Leveraging Multimodal Data Fusion for Improved Emotion Detection System

MSc Research Project
M. Sc. Data Analytics

Kamran Habib
Student ID: 22159827

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Kamran Habib
Student ID:	22159827
Programme:	M. Sc. Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Furqan Rustam
Submission Due Date:	31/01/2024
Project Title:	Leveraging Multimodal Data Fusion for Improved Emotion Detection System
Word Count:	277
Page Count:	30

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Kamran Habib
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Kamran Habib
22159827

1 Introduction

In this configuration manual, the author has given end-to-end details of all the process that was equipped in the study for the multi-modal emotion recognition. Based on these aspects, the results were predicted. This manual also highlights the technical study which shows all the Python libraries which were used to study the working of the different traditional machine and deep learning models which was used for emotion analysis. The aim of this configuration manual is to make it easy for the reader to understand how things were processed from start to end.

2 System Specification

2.1 Hardware Specification

Following are the hardware specification of the system that was used to develop the project:

Component	**Specification**
Processor	Ryzen 7 5000 Series
RAM	16GB
Storage	512GB
Graphics Card	RTX 3060 4GB
Operating System	Windows 11

Table 1: System Specifications

2.2 Software Specification

The specifications of the software utilized for the system were as follows:

Software	**Specifications**
Operating System	Windows 11 (64 bit)
IDE	Jupyter Notebook
Scripting Language	Python 3.7

Table 2: Software Specifications

3 Software Tools

Following are the software tools that were used to implement the project:

3.1 Python

This project was developed with the help of Python programming language. It's useful libraries for analysis, visualization, machine learning, and deep learning was the main reason to chose it. Python was downloaded from the main website [1]. Figure 1 shows the download page of Python's official website.

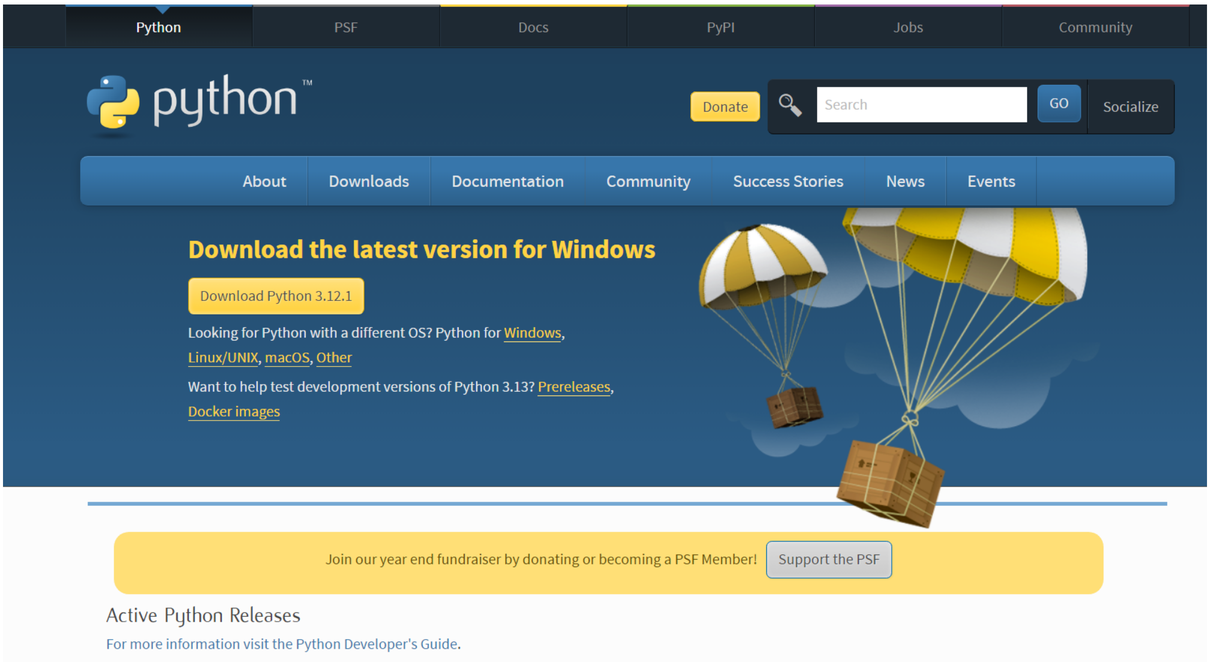

Figure 1: Python Download Page

3.2 Jupyter Noteboook

Jupyter Notebook was used as a compiler to run the code as it allows the users to implement all the code in one place and execute the codes in small parts like cells to allow the audience to check the output of each code with ease. Jupyter Notebook was downloaded from its official website and Figure 2 illustrates its download page [2] .

