```
#   *************** START OF PROGRAM *****************    #
#   *************** ACTUAL PROGRAM *****************    #
#   ***************** PROGRAM TO PREDICT HAND GESTURES AND UPLOAD THE PREDICTIONS
TO FIREBASE *****************    #
# **************** Imports from keras h5 model Prediction.py File **************** #

import tensorflow.keras
from PIL import ImageTk, Image, ImageOps
import numpy as np

# **************** Imports for GUI **************** #

from tkinter import Tk, Frame, Label, Button, Toplevel
from tkinter import messagebox

# **************** Imports from FB DB upload.py File **************** #

from firebase import firebase

# **************** Imports from OpenCV DC.py File **************** #

import cv2

# **************** Imports for Process Schedulers **************** #

import sys
import os
import time
import threading

# **************** Imports for data analysis and data manipulation **************** #

import pandas as pd
import seaborn as sns
import csv

# **************** Imports for heat map viewing and saving **************** #
from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg


# **************** Determine the way floating point numbers,
# arrays and other NumPy objects are displayed **************** #

np.set_printoptions(suppress=True)

# **************** Accessing firebase real-time database project **************** #

firebase = firebase.FirebaseApplication("https://rtmc-hg-default-rtdb.firebaseio.com/", None)

# **************** Assigning the saved keras.h5 model to a variable **************** #

model = tensorflow.keras.models.load_model('Keras/keras_model.h5')
```

```python
# *************** Assigning/Re-assigning the data in firebase with a Startup String *************** #

firebase.put("/Data", "Preds", "** PROGRAM START **")
time.sleep(3)

# *************** Creating global variables for later usage inside functions *************** #

kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
size = (224, 224)
t1, t2, t3, round_prediction, cap = None, None, None, None, None
off_data_upload_normal, off_data_upload_automate, off_video_capture, start_process,
automate_process, automateprev = \
    False, False, False, False, False, False
list_preds = []

fields = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]


# *************** START of Functions that are used as commands for buttons *************** #


def disable(button):
    # *************** A function that is used to disable buttons based on program's state *************** #

    if button == "start":
        start_button["state"] = "disabled"
        start_button["bg"] = "#d8d8d8"
        start_button["fg"] = "#acacac"

    elif button == "automate":
        automate_button["state"] = "disabled"
        automate_button["bg"] = "#d8d8d8"
        automate_button["fg"] = "#acacac"

    elif button == "automate_prev":
        automate_prev_button["state"] = "disabled"
        automate_prev_button["bg"] = "#d8d8d8"
        automate_prev_button["fg"] = "#acacac"


def start():
    # *************** A function that is bind to start_button to start the program *************** #

    global cap, t1, t2, t3, start_process

    # *************** Determining when start can work *************** #

    if not automateprev and not start_process:

        # *************** Setting the start_process variable to True, Hence multiple clicks doesnt work
*************** #
```

```
        start_process = True

        # *************** Changing the camera_status label to LIVE CAPTURE ON ************** #

        camera_status.config(text="□ LIVE CAPTURE ON", bg="black", fg="#00D100")

        # *************** Removing any existing log.csv file from the system *************** #

        try:
            os.remove("log.csv")
        except FileNotFoundError:
            pass

        # *************** Creating a new log.csv file with initial columns *************** #

        try:
            csvfile_test = open("log.csv", "r")
        except FileNotFoundError:
            csvfile_test = open("log.csv", "w", newline="")
            csv_writer = csv.writer(csvfile_test)
            csv_writer.writerow(fields)
        finally:
            csvfile_test.close()

        # *************** Using OpenCV VideoCapture for capturing live video and setting window
parameters *********** #

        cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
        cap.set(3, 1000)
        cap.set(4, 1000)

        # *************** threads for keeping active the function video_stream and
        # constant_upload without both interfering *************** #

        t1 = threading.Thread(target=video_stream)
        t1.start()
        time.sleep(1)
        t2 = threading.Thread(target=constant_upload)
        t2.start()
        t3 = threading.Thread(target=timed_loop_prediction)

        # *************** Disabling automate_prev button *********** #

        disable("automate_prev")

