National College of Ireland

Configuration Manual

MSc. Research Project
MSc. in Data Analytics

Ramam Rajdev
Student ID: X22237216

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Ramam Rajdev……

Student ID: …… x22237216……..……

Programme:…… MSc. in Data Analytics …… **Year:** …2023-24…..

Module: …… MSc. Research Project ……..……

Lecturer: …… Mr. Vikas Tomer ……..……

Submission Due Date: ……16/09/2024……..……

Project Title: ……Music Recommendation System Based On The Analysis Of The Images……..……

Word Count: ……943…… **Page Count:** ……10……..……..……

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: …… Ramam Rajdev ……

Date: ……16/09/2024……

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

Ramam Rajdev
Student ID: x22237216

1 Introduction

The following configuration manual aims at outlining the processes that have to be followed in order to create a duplicate of the music recommendation system that was designed given the mood detection. The project has used image processing, machine learning and deep learning for recommending musicals in accordance with the mood of the user. The manual divided into sections such as environment, data capturing, data analysis, model deployment, and the evaluation part.

2 Environment

2.1 Hardware Requirements

The project is implemented using the Google Colab platform, which provides the following hardware specifications:

- **Processor:** Intel Xeon CPU @ 2.20 GHz
- **RAM:** 13 GB
- **GPU:** Tesla K80 with 12 GB GDDR5 RAM or Cloud TPU with 180 teraflops computational power

2.2 Software Requirements

To execute the code, ensure you have access to the following software:

- **Web Browser:** Google Chrome (version 100+), Microsoft Edge (version 100+), Mozilla Firefox (version 100+)
- **Python Libraries:**
    - os, cv2, shutil, warnings, numpy, pandas, PIL, tqdm, matplotlib, keras, tensorflow, sklearn

Figure 1 below depicts the importing of important libraries for the study.

```
import os
import cv2
import shutil
import warnings
import numpy as np
import pandas as pd
from PIL import Image
from tqdm import tqdm
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from tensorflow.keras import regularizers
from tensorflow.keras.utils import get_file
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.resnet_v2 import ResNet50V2
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.layers import Flatten, Dense, Dropout, GlobalAveragePooling2D, BatchNormalization,Conv2D,MaxPooling2D

warnings.filterwarnings('ignore')
```

Figure 1: Necessary libraries for the implementation

3 Data Collection

The dataset used in this project is hosted on Kaggle and can be accessed at the following locations:

- **Emotion Detection Dataset**: https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer

Figure 2 below shows the contents of the test dataset.

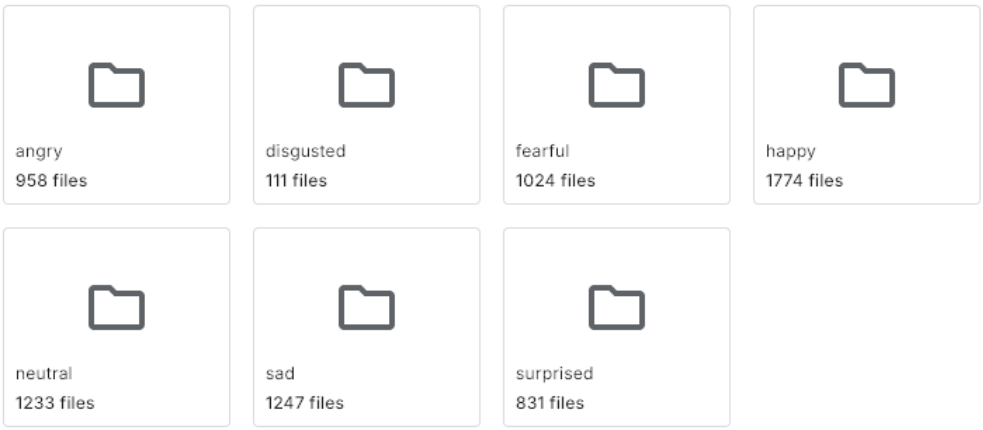

Figure 2: Directories in Test Data Folder

- **Spotify Music Dataset:** https://www.kaggle.com/datasets/musicblogger/spotify-music-data-to-identify-the-moods