4 Implementation

Following are the Python packages which were installed using pip and used to implement the project:

- Pandas
- Numpy

[1] `https://www.python.org/downloads/`
[2] `https://jupyter.org/`

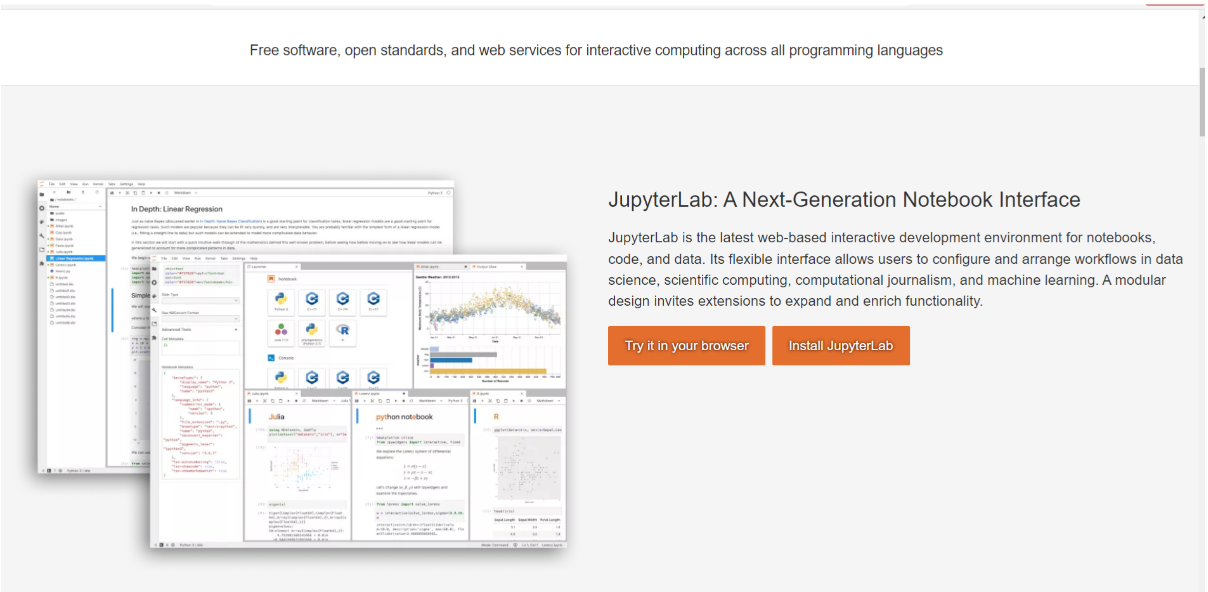

Figure 2: Jupyter Download Page

- Matplotlib
- Keras
- NLTK
- text2emotion
- transformers
- wordcloud
- opencv-python
- Pillow

Figure 3 Shows all the necessary libraries.
Figure 4 shows text dataset loading.
Figure 5 shows preprocessing of text data which includes Conversion to lowercase, Removal of HTML tags, Removal of URLs, Removal of numbers, Tokenization, Removal of stopwords, Stemming, Lemmatization, and Joining tokens.

```python
# Standard Libraries
import os
import re
import operator

# Data Manipulation and Analysis Libraries
import numpy as np
import pandas as pd

# Machine Learning Libraries
from sklearn.model_selection import (
    train_test_split,
    GridSearchCV,
    cross_val_score,
    StratifiedKFold,
)
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
)
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from tensorflow.keras.models import Model

# Natural Language Processing (NLP) Libraries
import nltk
from nltk.tokenize import RegexpTokenizer, word_tokenize
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.corpus import stopwords
import text2emotion as te
```

(a) Necessary Library 1

```python
# Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Deep Learning Libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import (
    Embedding,
    Conv1D,
    Conv2D,
    MaxPooling1D,
    MaxPooling2D,
    Flatten,
    Dense,
    Dropout,
    GlobalAveragePooling1D,
    GlobalMaxPooling1D,
    Input,
    concatenate,
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras import layers, models

# Transformers Library
from transformers import BertTokenizer, TFBertModel
```

(b) Necessary Library 2

```python
# Image Processing Libraries
import cv2
from PIL import Image
import numpy as np
from wordcloud import WordCloud
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import KFold
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score
from skimage.feature import hog

from memory_profiler import memory_usage
import time
```

(c) Necessary Library 3

Figure 3: All the necessary libraries

```
file = r"C:\Users\Kamran Habib\Desktop\Research in Computing\Datasets\TextemotionKaggle\tweet_emotions.csv"
data=pd.read_csv(file)
data.head()
```

	tweet_id	sentiment	content
0	1956967341	empty	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	wants to hang out with friends SOON!
4	1956968416	neutral	@dannycastillo We want to trade with someone w...

Figure 4: Loading and checking Text Dataset

```
#Preprocessing of text tweets to remove : Punction, Numbers, stropwords hash tages , links stemming, lematization
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

def preprocess(sentence):
    sentence=str(sentence)
    sentence = sentence.lower()
    sentence=sentence.replace('{html}',"")
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, '', sentence)
    rem_url=re.sub(r'http\S+', '',cleantext)
    rem_num = re.sub('[0-9]+', '', rem_url)
    tokenizer = RegexpTokenizer(r'\w+')
    tokens = tokenizer.tokenize(rem_num)
    filtered_words = [w for w in tokens if len(w) > 2 if not w in stopwords.words('english') and "amp"]
    stem_words=[stemmer.stem(w) for w in filtered_words]
    lemma_words=[lemmatizer.lemmatize(w) for w in stem_words]
    return " ".join(lemma_words)


data['cleanText']=data['content'].map(lambda s:preprocess(s))
```

Figure 5: Prepocessing of Text Data

[26]:
```
vectorizer = TfidfVectorizer()
X_train_tf = vectorizer.fit_transform(X_train)
X_test_tf = vectorizer.transform(X_test)
```

