Configuration Manual

MSc Research Project
Data Analytics

Karan Kohli
Student ID: x21179212

School of Computing
National College of Ireland

Supervisor: Abdul Shahid

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …Karan Kohli……………………..…………………………………………………………………

Student ID: …x21179212………………………………………………………………………………………..……

Programme: …Data Analytics…………………………………………… **Year:** ……2023…………..

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

Lecturer: …Abdul Shahid………………………………………………………………………………….………

Submission Due Date: …14-08-2023…………………………………………………………………………………….………

Project Title: …Neural Network-Based Detection of Disengagement in Virtual Environment…….……….

Word Count: …639……………………………… **Page Count:** …12……………………………….…….………

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: …Karan Kohli……………………………………………………………………………………

Date: …14th August 2023…….……………………………………………………………………

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

Karan Kohli
Student ID: x21179212

1 Introduction

This document contains the configuration manual used to develop the architecture of detecting student disengagement in virtual environment. Additionally, in the implementation section, the document has discussed the phases of code.

2 System Configuration

In the below sections, the work has discussed the hardware configuration and software setup which is used to develop the entire detection architecture.

2.1 System Configuration

For the development, the work has used Dell Inspiron 15 7570. The configuration of the system is: Operating system- Microsoft Window 10 Home Single Language, Processor- Intel Core i5, Ram- 16GB, GPU- Nvidia GeForce 940MX, SSD- 1 TB. Please refer the Figure 1 for more information.

Item	Value
OS Name	Microsoft Windows 10 Home Single Language
Version	10.0.19045 Build 19045
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	DESKTOP-18TA4NC
System Manufacturer	Dell Inc.
System Model	Inspiron 7570
System Type	x64-based PC
System SKU	07EA
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date	Dell Inc. 1.25.0, 7/13/2022
SMBIOS Version	3.0
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Dell Inc.
BaseBoard Product	0RP6XK
BaseBoard Version	A00
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\Windows
System Directory	C:\Windows\system32
Boot Device	\Device\HarddiskVolume4
Locale	United States
Hardware Abstraction Layer	Version = "10.0.19041.2728"
User Name	DESKTOP-18TA4NC\Dell
Time Zone	GMT Daylight Time
Installed Physical Memory (RAM)	16.0 GB
Total Physical Memory	15.8 GB
Available Physical Memory	3.41 GB

Figure 1: System Configuration

Due to less computational resources, the scripting of code made in the Google colab.

2.2 Software Configuration

The entire development was made on Google colab and the configuration manual as below:

- Google Colab- It is used as the primary GUI for creating proposed model architecture.
- Python- Python 3.8 version is used as a programming language.
- Libraries: Libraries used for the development are numpy, pandas, scikit-plot, matplotlib,scikit-learn, tensorflow, and seaboarn.

3 Implementation

3.1 Data Source

The dataset is available on Kaggle website. This is a publicly available dataset which was introduced by Kaggle for a competition held in 2013.

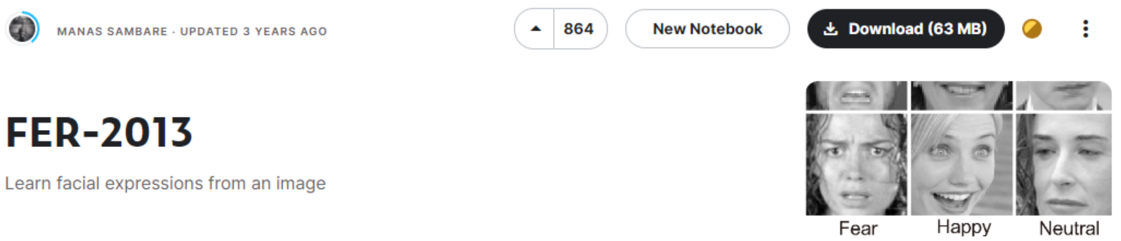


Figure 2: FER-2013 dataset

3.2 Importing required libraries

Importing required libraries

```
import math
import numpy as np
import pandas as pd

import scikitplot
import seaborn as sns
from matplotlib import pyplot

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report

import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dropout, BatchNormalization, LeakyReLU, Activation
from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from keras.utils import np_utils
```

3.3 Mapping of emotion labels to corresponding text label

Mapping of emotion labels to corresponding text labels

```
emotion_label_to_text = {
  0: 'anger',
  1: 'disgust',
  2: 'fear',
  3: 'happiness',
  4: 'sadness',
  5: 'surprise',
  6: 'neutral'
}
```

