National College of Ireland

Configuration Manual

MSc Research Project
MSc in Artificial Intelligence

Ayush Gole
Student ID:x23224100

School of Computing
National College of Ireland

Supervisor: Arundev Vamadevan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……. ……Ayush Gole……
Student ID:	……x23224100……
Programme:	…MSc in Artificial Intelligence…… **Year:** …2024-25……..
Module:	……MSc Research Project……
Lecturer:	…… Arundev Vamadevan……
Submission Due Date:	……12/12/24……
Project Title:	A Comparative Study of CNN, RNN-LSTM, and Transfer Learning Models for Facial Emotion Recognition in context of gaming
Word Count:	……967…… **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: ……A.V.Gole……

Date: ……12/12/24……

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

Ayush Gole
x23224100

1 Introduction

This configuration manual have all details about hardware and software specifications that has been used in this research process. The below sections shows steps that should be follow to setup running environment for this research study. It also contain different application that should configured and utilized.

2 System Specifications:

The system specification contains all the resources used in order to complete the research study. The Figure 1 shows local system used to run this project while Figure 2 (A)show all the run time utilized for research. Figure 2 (B) shows detailed information about memory with detailed specification limits. All hardware accelerators have been utilized for research and needs to be used with caution due to usage limit.

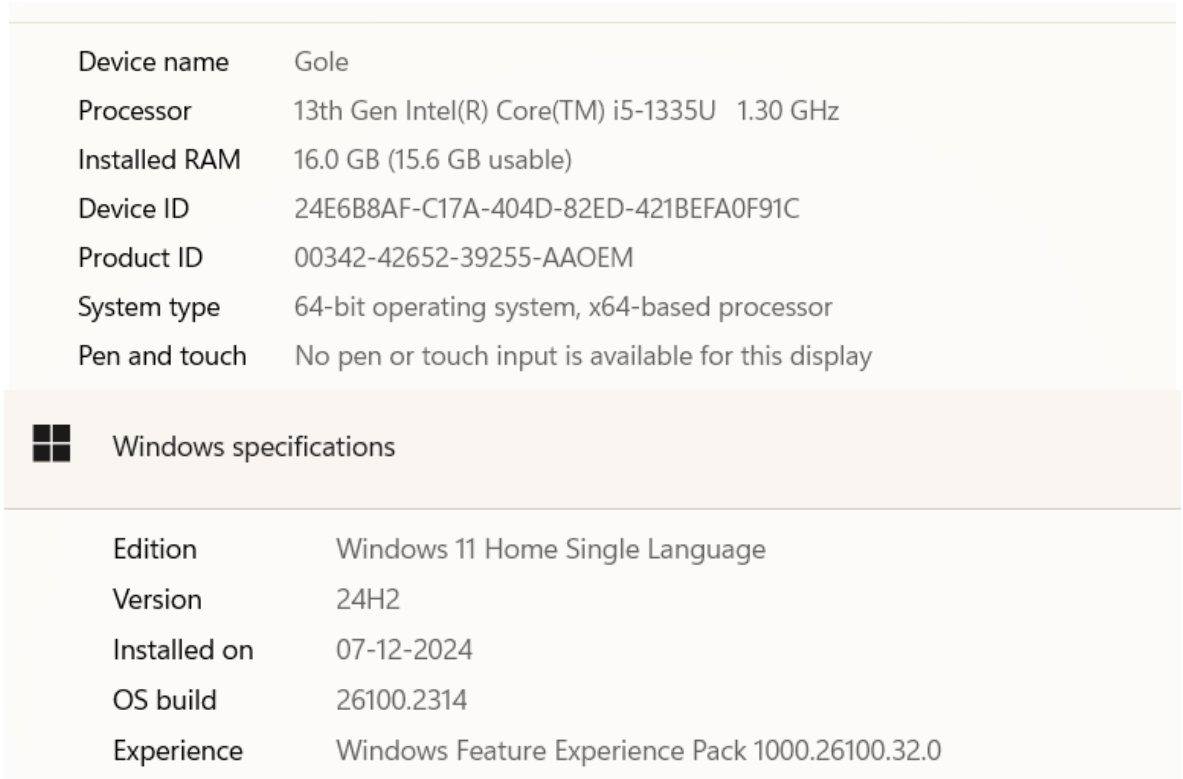

Figure 2. System specifications

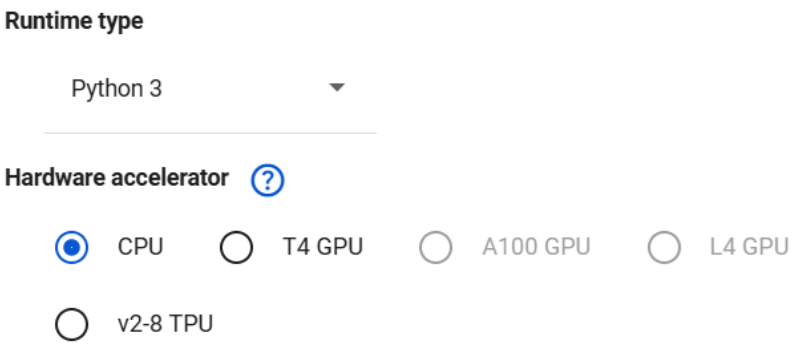

Figure 2 (A): Google Colab RunTime Types

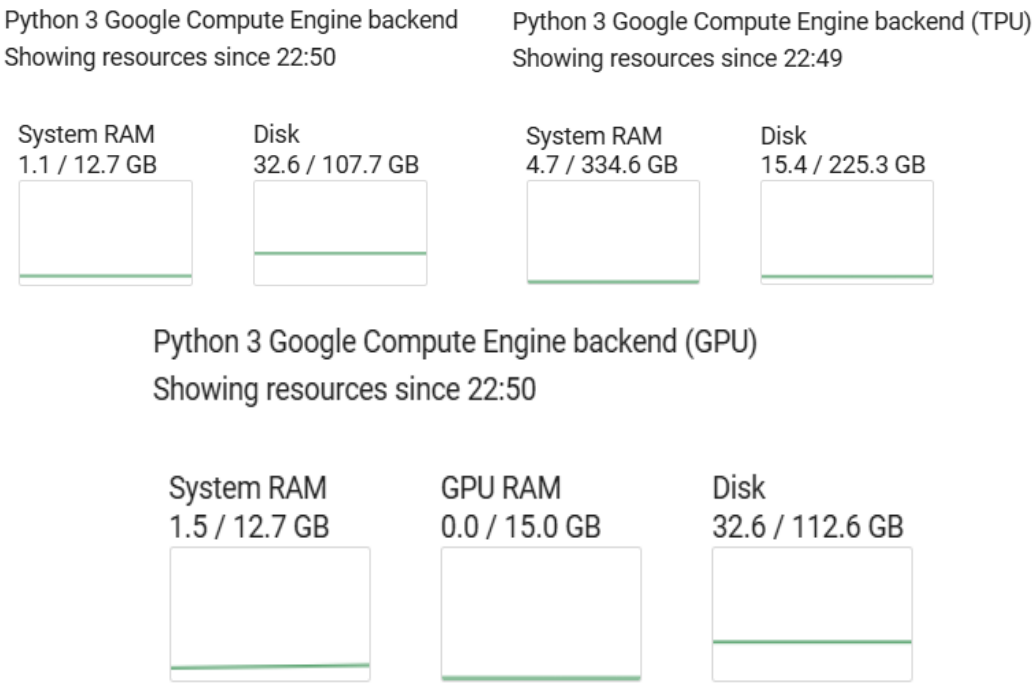

Figure 2 (B): Google Colab Resources

Software Used:

• Microsoft excel: Used for custom dataset description.
• Google Collab: Used for all processing and as code runtime environment
• Google Drive: Used to store models, datasets and handle while runtime

3 Dataset specification

This research study used 3 datasets:FER2013, CK , Custom dataset. It was introduced by (Lucey et al.; 2010) using CK dataset for facial emotion detection. The dataset consists of more than 950 face image data points. For research purpose we have used subset of CK dataset CK48 here as shown in Figure 3 (a). It is available to download at ck Dataset Custom dataset is created by author of this research. It contains total 245 images. Figure 3(b) shows custom dataset face image data and Figure 3 (c) shows Custom dataset game scene image dataset. This is not publicly available dataset but can be accessed from research resource or Drive link.