[30]:
```
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the Naive Bayes Classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_tf, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
y_pred = nb_classifier.predict(X_test_tf)
nb_accuracy = accuracy_score(y_test, y_pred)
nb_report = classification_report(y_test, y_pred)
nb_time_taken = end_time - start_time

# Print the results
print("Naive Bayes Accuracy: {:.2f}%".format(nb_accuracy * 100))
print("Time taken: {:.2f} seconds".format(nb_time_taken))
print("Classification Report:\n", nb_report)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Get all parameters of the model
parameters = nb_classifier.get_params()

# Print the parameters
print("Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
Naive Bayes Accuracy: 56.59%
Time taken: 0.03 seconds
```

Figure 6: TFIDF for Naive Bayes

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the SVM Classifier
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train_tf, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
y_pred = svm_classifier.predict(X_test_tf)
svm_accuracy = accuracy_score(y_test, y_pred)
svm_report = classification_report(y_test, y_pred)
svm_time_taken = end_time - start_time

# Print the results
print("SVM Accuracy: {:.2f}%".format(svm_accuracy * 100))
print("Time taken: {:.2f} seconds".format(svm_time_taken))
print("Classification Report:\n", svm_report)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Get all parameters of the SVM model
parameters = svm_classifier.get_params()

# Print the parameters
print("SVM Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
SVM Accuracy: 81.46%
Time taken: 17.20 seconds
```

Figure 7: TFIDF for SVM

```
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100)
rf_classifier.fit(X_train_tf, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
y_pred = rf_classifier.predict(X_test_tf)
rf_accuracy = accuracy_score(y_test, y_pred)
rf_report = classification_report(y_test, y_pred)
rf_time_taken = end_time - start_time

# Print the results
print("Random Forest Accuracy: {:.2f}%".format(rf_accuracy * 100))
print("Time taken: {:.2f} seconds".format(rf_time_taken))
print("Classification Report:\n", rf_report)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Get all parameters of the Random Forest model
parameters = rf_classifier.get_params()

# Print the parameters
print("Random Forest Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
Random Forest Accuracy: 75.65%
Time taken: 66.67 seconds
```

Figure 8: TFIDF for Random Forest

```
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Define the KNN classifier and parameter grid
knn_classifier = KNeighborsClassifier()
param_grid = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance']}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=knn_classifier, param_grid=param_grid, cv=5)
grid_search.fit(X_train_tf, y_train)

# Get the best parameters and estimator
best_params = grid_search.best_params_
best_knn_classifier = grid_search.best_estimator_

# Fit the best classifier to the training data
best_knn_classifier.fit(X_train_tf, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
y_pred = best_knn_classifier.predict(X_test_tf)
knn_accuracy = accuracy_score(y_test, y_pred)
knn_report = classification_report(y_test, y_pred)
knn_time_taken = end_time - start_time

# Print the results
print(f"Best Parameters: {best_params}")
print(f"KNN Accuracy: {knn_accuracy}")
print("Classification Report:\n", knn_report)
print("Time taken: {:.2f} seconds".format(knn_time_taken))

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Best Parameters: {'n_neighbors': 3, 'weights': 'distance'}
KNN Accuracy: 0.37603734439834025
```

Figure 9: TFIDF for KNN

```python
# learn training data vocabulary, then use it to create a document-term matrix
vect = CountVectorizer()
X_train_dtf = vect.fit_transform(X_train)
X_test_dtf = vect.transform(X_test)
```

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the Naive Bayes Classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_dtf, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
y_pred = nb_classifier.predict(X_test_dtf)
nb2_accuracy = accuracy_score(y_test, y_pred)
nb2_time_taken = end_time - start_time

# Print the results
print(f"Naive Bayes Accuracy: {nb2_accuracy}")
print("Time taken: {:.2f} seconds".format(nb2_time_taken))
print("Classification Report:\n", classification_report(y_test, y_pred))

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Get all parameters of the Naive Bayes model
parameters = nb_classifier.get_params()

# Print the parameters
print("Naive Bayes Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
Naive Bayes Accuracy: 0.6815352697095436
Time taken: 0.03 seconds
```

Figure 10: BoW for Naive Bayes

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the SVM Classifier
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train_dtf, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
y_pred = svm_classifier.predict(X_test_dtf)
svm2_accuracy = accuracy_score(y_test, y_pred)
svm2_time_taken = end_time - start_time

# Print the results
print(f"SVM Accuracy: {svm2_accuracy}")
print("Time taken: {:.2f} seconds".format(svm2_time_taken))
print("Classification Report:\n", classification_report(y_test, y_pred))

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Get all parameters of the SVM model
parameters = svm_classifier.get_params()

# Print the parameters
print("SVM Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
SVM Accuracy: 0.833246887966805
Time taken: 14.53 seconds
```

Figure 11: BoW for SVM

```
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100)
rf_classifier.fit(X_train_dtf, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
y_pred = rf_classifier.predict(X_test_dtf)
rf2_accuracy = accuracy_score(y_test, y_pred)
rf2_time_taken = end_time - start_time

# Print the results
print(f"Random Forest Accuracy: {rf2_accuracy}")
print("Time taken: {:.2f} seconds".format(rf2_time_taken))
print("Classification Report:\n", classification_report(y_test, y_pred))

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Get all parameters of the Random Forest model
parameters = rf_classifier.get_params()