Sample contents of the dataset obtained from the Kaggle website is shown in Figure 3 below.

name	album	artist	id	release_date	#
686 unique values	661 unique values	Various Artists 1% Wilson Trouvé 1% Other (671) 98%	686 unique values	2020-05-01 1% 2020-04-24 1% Other (670) 98%	0
1999	1999	Prince	2H7PHVdQ3mXqEHXcvclTB0	1982-10-27	6
23	23	Blonde Redhead	4HIwL9ii9CcXpTOTzMq0MP	2007-04-16	4
9 Crimes	9	Damien Rice	5GZEeowhvSieFDiR8fQ2im	2006-11-06	6
99 Luftballons	99 Luftballons	Nena	6HA97v4wEGQ5TUClRM0XLc	1984-08-21	2
A Boy Brushed Red Living In Black And White	They're Only Chasing Safety	Underoath	47IWLfIKOKhFnz1FUEUIkE	2004-01-01	6

Figure 3: Data Snippet of Spotify Music Dataset

Upload these datasets to the Google Colab environment to ensure easy access during execution.

4 Data Preparation

4.1 Data Exploration

The emotion detection dataset is structured into train and test directories, each containing images categorized by emotions (angry, happy, neutral, etc.). Use Python libraries like os and shutil to organise these images into appropriate directories for processing.

```
dataset_path = 'image-data/'
folders = os.listdir(dataset_path)
folders
```

```
['test', 'train']
```

```
for folder in folders:
    print(folder, len(os.listdir(dataset_path + '/' + folder)))
```

```
test 7
train 7
```

```
# Define the paths to the dataset directories
train_dir = 'image-data/train/'
test_dir = 'image-data/test/'

# Define emotions
emotions = ['angry', 'disgusted', 'fearful', 'happy', 'neutral', 'sad', 'surprised']
```

Figure 4: Data Exploration

Figure 4 above shows the code for the data exploration part of the study.

4.2 Data Preprocessing

Balancing the Dataset: Use the script to distribute images evenly across different emotions. The data is then split into training and test sets.
Figure 5 below shows the Python code for reading and balancing the dataset.

```
def generateData(emo):
    lst = os.listdir(train_dir + '/' + emo)
    for i in range(436):
        if(i<=(len(lst)-len(lst)*.2)):
            destination=os.path.join(train_path, emo)
        else:
            destination=os.path.join(test_path, emo)
        img = train_dir + '/' + emo + '/' + lst[i]
        shutil.copy(img, destination)
```

```
for emo in tqdm(emotions):
    generateData(emo)
```

```
100%|██████████| 7/7 [00:02<00:00,  2.35it/s]
```

Data Balancing

```
train_path = 'data/train'
test_path = 'data/test'
```

```
os.makedirs(train_path,exist_ok=True)
os.makedirs(test_path,exist_ok=True)
```

```
try:
    for emo in emotions:
        path = os.path.join(train_path, emo)
        os.makedirs(path)
        path = os.path.join(test_path, emo)
        os.makedirs(path)
    print("Folders created")
except:
    print("Folders already created")
```

```
Folders already created
```

Figure 5: Code snippet for Data Balancing

Image Resizing and Normalization: Images are resized to (224, 224) and normalized using ImageDataGenerator from Keras. Figure 6 below shows the data loading through scaling and resizing.

Balanced Data Loading

```
train_dir = 'data/train/'
test_dir = 'data/test/'
```

```
# Create batches from the train and test directories
image_size = (224, 224)

datagen = ImageDataGenerator(rescale=1/255.)
train_generator = datagen.flow_from_directory(train_dir, target_size=image_size)
test_generator = datagen.flow_from_directory(test_dir, target_size=image_size)
```

```
Found 2965 images belonging to 7 classes.
Found 87 images belonging to 7 classes.
```