3.4 Relevant Data selection

```
# List of emotion labels of interest
INTERESTED_LABELS = [3, 4, 6]

# Filter the DataFrame to include only rows with specified emotion labels
df = df[df.emotion.isin(INTERESTED_LABELS)]

# Print the shape of the filtered DataFrame
df.shape # (21264, 3)

```

```
(21264, 3)
```

3.5 Preprocessing Images and Labels

```
# Preprocessing Images and Labels

# Convert the 'pixels' column of DataFrame into a 3D image array
img_array = df.pixels.apply(lambda x: np.array(x.split(' ')).reshape(48, 48, 1).astype('float32'))

# Stack the individual image arrays to create a 4D image array
img_array = np.stack(img_array, axis=0)

# Print the shape of the image array
print(img_array.shape)  # Output: (21264, 48, 48, 1)

# Create a LabelEncoder instance to encode emotion labels
le = LabelEncoder()

# Transform emotion labels into numerical encoded labels
img_labels = le.fit_transform(df.emotion)

# Convert numerical encoded labels into categorical format
img_labels = np_utils.to_categorical(img_labels)

# Print the shape of the label array in categorical format
print(img_labels.shape)  # Output: (21264, 3)

# Create a mapping of emotion labels to their encoded values
le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))

# Print the mapping of emotion labels to encoded values
print(le_name_mapping)  # Output: {3: 0, 4: 1, 6: 2}

```

3.6 Data Splitting

```
# Data Splitting and Memory Cleanup

# Split the data into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(
    img_array,
    img_labels,
    shuffle=True,
    stratify=img_labels,
    test_size=0.1,
    random_state=42
)

# Clean up memory by deleting unnecessary variables
del df
del img_array
del img_labels

# Print the shapes of the training and validation sets
print(X_train.shape, X_valid.shape, y_train.shape, y_valid.shape)

```

```
(19137, 48, 48, 1) (2127, 48, 48, 1) (19137, 3) (2127, 3)
```

3.7 Data Normalization

```
# Normalizing arrays, as neural networks are very sensitive to unnormalized data.
# Data Normalization and Dimension Calculations

# Normalize image arrays to range [0, 1] for neural network
X_train = X_train / 255.
X_valid = X_valid / 255.

# Calculate dimensions and number of classes
img_width = X_train.shape[1]      # Width of each image
img_height = X_train.shape[2]     # Height of each image
img_depth = X_train.shape[3]      # Depth (number of channels) of each image
num_classes = y_train.shape[1]    # Number of emotion classes

```

3.8 Import Metrics for Evaluation

```
# TensorFlow Imports for Optimization and Metrics

# Import the TensorFlow library
import tensorflow as tf

# If needed, import a specific optimizer from Keras
# from tensorflow.keras.optimizers import your_optimizer_here

# Import specific metrics from Keras for model evaluation
from tensorflow.keras.metrics import AUC, Precision, Recall

```

4 Model and Layers Deployment

4.1 Model building and Adding Layers

```
# Build Deep Convolutional Neural Network (DCNN)

def build_net(optim):
    # Create a Sequential model named 'DCNN'
    net = Sequential(name='DCNN')

    # Add first convolutional layer
    net.add(
        Conv2D(
            filters=64,
            kernel_size=(5,5),
            input_shape=(img_width, img_height, img_depth),
            activation='relu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_1'
        )
    )
    net.add(BatchNormalization(name='batchnorm_1'))

    # Add second convolutional layer
    net.add(
        Conv2D(
            filters=64,
            kernel_size=(5,5),
            activation='relu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_2'
        )
    )
    net.add(BatchNormalization(name='batchnorm_2'))
```

4.2 Adding Max Pooling and Dropout Layers

```
    # Add max pooling and dropout layers
    net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_1'))
    net.add(Dropout(0.4, name='dropout_1'))

    # Continue adding convolutional layers
    # ... (Repeat similar blocks for more layers)

    # Flatten the output for fully connected layers
    net.add(Flatten(name='flatten'))

    # Add dense layers
    net.add(
        Dense(
            128,
            activation='relu',
            kernel_initializer='he_normal',
            name='dense_1'
        )
    )
    net.add(BatchNormalization(name='batchnorm_7'))
```