# Print the parameters
print("Random Forest Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
Random Forest Accuracy: 0.7598547717842323
Time taken: 71.93 seconds
```

Figure 12: BoW for Random Forest

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Define the KNN classifier and parameter grid
knn_classifier = KNeighborsClassifier()
param_grid = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance']}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=knn_classifier, param_grid=param_grid, cv=5)
grid_search.fit(X_train_dtf, y_train)

# Get the best parameters and estimator
best_params = grid_search.best_params_
best_knn_classifier = grid_search.best_estimator_

# Fit the best classifier to the training data
best_knn_classifier.fit(X_train_dtf, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
y_pred = best_knn_classifier.predict(X_test_dtf)
knn2_accuracy = accuracy_score(y_test, y_pred)
knn2_time_taken = end_time - start_time

# Print the results
print(f"Best Parameters: {best_params}")
print(f"KNN Accuracy: {knn2_accuracy}")
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Time taken: {:.2f} seconds".format(knn2_time_taken))

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Optionally, print all parameters of the best KNN model
print("Best KNN Model Parameters:")
for param, value in best_knn_classifier.get_params().items():
    print(f"{param}: {value}")
```

```
Best Parameters: {'n_neighbors': 5, 'weights': 'distance'}
KNN Accuracy: 0.47199170124481327
```

Figure 13: BoW for KNN

```
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
tokenizer.fit_on_texts(data['cleanText'])
sequences = tokenizer.texts_to_sequences(data['cleanText'])
word_index = tokenizer.word_index
max_length = 100  # Adjust based on your data
padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post', truncating='post')
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(data["Emotion"])
num_classes = len(np.unique(encoded_labels))
# Splitting the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, encoded_labels, test_size=0.2, random_state=42)
initial_mem_usage = memory_usage()[0]
# Start timer
start_time = time.time()
# Model definition
model = Sequential([
    Embedding(10000, 128, input_length=max_length),  # Embedding layer
    Conv1D(128, 5, activation='relu'),  # Convolutional layer
    MaxPooling1D(pool_size=4),  # Pooling layer
    Conv1D(128, 5, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')  # Output layer for classification
])
# Model compilation
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Model training
model.fit(padded_sequences, encoded_labels, epochs=10, batch_size=32, validation_split=0.2)
# Stop timer
end_time = time.time()
# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)
# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage
# Print the results
print("Model Architecture:\n")
model.summary()
print("\nTest Accuracy:", test_accuracy)
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")
```

Figure 14: CNN for Text

```
Epoch 1/10
1000/1000 [==============================] - 36s 36ms/step - loss: 0.8345 - accuracy: 0.7014 - val_loss: 0.4290 - val_accuracy:
0.8646
Epoch 2/10
1000/1000 [==============================] - 35s 35ms/step - loss: 0.3595 - accuracy: 0.8845 - val_loss: 0.3919 - val_accuracy:
0.8760
Epoch 3/10
1000/1000 [==============================] - 34s 34ms/step - loss: 0.1918 - accuracy: 0.9405 - val_loss: 0.4027 - val_accuracy:
0.8776
Epoch 4/10
1000/1000 [==============================] - 34s 34ms/step - loss: 0.1010 - accuracy: 0.9684 - val_loss: 0.5074 - val_accuracy:
0.8689
Epoch 5/10
1000/1000 [==============================] - 34s 34ms/step - loss: 0.0674 - accuracy: 0.9788 - val_loss: 0.5740 - val_accuracy:
0.8677
Epoch 6/10
1000/1000 [==============================] - 34s 34ms/step - loss: 0.0484 - accuracy: 0.9851 - val_loss: 0.6788 - val_accuracy:
0.8626
Epoch 7/10
1000/1000 [==============================] - 36s 36ms/step - loss: 0.0389 - accuracy: 0.9877 - val_loss: 0.6786 - val_accuracy:
0.8619
Epoch 8/10
1000/1000 [==============================] - 34s 34ms/step - loss: 0.0354 - accuracy: 0.9883 - val_loss: 0.8283 - val_accuracy:
0.8534
Epoch 9/10
1000/1000 [==============================] - 34s 34ms/step - loss: 0.0315 - accuracy: 0.9902 - val_loss: 0.8237 - val_accuracy:
0.8593
Epoch 10/10
1000/1000 [==============================] - 34s 34ms/step - loss: 0.0278 - accuracy: 0.9904 - val_loss: 0.8329 - val_accuracy:
0.8620
250/250 - 1s - loss: 0.1709 - accuracy: 0.9684 - 1s/epoch - 5ms/step
Model Architecture:

Model: "sequential"
```

Figure 15: CNN Text Accuracy

```python
img_dataset_path = r"C:\Users\Kamran Habib\Desktop\Project Extension\Image Dataset"
```

```python
# Initialize lists to hold images and labels
images = []
labels = []

# Define image size for resizing
IMG_SIZE = (48, 48)

# Loop through each emotion subfolder and load images
for emotion_folder in os.listdir(img_dataset_path):
    emotion_path = os.path.join(img_dataset_path, emotion_folder)

    # Only consider directories (emotion subfolders)
    if os.path.isdir(emotion_path):
        for img_name in os.listdir(emotion_path):
            img_path = os.path.join(emotion_path, img_name)

            # Read Image
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

            #Preprocess image
            img = cv2.resize(img, IMG_SIZE)
            img = img / 255.0  # Normalize pixel values

            images.append(img)
            labels.append(emotion_folder)

# Convert to numpy arrays
images = np.array(images).reshape(-1, IMG_SIZE[0], IMG_SIZE[1], 1)

fig, axes = plt.subplots(1, 5, figsize=(15, 3))
for i, ax in enumerate(axes):
    ax.imshow(images[i].reshape(IMG_SIZE), cmap='gray')
    ax.set_title(labels[i])
plt.show()
```