    else:
        pass


def automate():
    # *************** A function that is bind to automate_button to automate the program *************** #
    global off_video_capture, off_data_upload_normal, list_preds, t3, cap, automate_process

    # *************** Determining when automate can work *************** #

    if start_process and not automate_process:

        # *************** Setting the automate_process variable to True, Hence multiple clicks doesnt work
************ #
        # *************** Setting the off_video_capture, off_data_upload_normal to True to stop the
process *********** #
        # *************** Stopping the live video capture by cap.release() *************** #

        automate_process = True
        off_video_capture = True
        off_data_upload_normal = True
        cap.release()

        # *************** Changing the camera_status label to LIVE CAPTURE OFF ************** #

        camera_status.config(text="▯ LIVE CAPTURE OFF", bg="black", fg="red")

        # *************** Giving the lmain label the initial image ************** #

        lmain.config(image=initial_image)

        # *************** Storing the predictions to a .csv file ************** #

        with open("log.csv", "a", newline="") as log:
            log_writer = csv.writer(log)
            log_writer.writerows(list_preds)

        # *************** Starting the thread t3 for automated data upload ************** #

        t3.start()

        # *************** Disabling start button *********** #

        disable("start")

    else:
        pass


def automate_prev():
    # *************** A function that is bind to automate_prev_button to automate from the existing .csv
file ******** #

    global automateprev

    # *************** Determining when automate_prev can work *************** #

    if not start_process and not automateprev:
        # *************** Reading from the existing .csv file *************** #
        # *************** Starting the thread t3 for automated data upload *************** #

        try:

            with open("log.csv", "r") as log_read:
                log_reader = csv.reader(log_read)
                for row in log_reader:
                    if row == fields:
                        pass
                    else:
                        current_pred = [int(i) for i in row]
                        list_preds.append(current_pred)

                t3 = threading.Thread(target=timed_loop_prediction)
                t3.start()

            # *************** Setting the automateprev variable to True, Hence multiple clicks doesnt work
************ #

            automateprev = True

            # *************** Disabling start and automate button ************ #

            disable("start")
            disable("automate")

        except FileNotFoundError:
            messagebox.showwarning(title="WARNING", message="THERE IS NO PREVIOUS log.csv
FILE\n"
                                                            "CLICK THE START BUTTON TO CREATE ONE")
            pass


def heatmap_popup(figure):

    # *************** A function that is used to view the graph in a pop-up window ******** #

    top = Toplevel(root)
    top.title("CLUSTER-MAP")
    canvas = FigureCanvasTkAgg(figure, master=top)  # A tk.DrawingArea.
    canvas.draw()
    canvas.get_tk_widget().pack()



def insight():
    # *************** A function that is bind to insight_button for generating useful graphs ******** #

    try:
        log = pd.read_csv("log.csv")
        htmap = sns.clustermap(log, cmap="viridis", figsize=(12, 6), linewidths=0.1, annot=True,
linecolor='white')
        plt.savefig('CLUSTER-MAP.png')
        heatmap_popup(htmap.fig)

    except OSError or FileNotFoundError:
        messagebox.showwarning(title="WARNING", message="FILE IN USE/NO log.csv
EXISTS\nCLOSE THE FILE AND TRY AGAIN\n"
                                                     "OR PRESS START TO CREATE A NEW log.csv")
        pass

    except ValueError:
        pass


def stop():
    # *************** A function that is bind to stop_button to stop the program *************** #

    global off_data_upload_normal, off_video_capture, off_data_upload_automate, list_preds

    # *************** Creating a .csv file when start is pressed and automate is not pressed ***************
#
    # *************** Setting required variable to true to stop the process *************** #

    if start_process and not automate_process:
        with open("log.csv", "a", newline="") as log:
            log_writer = csv.writer(log)
            log_writer.writerows(list_preds)
            root.destroy()
            off_data_upload_normal = True
            off_data_upload_automate = True
            off_video_capture = True
            sys.exit()
    else:
        root.destroy()
        off_data_upload_normal = True
        off_data_upload_automate = True