4.3 Additing Additional Layer with Softmax activation

```
    # Add output layer with softmax activation
    net.add(
        Dense(
            num_classes,
            activation='softmax',
            name='out_layer'
        )
    )

    # Compile the model with specified loss, optimizer, and metrics
    net.compile(
        loss='categorical_crossentropy',
        optimizer=optim,
        metrics=[
            'accuracy',
            tf.keras.metrics.AUC(),
            tf.keras.metrics.Precision(),
            tf.keras.metrics.Recall()
        ]
    )

    # Print model summary
    net.summary()

    return net
```

4.4 Define Early Stopping and Learning Rate

```
# Define Early Stopping and Learning Rate Scheduling Callbacks

# Early Stopping callback to stop training when validation accuracy plateaus
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    min_delta=0.00005,
    patience=11,
    verbose=1,
    restore_best_weights=True,
)

# Learning Rate Reduction callback to adjust learning rate when improvement slows
lr_scheduler = ReduceLROnPlateau(
    monitor='val_accuracy',
    factor=0.5,
    patience=7,
    min_lr=1e-7,
    verbose=1,
)

# List of callbacks to be used during model training
callbacks = [
    early_stopping,
    lr_scheduler,
]
```

4.5 Data Augmentation Steps Followed

```
# Setting Up and Fitting Image Data Generator for Training Data Augmentation

# Create an ImageDataGenerator for training data augmentation
train_datagen = ImageDataGenerator(
    rotation_range=15,          # Random rotation up to 15 degrees
    width_shift_range=0.15,     # Random horizontal shift up to 15% of image width
    height_shift_range=0.15,    # Random vertical shift up to 15% of image height
    shear_range=0.15,           # Random shear transformation
    zoom_range=0.15,            # Random zoom in/out
    horizontal_flip=True,       # Randomly flip images horizontally
)

# Fit the data generator to the training data
train_datagen.fit(X_train)
```

4.6 Setting Up batch Size, Number of Epochs, and Optimizers

```
# Setting Up Batch Size, Number of Epochs, and Optimizers

# Batch size for training
batch_size = 32  # A batch size of 32 performs the best.

# Number of training epochs
epochs = 100

# List of optimizers to be tested
optims = [
    optimizers.Nadam(
        learning_rate=0.001,
        beta_1=0.9,
        beta_2=0.999,
        epsilon=1e-07,
        name='Nadam'
    ),
    optimizers.Adam(0.001),
]

```

4.7 Compiling and Fiting Model

```
# Compiling and Fitting the Model

# Import Keras backend module
import tensorflow.keras.backend as K

# Build the neural network model using the selected optimizer
model = build_net(optims[1])

# Fit the model using a generator for training data and validation data
history = model.fit_generator(
    train_datagen.flow(X_train, y_train, batch_size=batch_size),   # Training data generator
    validation_data=(X_valid, y_valid),                            # Validation data
    steps_per_epoch=len(X_train) / batch_size,                     # Steps per epoch
    epochs=epochs,                                                # Number of epochs
    callbacks=callbacks,                                          # Callbacks for training
    use_multiprocessing=True                                      # Use multiprocessing for data loading
)

```

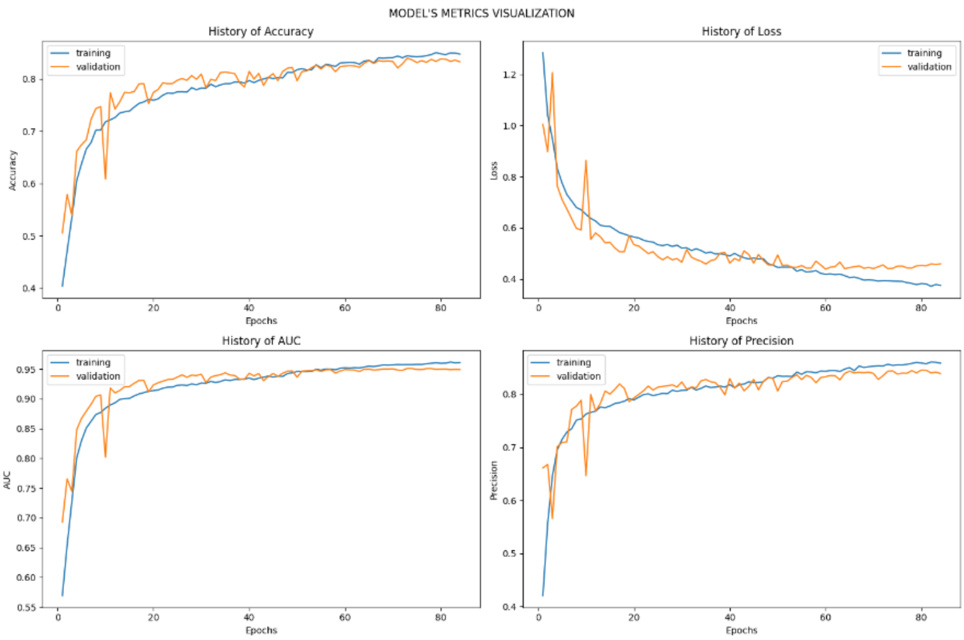
4.8 Save the Trained Model to a File for Deployment Use

```
# Save the Trained Model to a File

# Save the trained model to the specified file path
model.save("/content/drive/MyDrive/cnn_83_2.h5")