Figure 16: Preprocessing of Image files

```python
# Define HOG parameters
hog_params = {
    'orientations': 9,
    'pixels_per_cell': (8, 8),
    'cells_per_block': (2, 2),
    'block_norm': 'L2-Hys',
    'visualize': False,
    'transform_sqrt': True,
    'feature_vector': True
}

# Initialize lists to hold HOG features and labels
hog_features = []

# Loop through each image and compute HOG features
for img in images:
    img_flat = img[:, :, 0]  # Remove the channel dimension
    hog_feature = hog(img_flat, **hog_params)
    hog_features.append(hog_feature)

# Convert HOG features to a numpy array
hog_features = np.array(hog_features)

# Split data into training and testing sets
X_train_hog, X_test_hog, y_train_hog, y_test_hog = train_test_split(hog_features,
                                                                     labels, test_size=0.2, shuffle=True, random_state=42)
```

Figure 17: HOG Parameters for Image

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Create and train the Random Forest classifier with HOG features
rf_classifier_hog = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier_hog.fit(X_train_hog, y_train_hog)

# Stop timer
end_time = time.time()

# Make predictions
rf_predictions_hog = rf_classifier_hog.predict(X_test_hog)

# Evaluate the classifier
rf_accuracy_hog = accuracy_score(y_test_hog, rf_predictions_hog)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("Random Forest Classifier (HOG) Accuracy:", rf_accuracy_hog)
print("Classification Report:\n", classification_report(y_test_hog, rf_predictions_hog))
print("Confusion Matrix:\n", confusion_matrix(y_test_hog, rf_predictions_hog))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Get all parameters of the Random Forest model
parameters = rf_classifier_hog.get_params()

# Print the parameters
print("Random Forest (HOG) Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
Random Forest Classifier (HOG) Accuracy: 0.5505705394190872
```

Figure 18: HOG for Random Forest

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Create and train the Support Vector Classifier with HOG features
svc_classifier_hog = SVC(kernel='linear', C=1.0, random_state=42)
svc_classifier_hog.fit(X_train_hog, y_train_hog)

# Stop timer
end_time = time.time()

# Make predictions
svc_predictions_hog = svc_classifier_hog.predict(X_test_hog)

# Evaluate the classifier
svc_accuracy_hog = accuracy_score(y_test_hog, svc_predictions_hog)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("Support Vector Classifier (HOG) Accuracy:", svc_accuracy_hog)
print("Classification Report:\n", classification_report(y_test_hog, svc_predictions_hog))
print("Confusion Matrix:\n", confusion_matrix(y_test_hog, svc_predictions_hog))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Get all parameters of the SVC model
parameters = svc_classifier_hog.get_params()

# Print the parameters
print("Support Vector Classifier (HOG) Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
Support Vector Classifier (HOG) Accuracy: 0.5324170124481328
```

Figure 19: HOG for SVM/SVC

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Create and train the Multinomial Naive Bayes classifier with HOG features
mnb_classifier_hog = MultinomialNB()
mnb_classifier_hog.fit(X_train_hog, y_train_hog)

# Stop timer
end_time = time.time()

# Make predictions
mnb_predictions_hog = mnb_classifier_hog.predict(X_test_hog)

# Evaluate the classifier
mnb_accuracy_hog = accuracy_score(y_test_hog, mnb_predictions_hog)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("Multinomial Naive Bayes (HOG) Accuracy:", mnb_accuracy_hog)
print("Classification Report:\n", classification_report(y_test_hog, mnb_predictions_hog))
print("Confusion Matrix:\n", confusion_matrix(y_test_hog, mnb_predictions_hog))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Get all parameters of the Multinomial Naive Bayes model
parameters = mnb_classifier_hog.get_params()

# Print the parameters
print("Multinomial Naive Bayes (HOG) Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
Multinomial Naive Bayes (HOG) Accuracy: 0.4927385892116183
```

Figure 20: HOG for Naive Bayes

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Create and train the K-Nearest Neighbors classifier with HOG features
knn_classifier_hog = KNeighborsClassifier(n_neighbors=5)
knn_classifier_hog.fit(X_train_hog, y_train_hog)

# Stop timer
end_time = time.time()

# Make predictions
knn_predictions_hog = knn_classifier_hog.predict(X_test_hog)

# Evaluate the classifier
knn_accuracy_hog = accuracy_score(y_test_hog, knn_predictions_hog)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("K-Nearest Neighbors Accuracy:", knn_accuracy_hog)
print("Classification Report:\n", classification_report(y_test_hog, knn_predictions_hog))
print("Confusion Matrix:\n", confusion_matrix(y_test_hog, knn_predictions_hog))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Get all parameters of the KNN model
parameters = knn_classifier_hog.get_params()

# Print the parameters
print("K-Nearest Neighbors Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
K-Nearest Neighbors Accuracy: 0.5350103734439834
```