FER2013 is large dataset employed in this research. (Zahara et al.; 2020) introduced usage of dataset for facial emotion recognition. FER2013 dataset contains more than 35 thousand images and more than 28 thousand used for training purpose. Figure 3 (d) shows FER2013 dataset. The dataset is publicly available and can be downloaded from [FER2013 DATASET]

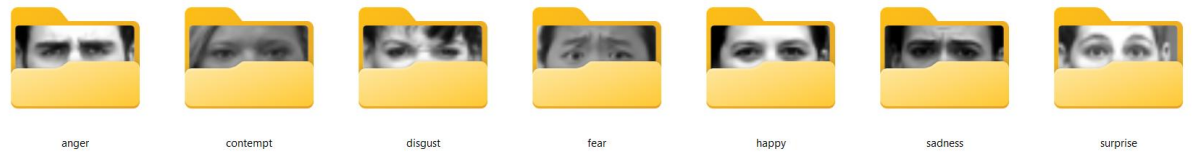

Figure 3 (a): Ck dataset

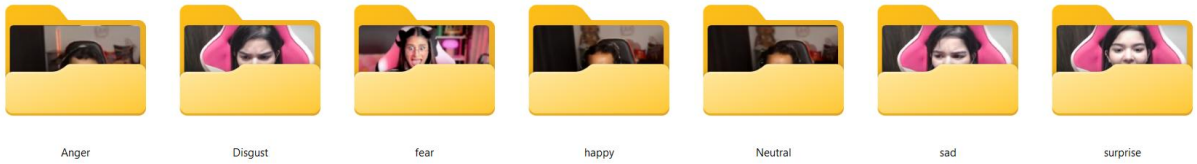

Figure 3 (b) : Custom dataset - face data

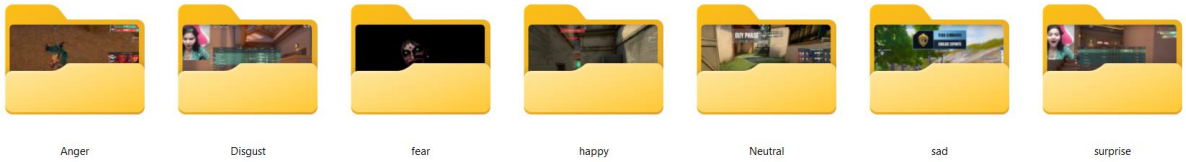

Figure 3 (c) : Custom dataset – game scene data

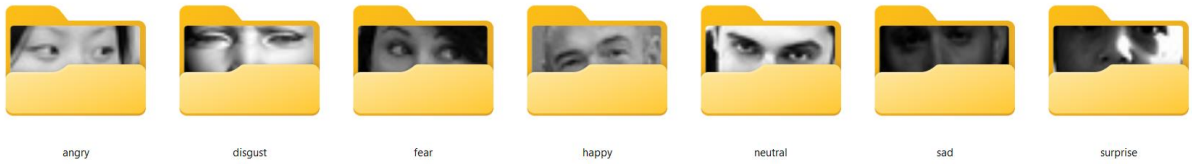

Figure 3 (d) : FER2013 dataset

4 Project Development

After collecting all data and storing it on drive or storage place, Colab notebook can be launched. Click on File , followed by Open notebook. As all the code files are stored in .ipynb file it can be opened easily and will contain all previous run results. You can mount drive or use notebook storage space for uploading and accessing dataset. There is option to run one by one or can be run simultaneously.