```

4.9 Fuctions used to Plot Training and Validate Metrics

```python
# Function to Plot Training and Validation Metrics

def Train_Val_Plot(acc, val_acc, loss, val_loss, auc, val_auc, precision, val_precision):
    # Create a 2x2 grid of subplots
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
    fig.suptitle("MODEL'S METRICS VISUALIZATION")

    # Plot history of accuracy
    ax1.plot(range(1, len(acc) + 1), acc)
    ax1.plot(range(1, len(val_acc) + 1), val_acc)
    ax1.set_title('History of Accuracy')
    ax1.set_xlabel('Epochs')
    ax1.set_ylabel('Accuracy')
    ax1.legend(['training', 'validation'])

    # Plot history of loss
    ax2.plot(range(1, len(loss) + 1), loss)
    ax2.plot(range(1, len(val_loss) + 1), val_loss)
    ax2.set_title('History of Loss')
    ax2.set_xlabel('Epochs')
    ax2.set_ylabel('Loss')
    ax2.legend(['training', 'validation'])

    # Plot history of AUC
    ax3.plot(range(1, len(auc) + 1), auc)
    ax3.plot(range(1, len(val_auc) + 1), val_auc)
    ax3.set_title('History of AUC')
    ax3.set_xlabel('Epochs')
    ax3.set_ylabel('AUC')
    ax3.legend(['training', 'validation'])

    # Plot history of Precision
    ax4.plot(range(1, len(precision) + 1), precision)
    ax4.plot(range(1, len(val_precision) + 1), val_precision)
    ax4.set_title('History of Precision')
    ax4.set_xlabel('Epochs')
    ax4.set_ylabel('Precision')
    ax4.legend(['training', 'validation'])

    # Adjust layout and display the plot
    plt.tight_layout()
    plt.show()

# Call the function with appropriate metric values
Train_Val_Plot(history.history['accuracy'], history.history['val_accuracy'],
               history.history['loss'], history.history['val_loss'],
               history.history['auc'], history.history['val_auc'],
               history.history['precision'], history.history['val_precision'])