        off_video_capture = True
        sys.exit()


# *************** END of Functions that are used as commands for buttons *************** #

# *************** Function that returns the README content in messagebox *************** #


def general_message():
    text = "General Instructions :\n\n" \
           "1. Initially, the START, AUTOMATE(prev.), INSIGHT, STOP buttons work.\n\n" \
           "2. The START button starts a new process by clearing the previous\n" \
           "log.csv proceeding to start the live data capture and prediction while\n" \
           "uploading the predicted data to the online database system in the\n" \
           "interval of 1 second.\n\n" \
           "3. The AUTOMATE(prev.) button loops through the data that is present\n" \
           "in the log.csv that is generated before, meanwhile uploading the data\n" \
           "to the online database system in intervals of 1 second.\n\n" \
           "4. The INSIGHT button generates a HEATMAP with annotations from the\n" \
           "log.csv.\n\n" \
           "5. The STOP button stops any process that is happening and closes\n" \
           "the window.\n\n" \
           "Constraints and Specifics :\n\n" \
           "1. The AUTOMATE button works only after the START button is pressed,\n" \
           "and the AUTOMATE button loops through the data that is already predicted\n" \
           "meanwhile uploading it to the online database system in intervals of 1\n" \
           "second, It also generates a new log.csv file that contains the data that is\n" \
           "predicted.\n\n" \
           "2. Initially the INSIGHT button works, when the START button is pressed\n" \
           "INSIGHT works only after AUTOMATE button is also pressed.\n\n" \
           "3. The AUTOMATE(prev.) button works only when the START button is\n" \
           "not pressed and vice versa.\n\n" \
           "4. The STOP button generates a log.csv file only when the START\n" \
           "button is pressed and AUTOMATE button is not pressed.\n\n" \
           "5. The LIVE CAPTURE OFF text changes to LIVE CAPTURE ON only when\n" \
           "the START button is pressed and the PROCESS NOT STARTED text\n" \
           "changes to the predicted output(ie. Hand Closed, Hand Open, etc.) when\n" \
           "the START/AUTOMATE(prev.) button is pressed, also it denotes whether\n" \
           "the data being uploaded is LIVE or AUTOMATED.\n\n" \
           "6. If there is no log.csv and INSIGHT or AUTOMATE(prev.) is pressed\n" \
           "initially, then the INSIGHT and AUTOMATE(prev.) shows a descriptive warning. "

    return text


# *************** Creating a window called as root for GUI *************** #

root = Tk()
root.title("RTMC-HG")
root.config(bg="black")
root.iconbitmap("Image/robotic-arm.ico")
root.focus()

# *************** Creating a frame *************** #
app = Frame(root)
app.grid(column=1, row=1, rowspan=5)

# *************** Creating a label in the frame amd assigning a starting image for it *************** #

initial_image = ImageTk.PhotoImage(file="Image/Initial_Image.png")
lmain = Label(app)
lmain.config(image=initial_image)
lmain.grid()

# *************** Creating a label in the frame amd assigning the text - □ LIVE CAPTURE OFF
*************** #

camera_status = Label()
camera_status.config(text="□ LIVE CAPTURE OFF", bg="black", fg="red")
camera_status.grid(column=0, row=0, columnspan=2)
```

```
# **************** Creating a label in the frame amd assigning the text - PROCESS NOT STARTED
**************** #

prediction_status = Label()
prediction_status.config(text="PROCESS NOT STARTED", bg="black", fg="red")
prediction_status.grid(column=0, row=6, columnspan=2)

# **************** Creating buttons inside the window to perform different tasks **************** #

start_button = Button(text="START", command=start, bg="#59981A", fg="white", width=9)

start_button.grid(column=0, row=1)
automate_button = Button(text="AUTOMATE", bg="#191970", fg="white", width=9,
command=automate)
automate_button.grid(column=0, row=2)
automate_prev_button = Button(text="AUTOMATE\n(prev.)", bg="#191970", fg="white", width=9,
command=automate_prev)
automate_prev_button.grid(column=0, row=3)
insight_button = Button(text="INSIGHT", width=9, bg="#191970", fg="white", command=insight)
insight_button.grid(column=0, row=4)
stop_button = Button(text="STOP", command=stop, bg="red", fg="white", width=9)
stop_button.grid(column=0, row=5)

# **************** Creating a popup that contains the general instructions, constraints and specifics
**************** #

messagebox.showinfo(title="README", message=general_message())

# **************** Function that are bind to the threads **************** #


# noinspection PyUnresolvedReferences,PyTypeChecker
def video_stream():
    # **************** Function that Constantly predicts the output for the input image and