4.1 Importing Library:

All the required python libraries and packages are displayed in Figure 4. The colab platform comes with several preinstalled version and libraries. If necessary, please install required libraries in your environment. Thes libraries are freely available and can be easily installed from official python site.

```
#import all required libraries
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.layers import Input, Dense, Concatenate, Flatten
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import os
from keras.models import load_model
from keras.layers import Input
from tensorflow.keras.layers import Concatenate
#Import Required libraries
import tensorflow as tf
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM, Dense, Dropout, TimeDistributed, Flatten
from tensorflow.keras.preprocessing.sequence import pad_sequences
import cv2
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import ReduceLROnPlateau
import dlib
import matplotlib.pyplot as plt
```

Figure 4: Packages used in Project

4.2 Importing Files:

As Code environment is google colab, all dataset and files are stored in drive for easy access. Figure 5 shows importing process of Google Drive where data is stored and how files are accessed throughout research. As multiple code files used in this research models should be saved in desired storage space and should be imported. Figure 5 shows method loading of pretrained models in this project.

```
from google.colab import drive
drive.mount('/content/drive')
# Define data directory path
data_dir = '/content/drive/MyDrive/Untitled folder/Custom dataset/Game scne/Test4'
```

```
# Load datasets
dataset1_path = '/content/drive/MyDrive/Untitled folder/Custom dataset/Face/Preporcessed'
dataset3_path = '/content/drive/MyDrive/Untitled folder/CK+48/CK+48'
dataset3_path = '/content/drive/MyDrive/Untitled folder/FER2013/FER2013/train'
# Replace with the path to your root directory containing the emotion folders
root_dir = '/content/drive/MyDrive/Untitled folder/Custom dataset/Game scne/Train'
# Load the pre-trained models
face_emotion_model = load_model('/content/drive/MyDrive/Untitled folder/Custom dataset/Method 2 models/Model2gamescene.h5')
game_scene_emotion_model = load_model('/content/drive/MyDrive/Untitled folder/Custom dataset/Method 2 models/Model6custom_CK.h5')
```

Figure 5: Importing Files

4.3 Preprocessing and data validation:

Data validation is important part of exploration. Figure 6 shows code implementation for Data validation techniques used.

```
# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Load the shape predictor
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')

# Detect face landmarks
detector = dlib.get_frontal_face_detector()
dets = detector(gray, 1)
for k, d in enumerate(dets):
    shape = predictor(gray, d)

# Extract the coordinates of eyes, lips, and nose
eyes = [(shape.part(36).x, shape.part(36).y), (shape.part(45).x, shape.part(45).y)]
lips = [(shape.part(48).x, shape.part(48).y), (shape.part(54).x, shape.part(54).y)]
nose = [(shape.part(30).x, shape.part(30).y)]

# Plot the original image with landmarks
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
for x, y in eyes:
    plt.scatter(x, y, c='r')
    print(f"Pixel ({x}, {y})")
for x, y in lips:
    plt.scatter(x, y, c='g')
    print(f"Pixel ({x}, {y})")
for x, y in nose:
    plt.scatter(x, y, c='b')
    print(f"Pixel ({x}, {y})")
plt.show()
```

```
# Flip the image
flipped_img = cv2.flip(img, 1)

# Plot the flipped image with landmarks
plt.imshow(cv2.cvtColor(flipped_img, cv2.COLOR_BGR2RGB))
for x, y in eyes:
    plt.scatter(img.shape[1] - x, y, c='r')
    print(f"Pixel ({img.shape[1] - x}, {y})")
for x, y in lips:
    plt.scatter(img.shape[1] - x, y, c='g')
    print(f"Pixel ({img.shape[1] - x}, {y})")
for x, y in nose:
    plt.scatter(img.shape[1] - x, y, c='b')
    print(f"Pixel ({img.shape[1] - x}, {y})")
plt.show()

# Print the pixel coordinates where the dot is plotted but not in the image outside
for x, y in eyes + lips + nose:
    if x < 0 or x >= img.shape[1] or y < 0 or y >= img.shape[0]:
        print(f"Pixel ({x}, {y}) is outside the image")
```