```

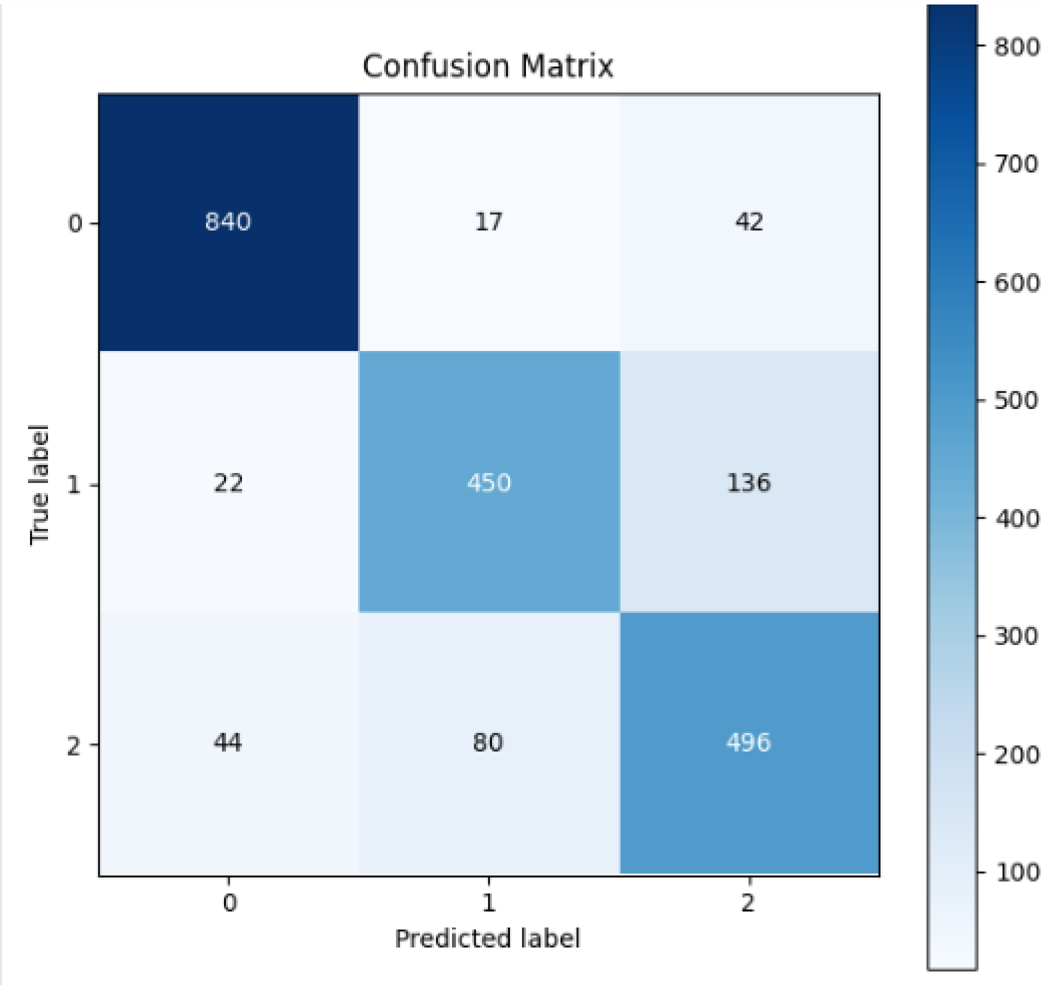
5 Results



5.1 Calculate and Visualize Confusion Matrix

```python
# Calculate and Visualize Confusion Matrix
yhat_valid_probs = model.predict(X_valid)  # Predicted probability scores for each class
yhat_valid_labels = np.argmax(yhat_valid_probs, axis=1)  # Convert probabilities to class labels

# Use scikit-plot to plot the confusion matrix
scikitplot.metrics.plot_confusion_matrix(np.argmax(y_valid, axis=1), yhat_valid_labels, figsize=(7, 7))
pyplot.savefig("confusion_matrix_dcnn.png")

# Calculate and print the total number of wrong validation predictions
print(f'total wrong validation predictions: {np.sum(np.argmax(y_valid, axis=1) != yhat_valid_labels)}\n\n')

# Print the classification report for evaluation metrics
print(classification_report(np.argmax(y_valid, axis=1), yhat_valid_labels))

```

5.2 Confusin Matrix



6 Visualization of Random Sad and Neutral Images with Predictions

```python
# Visualization of Random Sad and Neutral Images with Predictions

# Map class indices to labels
mapper = {
    0: "happy",
    1: "sad",
    2: "neutral",
}

# Set random seed for reproducibility
np.random.seed(2)

# Choose random sad and neutral images for visualization
random_sad_imgs = np.random.choice(np.where(y_valid[:, 1]==1)[0], size=9)
random_neutral_imgs = np.random.choice(np.where(y_valid[:, 2]==1)[0], size=9)

# Create a figure for visualization
fig = pyplot.figure(1, (18, 4))

# Loop through randomly selected images
for i, (sadidx, neuidx) in enumerate(zip(random_sad_imgs, random_neutral_imgs)):
        ax = pyplot.subplot(2, 9, i+1)
        sample_img = X_valid[sadidx,:,:,0]
        ax.imshow(sample_img, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        predicted_class = np.argmax(model.predict(sample_img.reshape(1, 48, 48, 1)))
        ax.set_title(f"true:sad, pred:{mapper[predicted_class]}")

        ax = pyplot.subplot(2, 9, i+10)
        sample_img = X_valid[neuidx,:,:,0]
        ax.imshow(sample_img, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        predicted_class = np.argmax(model.predict(sample_img.reshape(1, 48, 48, 1)))
        ax.set_title(f"t:neut, p:{mapper[predicted_class]}")

# Adjust layout and display the visualization
pyplot.tight_layout()
```