Figure 21: HOG for KNN

```
# Initialize lists to hold images and labels
images = []
labels = []

# Define image size for resizing
IMG_SIZE = (64, 64)

# Loop through each emotion folder
for emotion_folder in os.listdir(img_dataset_path):
    emotion_folder_path = os.path.join(img_dataset_path, emotion_folder)

    if os.path.isdir(emotion_folder_path):
        for img_file in os.listdir(emotion_folder_path):
            img_path = os.path.join(emotion_folder_path, img_file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            img = cv2.resize(img, IMG_SIZE)

            # Normalize pixel values
            img = img / 255.0

            images.append(img)
            labels.append(emotion_folder)

# Convert to numpy arrays and reshape
images = np.array(images).reshape(-1, IMG_SIZE[0], IMG_SIZE[1], 1)
labels = np.array(labels)

# Encode labels to one-hot vectors
label_encoder = LabelEncoder()
labels_encoded = to_categorical(label_encoder.fit_transform(labels))

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, shuffle=True, random_state=42)

# Flatten the images
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

encoder = LabelEncoder()
y_train_encoded = encoder.fit_transform(y_train)
y_test_encoded = encoder.transform(y_test)
```

Figure 22: CNN for Preprocessing Image

```
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Create and compile the CNN model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(X_train.shape[1:])),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(encoder.classes_), activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train_encoded, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test_encoded)

# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred_classes = tf.argmax(y_pred, axis=1)

# Stop timer
end_time = time.time()

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("CNN Model Summary:")
model.summary()
# Print the training hyperparameters
print("\nTraining Hyperparameters:")
print("Optimizer:", model.optimizer.get_config())
print("Loss:", model.loss)
print("Metrics:", model.metrics)

# Print the accuracy and time taken
print("\nCNN Accuracy:", test_accuracy)
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")
```

Figure 23: CNN for Image

```
Epoch 1/10
386/386 [==============================] - 37s 94ms/step - loss: 1.3976 - accuracy: 0.4217 - val_loss: 1.2938 - val_accuracy:
0.4775
Epoch 2/10
386/386 [==============================] - 35s 90ms/step - loss: 1.1923 - accuracy: 0.5282 - val_loss: 1.1794 - val_accuracy:
0.5300
Epoch 3/10
386/386 [==============================] - 35s 91ms/step - loss: 1.0772 - accuracy: 0.5743 - val_loss: 1.1701 - val_accuracy:
0.5423
Epoch 4/10
386/386 [==============================] - 35s 90ms/step - loss: 0.9886 - accuracy: 0.6142 - val_loss: 1.1390 - val_accuracy:
0.5579
Epoch 5/10
386/386 [==============================] - 35s 91ms/step - loss: 0.8916 - accuracy: 0.6569 - val_loss: 1.1442 - val_accuracy:
0.5605
Epoch 6/10
386/386 [==============================] - 35s 91ms/step - loss: 0.7949 - accuracy: 0.6983 - val_loss: 1.1624 - val_accuracy:
0.5533
Epoch 7/10
386/386 [==============================] - 35s 92ms/step - loss: 0.6753 - accuracy: 0.7458 - val_loss: 1.2064 - val_accuracy:
0.5608
Epoch 8/10
386/386 [==============================] - 38s 98ms/step - loss: 0.5520 - accuracy: 0.7963 - val_loss: 1.3354 - val_accuracy:
0.5630
Epoch 9/10
386/386 [==============================] - 37s 95ms/step - loss: 0.4270 - accuracy: 0.8489 - val_loss: 1.3773 - val_accuracy:
0.5549
Epoch 10/10
386/386 [==============================] - 36s 93ms/step - loss: 0.3128 - accuracy: 0.8932 - val_loss: 1.6393 - val_accuracy:
0.5536
121/121 [==============================] - 2s 16ms/step - loss: 1.5911 - accuracy: 0.5760
121/121 [==============================] - 2s 14ms/step
```

Figure 24: CNN Accuracy for Image

```python
# Create a new model that takes the same inputs as your original model but outputs from an intermediate layer
feature_extraction_model = Model(inputs=model.input, outputs=model.layers[-3].output)
```

```python
# Initialize a dictionary to store the extracted features for each emotion
extracted_text_features = {}

# Process and store features for each emotion
emotions = ['Angry', 'Fear', 'Happy', 'Sad', 'Surprise']
for emotion in emotions:
    emotion_data = data[data['Emotion'] == emotion]
    emotion_sequences = tokenizer.texts_to_sequences(emotion_data['cleanText'])
    emotion_padded = pad_sequences(emotion_sequences, maxlen=max_length, padding='post', truncating='post')
    emotion_features = feature_extraction_model.predict(emotion_padded)

    # Store the extracted features in the dictionary
    extracted_text_features[emotion] = emotion_features

# Now, extracted_text_features dictionary contains the features for each emotion
```

```
71/71 [==============================] - 1s 7ms/step
164/164 [==============================] - 1s 7ms/step
615/615 [==============================] - 4s 7ms/step
240/240 [==============================] - 2s 7ms/step
163/163 [==============================] - 1s 7ms/step
```

```python
all_features_df = pd.DataFrame()

for emotion, features in extracted_text_features.items():
    temp_df = pd.DataFrame(features)
    temp_df['Emotion'] = emotion  # Add a column for emotion
    all_features_df = pd.concat([all_features_df, temp_df])

# Save combined features to CSV
all_features_df.to_csv('FeaturesofTextN.csv', index=False)
```