Figure 6 : Data validation

Preprocessing: As custom datset contain different sizes and colored datapoints. Preprocessing techniques like resize and imread and data exported as shown in Figure 7 for both face and game scene image dataset.

```
# Function to preprocess an image
def preprocess_image(img_path):
  img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
  img = cv2.resize(img, (150, 100))
  return img

# Iterate through each emotion folder
for emotion_folder in os.listdir(root_dir):
    emotion_path = os.path.join(root_dir, emotion_folder)

    # Create output folder for the emotion
    output_emotion_path = os.path.join(output_dir, emotion_folder)
    os.makedirs(output_emotion_path, exist_ok=True)

    # Iterate through images in the emotion folder
    for image_file in os.listdir(emotion_path):
        image_path = os.path.join(emotion_path, image_file)

        # Preprocess the image
        preprocessed_img = preprocess_image(image_path)

        # Save the preprocessed image in the output folder
        output_file = os.path.join(output_emotion_path, image_file)
        cv2.imwrite(output_file, preprocessed_img)

print("Image preprocessing complete!")
```

```python
#define Preprocessing function
def preprocess_image(img_path, output_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (48, 48))
    cv2.imwrite(output_path, img)
```

Figure 7: Preprocessing

4.4 Modelling:

In this research 3 main methods used. All machine learning models are deep learning: CNN, RNN-LSTM and CNN transfer learning. As show in below implementations. For multi input CNN, make sure to train CNN models created with multi dataset combinations and save and later import them in order to combine them as shown below for final model part. In case of Transfer learning also similarly models should be saved and imported later as combination. For RNN-LSTM case, model features are saved and fed as input to LSTM models. All these implementations shown below.

A. Method 1: Multi-input CNN

Multi-input CNN model is implemented in research. CNN model used shown in Figure 8. For Mult input combination of CNN implemented shown in 11.

```python
print("Creating the CNN model...")
model3 = Sequential([
    Input(shape=(img_height, img_width, 1)),  # Use Input layer
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dense(len(le.classes_), activation='softmax')
])

# Compile the model
print("Compiling the model...")
model3.compile(optimizer='RMSprop', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model with a smaller batch size if necessary
print("Starting model training...")
from tensorflow.keras.callbacks import ReduceLROnPlateau

# adjusting learning rate
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-3)

#train model
history = model3.fit(
    train_datagen.flow(X_train, y_train, batch_size=16),  # Adjusted batch size
    epochs=20,
    validation_data=(X_test, y_test),
    callbacks=[reduce_lr]
)
```

Figure 8: CNN model

B. Method 2: Transfer Learning

Transfer learning methods used in this research are of two pretrained models VGG16 and RESNET CNN. Figure 9 shows CNN VGG16 pretrained model. Figure 10 shows CNN ResNet pretrained model. While this model is implemented in method 3 and method 1 its part of transfer learning so shown here.

```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))
from tensorflow.keras.layers import Dropout
# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Add custom top layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(8, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(7, activation='softmax')(x)  # Adjust the number of classes as needed

# Create the final model
model7 = Model(inputs=base_model.input, outputs=x)
model7.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
for _ in range(len(train_generator1) + len(train_generator4)):
    next(combined_generator)
model7.fit(combined_generator,
          steps_per_epoch=max(len(train_generator1), len(train_generator3)),
          epochs=5,
          validation_data=combined_generator,
          validation_steps=max(len(train_generator1), len(train_generator3)))
```

Figure 9 : Transfer learning VGG16

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))
from tensorflow.keras.layers import Dropout
# Freeze the base model layers
for layer in base_model.layers[-10:]:
    layer.trainable = False

# Add custom top layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(7, activation='softmax')(x)  # Adjusting the number of classes as per emotions

# Create the final model
model1 = Model(inputs=base_model.input, outputs=x)

# Compile the model
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
for _ in range(len(train_generator1) + len(train_generator2)):
    next(combined_generator)  # Consume data until exhaustion
# Train the model
model1.fit(combined_generator,
          steps_per_epoch=max(len(train_generator1), len(train_generator2)),
          epochs=5,
          validation_data=combined_generator,
          validation_steps=max(len(train_generator1), len(train_generator2)))
```

Figure 10 :Transfer learning RESNET 50

C. Method 3: RNN-LSTM

Figure 11 shows RNN-LSTM model implementation. RNN-LSTM is implemented along with transfer learning models which are same as show in figure 10. And features are saved for same.