6.1 Result and Challenged Faced Visual Discription

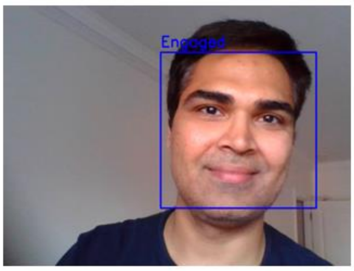
6.2 Model Architecture

```
Model: "DCNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 48, 48, 64)        1664

 batchnorm_1 (BatchNormaliza  (None, 48, 48, 64)       256
 tion)

 conv2d_2 (Conv2D)           (None, 48, 48, 64)        102464

 batchnorm_2 (BatchNormaliza  (None, 48, 48, 64)       256
 tion)

 maxpool2d_1 (MaxPooling2D)  (None, 24, 24, 64)        0

 dropout_1 (Dropout)         (None, 24, 24, 64)        0

 conv2d_3 (Conv2D)           (None, 24, 24, 128)       73856

 batchnorm_3 (BatchNormaliza  (None, 24, 24, 128)      512
 tion)

 conv2d_4 (Conv2D)           (None, 24, 24, 128)       147584

 batchnorm_4 (BatchNormaliza  (None, 24, 24, 128)      512
 tion)

 maxpool2d_2 (MaxPooling2D)  (None, 12, 12, 128)       0

 dropout_2 (Dropout)         (None, 12, 12, 128)       0

 conv2d_5 (Conv2D)           (None, 12, 12, 256)       295168

 batchnorm_5 (BatchNormaliza  (None, 12, 12, 256)      1024
 tion)

 conv2d_6 (Conv2D)           (None, 12, 12, 256)       590080

 batchnorm_6 (BatchNormaliza  (None, 12, 12, 256)      1024
 tion)

 maxpool2d_3 (MaxPooling2D)  (None, 6, 6, 256)         0

 dropout_3 (Dropout)         (None, 6, 6, 256)         0

 flatten (Flatten)           (None, 9216)              0

 dense_1 (Dense)             (None, 128)               1179776

 batchnorm_7 (BatchNormaliza  (None, 128)              512
 tion)

 dropout_4 (Dropout)         (None, 128)               0

 out_layer (Dense)           (None, 3)                 387

=================================================================
Total params: 2,395,075
Trainable params: 2,393,027
Non-trainable params: 2,048
```

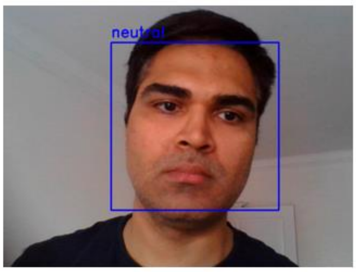
6.3 Webcam-based Engagement Detection System

```python
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator
#import matplotlib.pyplot as plt
from PIL import Image
import tensorflow as tf
from glob import glob
from sklearn.metrics import classification_report
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import img_to_array
import numpy as np

model = tf.keras.models.load_model(r'F:\NCI\Sem 3\Web\cnn_83_2.h5')
# Load the Haar Cascade for face detection
import cv2
face_haar_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(0)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    gray_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_haar_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        roi_gray = gray_image[y:y+h, x:x+w]
        roi_gray = cv2.resize(roi_gray, (48, 48))
        image_pixels = img_to_array(roi_gray)
        image_pixels = np.expand_dims(image_pixels, axis=0)
        image_pixels /= 255.0  # Normalize pixel values
        predictions = model.predict(image_pixels)
        max_index = np.argmax(predictions[0])
        emotion_detection = ( 'Engaged', 'Not Engaged', 'neutral')
        emotion_prediction = emotion_detection[max_index]

        # Display emotion text on the frame
        cv2.putText(frame, emotion_prediction, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

    # Display the frame
    cv2.imshow('Emotion Detection', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the capture and close all windows
cap.release()
cv2.destroyAllWindows()
```

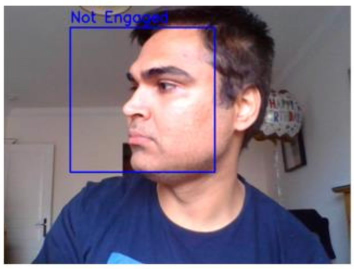
6.4 Webcam-based Engagement Detection System Result in Multi-Class Classification



"Engaged"



"Neutral"



"Not Engaged"