Figure 6: Resizing and Normalisation using the ImageDataGenerator

4.3 Data Visualisation

Data is visualised before modelling to ensure correctness of the variable names as well as the emotions. Figure 7 depicts the code snippet to visualise the dataset contents.

```
# Get a batch of images and labels from the generator
x_batch, y_batch = next(train_generator)

# Get the emotion labels from the generator class indices
emotion_labels = list(train_generator.class_indices.keys())

# Plot the images and labels
fig, axes = plt.subplots(4, 8, figsize=(15, 8))
axes = axes.ravel()

for i in np.arange(0, 4*8):
    axes[i].imshow(x_batch[i])
    axes[i].axis('off')
    emotion_idx = np.argmax(y_batch[i])
    axes[i].set_title(emotion_labels[emotion_idx])

plt.subplots_adjust(wspace=0.5, hspace=0.5)
plt.show()
```

Figure 7: Code snippet to visualize dataset contents

5 Model Implementation

5.1 Pre-trained Models

The project uses several pre-trained models available in Keras

5.1.1 VGG16

Fine-tuned for emotion classification. Figure 8 below shows the implementation of the VGG16 model for the presented study.

VGG16

```
# Load the pre-trained VGG16 model
vgg16 = VGG16(input_shape=(224, 224, 3), weights='imagenet', include_top=False)
# Freeze the layers of the pre-trained VGG16 model
for layer in vgg16.layers:
    layer.trainable = True
    # Add additional layers to the model
x = Flatten()(vgg16.output)
x = Dense(1024, activation='relu')(x)
x = Dense(len(emotions), activation='relu')(x)
model = Model(vgg16.input, x)
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Figure 8: VGG 16 implementation

```
# Create Early Stopping Callback to monitor the accuracy
Early_Stopping = EarlyStopping(monitor = 'val_accuracy', patience = 3, restore_best_weights = True, verbose
# Create ReduceLROnPlateau Callback to reduce overfitting by decreasing learning rate
Reducing_LR = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, verbose=1)
callbacks = [Early_Stopping, Reducing_LR]
```

Figure 9: Code snippet for creating early stopping callbacks to avoid overfitting

```
# Train the model
history = model.fit(train_generator, validation_data= test_generator, epochs=50, callbacks=callbacks)
```

Figure 10: Fitting the VGG16 model with 50 epochs and callbacks

5.1.2 VGG19

Figure 11 below shows the implementation of the VGG19 model in the study.

```
# Load the pre-trained VGG16 model
base = VGG19(input_shape=(224, 224, 3), weights='imagenet', include_top=False)
# Freeze the layers of the pre-trained VGG16 model except the last block
for layer in base.layers[:-4]:
    layer.trainable = False
    # Add additional layers to the model
x = Flatten()(base.output)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)  # Add dropout regularization
x = Dense(512, activation='relu')(x)
x = Dropout(0.2)(x)  # Add another dropout regularization
x = Dense(len(emotions), activation='softmax')(x)
# Create the fine-tuned model
vgg19 = Model(inputs=base.input, outputs=x)
 # Compile the model
optimizer = Adam(learning_rate=1e-4)
vgg19.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
vgg19.summary()
```

Figure 11: VGG19 Implementation

```
# Train the model
history = vgg19.fit(train_generator, validation_data= test_generator, epochs=50, callbacks=callbacks)
```

Figure 12: Fitting the VGG19 model

5.1.3 ResNet50

A powerful CNN model used after freezing initial layers to improve accuracy. Figure 13 below shows its implementation.

```
base = ResNet50V2(input_shape=(224, 224, 3), weights='imagenet', include_top=False)
# Freeze the layers of the ResNet50 model:
for layer in base.layers:
    layer.trainable = False
    # Add additional layers to the model
x = Dropout(.25)(base.output)
x = BatchNormalization()(x)
x = Flatten()(x)
x = Dense(64, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(.5)(x)
x = Dense(len(emotions), activation='softmax')(x)
resnet = Model(base.input, x)
# Compile the model
optimizer = Adam(learning_rate=1e-4)
resnet.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
resnet.summary()
```

Figure 13: ResNet50 Implementation

```
# Train the model
history = resnet.fit(train_generator, validation_data= test_generator, epochs=50, callbacks=callbacks)
```

Figure 14: Fitting ResNet50 model

5.1.4 Custom CNN

A simple CNN model designed specifically for this task. The construction of the CNN model as a sequential network is shown in Figure 15.