```
# Assume game_features and face_features are your extracted features
game_features = features2  # Example shape
face_features = features  # Example shape

# Pad the facial features to match the game features sequence length
face_features_padded = pad_sequences(face_features, maxlen=5, dtype='float32', padding='post', truncating='post')
#Reshape the features to have the same sequence length
game_features_reshaped = game_features.reshape(game_features.shape[0], game_features.shape[1], -1)  # (193, 5, 4*2048)
face_features_reshaped = face_features_padded.reshape(face_features_padded.shape[0], face_features_padded.shape[1], -1)  # (981, 5, 2*2048)

# Align the number of samples
min_samples = min(game_features_reshaped.shape[0], face_features_reshaped.shape[0])

# Truncate both feature sets to the minimum sample size
game_features_aligned = game_features_reshaped[:min_samples]
face_features_aligned = face_features_reshaped[:min_samples]
#combine learnt features
combined_features = np.concatenate((game_features_aligned, face_features_aligned), axis=-1)
num_samples = combined_features.shape[0]  # Number of samples after truncation
num_classes = 7
labels = np.zeros(num_samples)
labels[num_samples // 2:] = 1
model4 = Sequential()
model4.add(LSTM(256, return_sequences=True, input_shape=(combined_features.shape[1], combined_features.shape[2])))
model4.add(Dropout(0.5))
model4.add(LSTM(128))
model4.add(Dropout(0.5))
model4.add(Dense(1, activation='sigmoid'))  # Adjust output layer based on task for future adjust according to future dataset classes

# Compile the model
model4.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model4.fit(combined_features, labels, epochs=5, batch_size=32)
```

Figure 11 : RNN LSTM model

D. Combining models:

```
# Define input layers
face_input = Input(shape=(48, 48, 1))
game_scene_input = Input(shape=(150, 100, 1))
# Get outputs from each model
face_output = face_model(face_input)
game_scene_output = game_scene_model(game_scene_input)
from tensorflow.keras.layers import Concatenate
# If the output is a list, access the first element
if isinstance(face_output, list):
    face_output = face_output[0]
if isinstance(game_scene_output, list):
    game_scene_output = game_scene_output[0]
num_classes_face = face_output.shape[-1]
num_classes_scene = game_scene_output.shape[-1]
if num_classes_face != num_classes_scene:
    # Create a Dense layer to match the number of classes
    game_scene_output = Dense(num_classes_face, activation='softmax')(game_scene_output)
# Concatenate the outputs
combined_output = Concatenate()([face_output, game_scene_output])
x = Dense(128, activation='relu')(combined_output)
x = Dense(64, activation='relu')(x)
NUM_CLASSES = 7
final_output = Dense(NUM_CLASSES, activation='softmax')(x)  # NUM_CLASSES is the total number of emotions

# Create the final model
final_model = Model(inputs=[face_input, game_scene_input], outputs=final_output)

# Compile the model
final_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print the summary of the model
final_model.summary()
```

Figure 11:

References

Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z. and Matthews, I. (2010). The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression, *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition- Workshops, pp. 94–101*

Zahara, L., Musa, P., Prasetyo Wibowo, E., Karim, I. and Bahri Musa, S. (2020). The facial emotion recognition (fer-2013) dataset for prediction system of micro-expressions face using the convolutional neural network (cnn) algorithm based raspberry pi, *2020 Fifth International Conference on Informatics and Computing (ICIC), pp. 1–9.*