x.update_yaxes(title = "Accuracy")
x.update_layout(showlegend = True,
    title = {
        'text': 'Accuracy vs Number of Epochs',
        'y':0.94,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
x.show()
```




```
#training plot (loss vs number of epochs)
x = px.line(data_frame= history ,
            y= ["loss" , "val_loss"] , markers = True )
x.update_xaxes(title="Number of Epochs")
x.update_yaxes(title = "Loss")
x.update_layout(showlegend = True,
    title = {
        'text': 'Loss vs Number of Epochs',
        'y':0.94,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
x.show()
```



Answers to the 3 Data Science Questions:

1. **Distribution of Emotional Categories**: The dataset analysis revealed imbalances, with some emotions represented more frequently than others. Addressing this required techniques like data augmentation or re-sampling to create a balanced training set.

2. **Key Features for Differentiation:** Visual characteristics such as facial muscle movements and expressions were identified as impactful features. Feature extraction using convolutional layers in the CNN highlighted these important details.

3. **Feature Correlations and Impact:** The correlation analysis showed that certain features had strong relationships that influenced model performance. Adjustments were made during feature selection to maximize predictive power while minimizing redundancy.

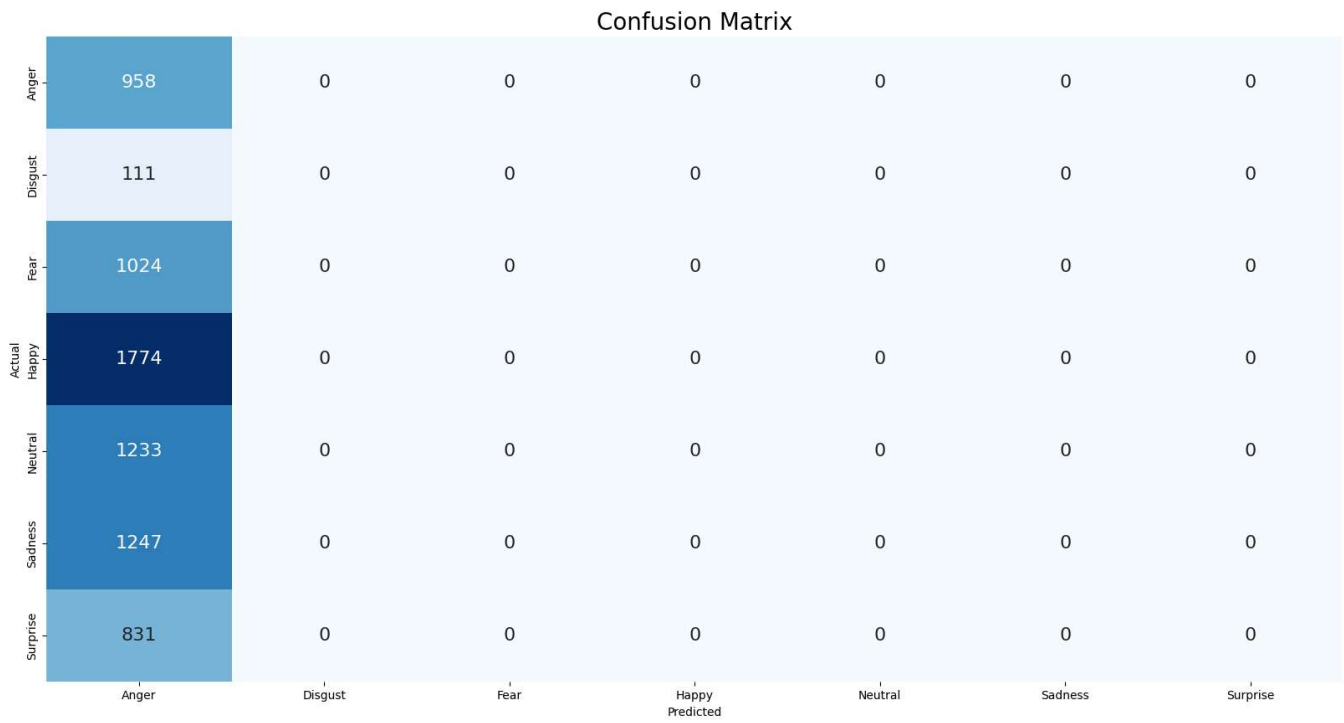
Evaluation

- The model's performance was evaluated using metrics such as precision, recall, F1-score, and a confusion matrix generated with scikit-learn. The ROC-AUC curve provided a comprehensive view of the model's capability to distinguish between different emotional classes. These evaluations were critical in identifying strengths and areas for improvement. The final evaluation included actionable insights, with recommendations for future enhancements such as tuning hyperparameters, expanding the dataset, and exploring additional classifiers for improved performance.

```
# Evaluate model on test data
model.evaluate(test_generator)
preds = model.predict(test_generator)
y_preds = np.argmax(preds , axis = 1 )
y_test = np.array(test_generator.labels)
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 64s 564ms/step - accuracy: 0.3941 - loss: 1.9446
113/113 ━━━━━━━━━━━━━━━━━━━━ 75s 606ms/step
```

```
# confusion matrix

cm_data = confusion_matrix(y_test , y_preds)
cm = pd.DataFrame(cm_data, columns=CLASS_LABELS, index = CLASS_LABELS)
cm.index.name = 'Actual'
cm.columns.name = 'Predicted'
plt.figure(figsize = (20,10))
plt.title('Confusion Matrix', fontsize = 20)
sns.set(font_scale=1.2)
ax = sns.heatmap(cm, cbar=False, cmap="Blues", annot=True, annot_kws={"size": 16}, fmt='g')
```




```
# Classification report
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.13      1.00      0.24       958
           1       0.00      0.00      0.00       111
           2       0.00      0.00      0.00      1024
           3       0.00      0.00      0.00      1774
           4       0.00      0.00      0.00      1233
           5       0.00      0.00      0.00      1247
           6       0.00      0.00      0.00       831

    accuracy                           0.13      7178
   macro avg       0.02      0.14      0.03      7178
weighted avg       0.02      0.13      0.03      7178
```

```
# ROC-AUC Curve
fig, c_ax = plt.subplots(1,1, figsize = (15,8))

def multiclass_roc_auc_score(y_test, y_pred, average="macro"):
    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    for (idx, c_label) in enumerate(CLASS_LABELS):
        fpr, tpr, thresholds = roc_curve(y_test[:,idx].astype(int), y_pred[:,idx])
        c_ax.plot(fpr, tpr,lw=2, label = '%s (AUC:%0.2f)'  % (c_label, auc(fpr, tpr)))
    c_ax.plot(fpr, fpr, 'black',linestyle='dashed', lw=4, label = 'Random Guessing')
    return roc_auc_score(y_test, y_pred, average=average)

print('ROC AUC score:', multiclass_roc_auc_score(y_test , preds  , average = "micro"))
plt.xlabel('FALSE POSITIVE RATE', fontsize=18)
plt.ylabel('TRUE POSITIVE RATE', fontsize=16)
plt.legend(fontsize = 11.5)
plt.show()
```

```
ROC AUC score: 0.5866111641007659
```