    # updates the global variable round_prediction **************** #

    global round_prediction
    # **************** Code for opening the log.csv file in append mode **************** #
    if not off_video_capture:
        # **************** Code for reading each frame, and preprocessing the data **************** #

        success, img = cap.read()
        imgGrey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgblur = cv2.GaussianBlur(imgGrey, (3, 3), sigmaX=0, sigmaY=0)
        imgOut = cv2.flip(imgblur, 1)
        imgOut = imgOut[300:850, 850:2700]
        imgOut = cv2.filter2D(imgOut, -1, kernel)
        imgOut = cv2.resize(imgOut, (224, 224))

        # **************** Code for converting the image to the suitable format so that it can be used as
        # image for label inside the GUI window **************** #
        img = Image.fromarray(imgOut).convert("RGB")
        imgtk = ImageTk.PhotoImage(image=img)
        lmain.imgtk = imgtk
        lmain.configure(image=imgtk)

        # *************** Code for prediction *************** #

        image = ImageOps.fit(img, size, Image.ANTIALIAS)
        image_array = np.asarray(image)
        normalized_image_array = (image_array.astype(np.float32) / 127.0) - 1
        data[0] = normalized_image_array
        prediction = model.predict(data)
        round_prediction = [round(i) for i in prediction[0]]

        # *************** Code for updating each frame to the label inside the GUI window *************** #

        lmain.after(1, video_stream)


def prediction_viewer(round_prediction):
    # *************** Function that checks the prediction and returns the appropriate text *************** #

    if round_prediction[0] == 1:
        return "Hand_Closed"
    elif round_prediction[1] == 1:
        return "Index"
    elif round_prediction[2] == 1:
        return "Middle"
    elif round_prediction[3] == 1:
        return "Ring"
    elif round_prediction[4] == 1:
        return "Little"
    elif round_prediction[5] == 1:
        return "Thumb"
    elif round_prediction[6] == 1:
        return "Index_Little"
    elif round_prediction[7] == 1:
        return "Index_Middle"
    elif round_prediction[8] == 1:
        return "Middle_Ring"
    elif round_prediction[9] == 1:
        return "Ring_Little"
    elif round_prediction[10] == 1:
        return "Thumb_Index"
    elif round_prediction[11] == 1:
        return "Thumb_Little"
    elif round_prediction[12] == 1:
        return "Index_Middle_Ring"
    elif round_prediction[13] == 1:
        return "Middle_Ring_Little"
    elif round_prediction[14] == 1:
        return "Thumb_Index_Little"
    elif round_prediction[15] == 1:
        return "Thumb_Index_Middle"
    elif round_prediction[16] == 1:
        return "Index_Middle_Ring_Little"
    elif round_prediction[17] == 1:
        return "Thumb_Index_Middle_Ring"
    elif round_prediction[18] == 1:
        return "Hand_Open"
    elif round_prediction[19] == 1:
        return "Partial"


def constant_upload():
    # *************** Function that Constantly uploads the predictions to the Firebase real time database
*********** #

    global round_prediction, list_preds
    # *************** Constant upload of the prediction to firebase on interval of 1 sec *************** #
    while not off_data_upload_normal:
        time.sleep(1)
        # *************** Appending the predictions that are sent to firebase to list_preds
        # so that it can be used for automation *************** #
        list_preds.append(round_prediction)

        # *************** Changes the label prediction_status with the text received from prediction_viewer
********** #

        try:
            if not automate_process:
                prediction_status.config(text=f"LIVE PREDICTION : {prediction_viewer(round_prediction)}"
                                              f" [{len(list_preds)}]", fg="#00D100")
        except RuntimeError:
            pass
        firebase.put("/Data", "Preds", str(round_prediction))


def timed_loop_prediction():
    # *************** Function that Constantly uploads when automate/automate_prev in use
    # to the Firebase real time database  *********** #

    global list_preds
    while not off_data_upload_automate or off_data_upload_normal:
        for i in list_preds:
            if not off_data_upload_automate:
                time.sleep(1)

                # *************** Changes the label prediction_status with the text received from
prediction_viewer ** #

                try:
                    prediction_status.config(text=f"AUTOMATED UPLOAD : {prediction_viewer(i)}",
fg="#00D100")
                except RuntimeError:
                    pass
        else:
            exit()
        firebase.put("/Data", "Preds", str(i))



#    *************** mainloop to maintain the window on screen ******************    #

root.mainloop()

#    *************** END OF PROGRAM ******************    #
```