```
cnn = Sequential()
cnn.add(Conv2D(32, (3,3), activation='relu', input_shape=(224, 224, 3)))
cnn.add(MaxPooling2D(pool_size=(2,2), padding='same'))
cnn.add(Dropout(0.25))
cnn.add(Flatten())
cnn.add(Dense(64, activation='relu'))
cnn.add(BatchNormalization())
cnn.add(Dense(len(emotions),activation='softmax'))
# Compile the model
optimizer = Adam(learning_rate=1e-4)
cnn.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
cnn.summary()
```

Figure 15: CNN Implementation

```
# Train the model
history = cnn.fit(train_generator, validation_data= test_generator, epochs=50, callbacks=callbacks)
```

Figure 16: Fitting CNN Model

Each model is trained using the training set, with early stopping and learning rate reduction applied to prevent overfitting.

5.2 Training and Evaluation

The models are trained for 50 epochs, and their performance is monitored using validation accuracy. The best model is selected based on the highest accuracy on the test set.

6 Music Recommendation System

The music recommendation system is built on the Spotify dataset. Key steps include:

6.1 Data Normalization

Features like danceability, acousticness, energy, etc., are normalized. Figure 17 below shows the python code to normalise the relevant features from the Spotify songs dataset.

```
# Normalize relevant features
features = ['danceability', 'acousticness', 'energy', 'valence', 'tempo']
data[features] = StandardScaler().fit_transform(data[features])
```

Figure 17: Normalisation of features using Standard Scaling

6.2 Mood Mapping

Predicted emotions are mapped to corresponding moods in the Spotify dataset. Figure 18 shows the mapping of the moods in the FER dataset to the ones in Spotify dataset.

```
# Mapping of input moods to dataset moods
mood_map = {
    'disgusted': 'Happy',
    'happy': 'Happy',
    'sad': 'Happy',
    'fearful': 'Calm',
    'angry': 'Calm',
    'surprised': 'Energetic',
    'neutral': 'Energetic'
}
```

Figure 18: Mapping the moods in the songs dataset with the ones in the FER dataset

6.3 KNN Model

The NearestNeighbors model from sklearn is used to find similar songs based on the normalized features. Figure 19 below shows the implementation of the KNN model and a recommendation system based on it.

```
# Function to train KNN model for a specific mood
def train_knn_for_mood(data, mood, n_neighbors=5):
    mood_data = data[data['mood'] == mood]
    knn = NearestNeighbors(n_neighbors=n_neighbors, metric='euclidean')
    knn.fit(mood_data[features])
    return knn, mood_data

# Function to recommend songs based on a predicted mood
def recommend_songs(data, predicted_mood, k=5):
    target_mood = mood_map.get(predicted_mood, 'Happy')
    knn, mood_data = train_knn_for_mood(data, target_mood, n_neighbors=k)

    # Get the closest songs
    distances, indices = knn.kneighbors(mood_data[features])
    recommended_songs = mood_data.iloc[indices.flatten()].drop_duplicates().head(k)
    return recommended_songs[['name', 'artist', 'mood', 'popularity']]
```

Figure 19: Training the KNN model to identify top K recommendation

6.4 Querying an image and getting the recommendation

Figure 20 below shows the function to detect the faces in the images given to the system.

```
def imageFaceDetect(filename, img_shape = 224):
    img = cv2.imread(filename)
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(img, 1.1, 4)
    for x,y,w,h in faces:
        roi_Img = img[ y: y + h , x: x + w ]
        roi_Img = img[ y: y + h , x: x + w ]
        cv2.rectangle(img, (x,y), (x+w, y+h), (0, 255, 0), 2)
        plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
        faces = faceCascade.detectMultiScale(roi_Img, 1.1, 4)
        if len(faces) == 0:
            print("No Faces Detected")
        else:
            for (ex, ey, ew, eh) in faces:
                img = roi_Img[ ey: ey+eh , ex: ex+ew ]
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    img= cv2.resize(img,(img_shape,img_shape))
    img = img/255.
    return img
```

Figure 20: Face Detection

Figure 21 below shows the code to detect the emotion for the query image.