Figure 25: Features Extraction of Texts

```
[58]: feature_extraction_model_Img = Model(inputs=model.input,
                                         outputs=model.layers[-2].output)
```

```
[59]: extracted_emotion_features = {}  # Dictionary to store features for each emotion

for emotion in np.unique(y_train):
    # Create a mask for the current emotion
    emotion_mask = [label == emotion for label in y_train]

    # Use the mask to filter images
    emotion_images = X_train[emotion_mask]

    # Check if the filtered images array is not empty
    if len(emotion_images) > 0:
        # Extract features and store them in the dictionary
        extracted_emotion_features[emotion] = feature_extraction_model_Img.predict(emotion_images)
    else:
        print(f"No images found for emotion: {emotion}")

# Now, extracted_emotion_features dictionary contains the features for each emotion
```

```
52/52 [==============================] - 1s 17ms/step
77/77 [==============================] - 1s 16ms/step
154/154 [==============================] - 3s 17ms/step
121/121 [==============================] - 2s 15ms/step
80/80 [==============================] - 1s 16ms/step
```

```
[60]: all_image_features_df = pd.DataFrame()

for emotion, features in extracted_emotion_features.items():
    temp_df = pd.DataFrame(features)
    temp_df['Emotion'] = emotion  # Add a column for emotion
    all_image_features_df = pd.concat([all_image_features_df, temp_df])

# Save combined features to CSV
all_image_features_df.to_csv('extracted_featuresNewCNN.csv', index=False)
```

Figure 26: Features Extraction of Images

	0	1	2	3	4	5	6	7	8	9	...	119	120	121	122	123	124	125	
0	2.951501	0.0	0.006768	0.703162	1.009076	0.210738	0.023643	0.0	0.618311	0.013353	...	0.0	0.0	0.000000	0.140160	0.204524	0.000000	0.06694	0.57
1	3.041102	0.0	0.006768	0.957298	1.536889	0.210738	0.023643	0.0	0.473427	0.124539	...	0.0	0.0	0.000000	1.846210	0.204524	0.147605	0.06694	0.08
2	3.540152	0.0	0.346147	0.699517	1.764352	0.210738	0.023643	0.0	0.319919	0.013353	...	0.0	0.0	0.563913	0.835976	0.835269	0.000000	0.06694	0.53
3	2.643020	0.0	0.197591	0.952973	2.552555	0.210738	0.324206	0.0	0.328923	2.732874	...	0.0	0.0	0.604440	0.140160	0.204524	0.569066	0.06694	0.30
4	2.504540	0.0	0.006768	0.774712	1.267292	0.210738	0.992287	0.0	0.280515	0.786413	...	0.0	0.0	0.000000	1.230530	0.204524	0.000000	0.06694	0.08

5 rows × 129 columns

```
image_features = r"C:\Users\Kamran Habib\Downloads\extracted_featuresNewCNNNew.csv"
df_image_features = pd.read_csv(image_features)
df_image_features.head()
```

	0	1	2	3	4	5	6	7	8	9	...	119	120	121	122	123	124	125	126	127	Emotion
0	0.0	4.640596	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	1.976921	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	angry
1	0.0	0.327910	0.0	0.0	0.0	0.0	0.0	2.815834	0.0	0.0	...	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	angry
2	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.623440	0.0	0.0	...	0.000000	2.118546	0.0	0.0	0.0	0.0	0.0	0.0	1.886850	angry
3	0.0	0.632937	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	2.098182	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	3.559995	angry
4	0.0	3.285811	0.0	0.0	0.0	0.0	0.0	0.650035	0.0	0.0	...	0.000000	0.253992	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	angry

5 rows × 129 columns

Figure 27: Loading text and Image Features

```
In [118]: # Standardize emotion labels (if necessary, for case-insensitivity)
df_text_features['Emotion'] = df_text_features['Emotion'].str.lower()
df_image_features['Emotion'] = df_image_features['Emotion'].str.lower()

# Find the minimum count for each emotion across both datasets
min_counts = {
    emotion: min(df_text_features['Emotion'].value_counts().get(emotion, float('inf')),
                 df_image_features['Emotion'].value_counts().get(emotion, float('inf')))
    for emotion in set(df_text_features['Emotion']).union(set(df_image_features['Emotion']))
}

# Downsample each emotion category in text data
balanced_text_data = pd.concat([
    df_text_features[df_text_features['Emotion'] == emotion].sample(n=min_count, random_state=42)
    for emotion, min_count in min_counts.items()
    if emotion in df_text_features['Emotion'].values
]).reset_index(drop=True)

# Downsample each emotion category in image data
balanced_image_data = pd.concat([
    df_image_features[df_image_features['Emotion'] == emotion].sample(n=min_count, random_state=42)
    for emotion, min_count in min_counts.items()
    if emotion in df_image_features['Emotion'].values
]).reset_index(drop=True)
```

```
In [119]: # Count the number of each emotion in the balanced text dataset
balanced_text_emotion_counts = balanced_text_data['Emotion'].value_counts()
print("Balanced Text Data Emotion Counts:")
print(balanced_text_emotion_counts)

# Count the number of each emotion in the balanced image dataset
balanced_image_emotion_counts = balanced_image_data['Emotion'].value_counts()
print("\nBalanced Image Data Emotion Counts:")
print(balanced_image_emotion_counts)
```

```
Balanced Text Data Emotion Counts:
happy       4909
sad         3872
surprise    2558
fear        2447
angry       1638
Name: Emotion, dtype: int64

Balanced Image Data Emotion Counts:
happy       4909
sad         3872
surprise    2558
fear        2447
angry       1638
```

Figure 28: Balancing text and image features with respect to each other

```
[120]: # Ensure both datasets have the same number of rows and are aligned
if len(balanced_text_data) == len(balanced_image_data):
    # Combine the features
    combined_features = pd.concat([balanced_text_data, balanced_image_data.drop('Emotion', axis=1)], axis=1)

    # Reorder columns to move 'Emotion' to the first position
    emotion_column = combined_features.pop('Emotion')  # Remove the 'Emotion' column and store it
    combined_features.insert(0, 'Emotion', emotion_column)  # Insert 'Emotion' column at the first position
else:
    print("The datasets are not aligned. They have different numbers of rows.")
```

Figure 29: Combining Text and Image Features

```python
# Separate features and target variable
X = combined_features.drop('Emotion', axis=1)
y = combined_features['Emotion']

# Encoding the target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)
```

Figure 30: Preparation on Combined Features

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("Random Forest Accuracy:", rf_accuracy)
print("Classification Report:\n", classification_report(y_test, rf_predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, rf_predictions))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Get all parameters of the Random Forest model
parameters = rf_classifier.get_params()

# Print the parameters
print("Random Forest Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
Random Forest Accuracy: 0.9737439222042139
```

Figure 31: Random Forest on Combined Features

```
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the KNN Classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("KNN Accuracy:", knn_accuracy)
print("Classification Report:\n", classification_report(y_test, knn_predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, knn_predictions))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Get all parameters of the KNN model
parameters = knn_classifier.get_params()

# Print the parameters
print("KNN Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
KNN Accuracy: 0.9711507293354943
```

Figure 32: KNN on Combined Features

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the Naive Bayes Classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
nb_predictions = nb_classifier.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("Naive Bayes Accuracy:", nb_accuracy)
print("Classification Report:\n", classification_report(y_test, nb_predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, nb_predictions))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")
print("\nGaussian Naive Bayes Model Parameters:", nb_classifier.get_params())
```

```
Naive Bayes Accuracy: 0.8693679092382496
```

Figure 33: Naive Bayes on Combined Features

```python
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Train the SVM Classifier
svm_classifier = SVC(kernel='linear', class_weight='balanced')  # You can change the kernel and other hyperparameters
svm_classifier.fit(X_train, y_train)

# Stop timer
end_time = time.time()

# Make predictions and evaluate the model
svm_predictions = svm_classifier.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print("SVM Accuracy:", svm_accuracy)
print("Classification Report:\n", classification_report(y_test, svm_predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, svm_predictions))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Get all parameters of the SVM model
parameters = svm_classifier.get_params()

# Print the parameters
print("SVM Model Parameters:")
for param, value in parameters.items():
    print(f"{param}: {value}")
```

```
SVM Accuracy: 0.9698541329011345
Classification Report:
```

Figure 34: SVM on Combined Features

```
initial_mem_usage = memory_usage()[0]

# Start timer
start_time = time.time()

# Define and compile the model
num_classes = len(data['Emotion'].unique())  # Number of unique emotion labels
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Stop timer
end_time = time.time()

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)

# Make predictions
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Calculate and print peak memory usage
peak_mem_usage = max(memory_usage()) - initial_mem_usage

# Print the results
print('Test accuracy:', test_accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred_classes))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_classes))
print("Time taken: {:.2f} seconds".format(end_time - start_time))
print(f"Peak memory usage: {peak_mem_usage} MiB")

# Print the model summary
print("\nModel Summary:")
model.summary()
```

Figure 35: ANN on Combined Features

```
Epoch 1/10
309/309 [==============================] - 1s 2ms/step - loss: 0.4433 - accuracy: 0.8608 - val_loss: 0.0841 - val_accuracy: 0.9
757
Epoch 2/10
309/309 [==============================] - 0s 2ms/step - loss: 0.1561 - accuracy: 0.9585 - val_loss: 0.0792 - val_accuracy: 0.9
785
Epoch 3/10
309/309 [==============================] - 0s 1ms/step - loss: 0.1269 - accuracy: 0.9685 - val_loss: 0.0765 - val_accuracy: 0.9
822
Epoch 4/10
309/309 [==============================] - 0s 2ms/step - loss: 0.1136 - accuracy: 0.9701 - val_loss: 0.0646 - val_accuracy: 0.9
810
Epoch 5/10
309/309 [==============================] - 0s 2ms/step - loss: 0.1067 - accuracy: 0.9730 - val_loss: 0.0669 - val_accuracy: 0.9
814
Epoch 6/10
309/309 [==============================] - 1s 2ms/step - loss: 0.0932 - accuracy: 0.9752 - val_loss: 0.0656 - val_accuracy: 0.9
814
Epoch 7/10
309/309 [==============================] - 0s 1ms/step - loss: 0.0885 - accuracy: 0.9743 - val_loss: 0.0656 - val_accuracy: 0.9
822
Epoch 8/10
309/309 [==============================] - 0s 1ms/step - loss: 0.0817 - accuracy: 0.9784 - val_loss: 0.0732 - val_accuracy: 0.9
806
Epoch 9/10
309/309 [==============================] - 0s 1ms/step - loss: 0.0691 - accuracy: 0.9810 - val_loss: 0.0716 - val_accuracy: 0.9
826
Epoch 10/10
309/309 [==============================] - 0s 1ms/step - loss: 0.0681 - accuracy: 0.9803 - val_loss: 0.0713 - val_accuracy: 0.9
801
97/97 - 0s - loss: 0.0833 - accuracy: 0.9818 - 88ms/epoch - 910us/step
97/97 [==============================] - 0s 688us/step
Test accuracy: 0.9818476438522339
```

Figure 36: ANN Accuracy for Combined Features of Texts and Images