```
def emotionPrediction(filename, class_names):
    # Import the target image and preprocess it
    img = imageFaceDetect(filename)
    # Make a prediction
    pred = resnet.predict(np.expand_dims(img, axis=0))
    # Get the predicted class
    pred_class = class_names[pred.argmax()]
    # Plot the image and predicted class
    plt.imshow(img)
    plt.title(f"Prediction: {pred_class}")
    plt.axis(False);
    return pred_class
```

Figure 21: Emotion Prediction

7 Model Evaluation and Results

After training, the accuracy of each model is compared. A bar plot visualizes the performance metrics. The best-performing model is then used to predict emotions on new images, and corresponding song recommendations are generated.

```
# Bar plot for accuracy Scores
plt.figure(figsize=(8, 6))
plt.bar(result['Models'], result['Scores'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.show()
```

Figure 22: Model Results

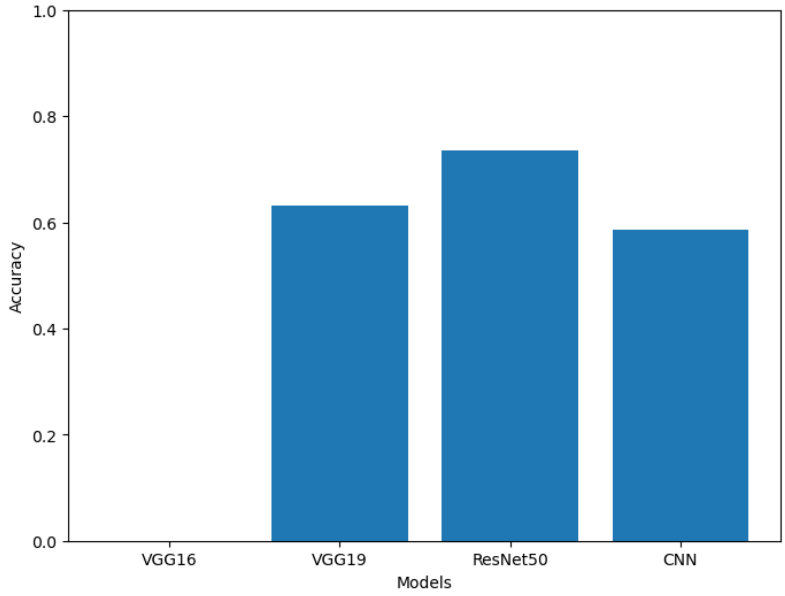

Figure 23: Modelling Results

```
pred_class = emotionPrediction("image-data/test/angry/im0.png", emotions) # with Resnet50
pred_class
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
'angry'
```

Figure 24: Querying Image for Emotion Prediction

```
recommendations = recommend_songs(data, pred_class, k=5)
display(recommendations)
```

	name	artist	mood	popularity
5	A Burden to Bear	Emmanuelle Rimbaud	Calm	27
636	While the Rest of the World Sleeps	Josh Kramer	Calm	35
28	Anaviosi	Alexi Musnitsky	Calm	38
43	Back+	Ton Snijders	Calm	26
418	Pierre	Hicham Chahidi	Calm	35

Figure 25: Song recommendations for the above prediction

```
pred_class = emotionPrediction("/kaggle/input/emotion-detection-fer/test/neutral/im1.png", emotions) # with ResNet50V2
pred_class
```

```
No Faces Detected
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step

'neutral'
```


```
recommendations = recommend_songs(data, pred_class, k=5)
display(recommendations)
```

	name	artist	mood	popularity
4	A Boy Brushed Red Living In Black And White	Underoath	Energetic	60
237	Holding On	Scary Kids Scaring Kids	Energetic	45
462	Same Direction	Hoobastank	Energetic	0
399	Out Of Control	Hoobastank	Energetic	0
592	Under A Killing Moon	Thrice	Energetic	41

Figure 26: Emotion Prediction with Song Recommendations

8 Conclusion

This manual is useful in replicating all the research explained in this study. The above steps should be repeated to recreate the environment, process the data, train all the necessary models, and integrate the music recommendation system.

References

https://www.tensorflow.org/api_docs
https://scikit-learn.org/0.21/documentation.html
https://docs.opencv.org/4.x/index.html
https://docs.python.org/3/library/os.html