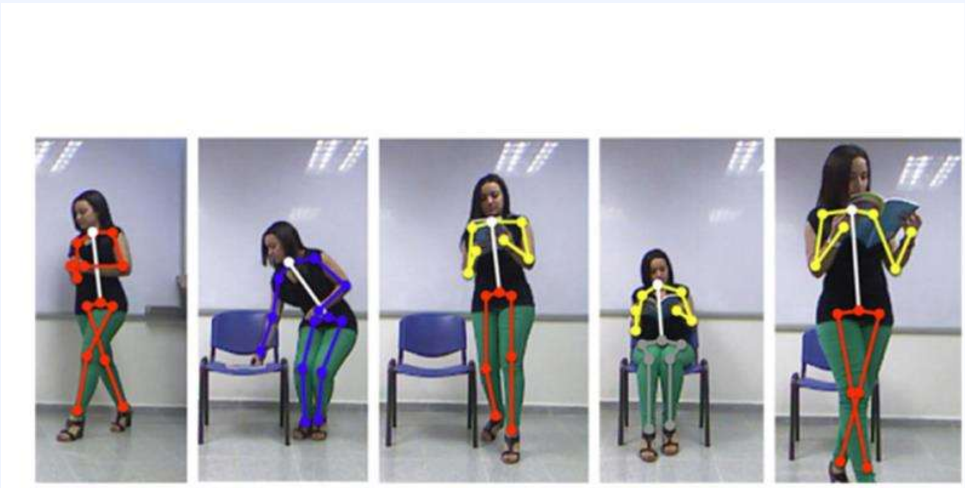
Project on Human Action Recognition

"Interactive User Interface for Human Action Recognition"



09/08/2023

Celebal Summer Internship

Akash Singh

Table of Contents:

1. Introduction
2. Data Collection and Preprocessing
3. Model Training
4. Model Evaluation
5. Model Saving and Inference Pipeline
6. Testing the Pipeline
7. Creating the Streamlit User Interface

1. Introduction:

The objective of this internship project was to develop a robust human action recognition system using machine learning techniques. This report presents the process and outcomes of implementing such a system.

PROBLEM STATEMENT:

- **Human Action Recognition (HAR)** aims to understand human behavior and assign a label to each action. It has a wide range of applications, and therefore has been attracting increasing attention in the field of computer vision. Human actions can be represented using various data modalities, such as RGB, skeleton, depth, infrared, point cloud, event stream, audio, acceleration, radar, and WiFi signal, which encode different sources of useful yet distinct information and have various advantages depending on the application scenarios.

- Consequently, lots of existing works have attempted to investigate different types of approaches for HAR using various modalities.

- Your Task is to build an Image Classification Model using CNN that classifies to which class of activity a human is performing.

2. Data Collection and Preprocessing:

The Human Action Recognition (HAR) dataset was sourced from Kaggle. Preprocessing steps included data cleaning and normalization to ensure consistency and reliability.

Code:

jupyter HAR_Classification Last Checkpoint: 07/26/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

In [11]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from keras.applications import VGG16
from keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping
```

In [12]:
```
# Load the dataset
dataset_path = r"C:\Human Action Recognition\train"
train_file = r"C:\Human Action Recognition\Training_set.csv" # not using the test csv because it does not contains labels
```

In [13]:
```
# Read the train file
train_df = pd.read_csv(train_file, sep=',', header=None, names=['filename', 'label'])
```

In [14]:
```
# separating the train data into two parts training and testing
train_df, test_df = train_test_split(train_df, test_size=0.2, random_state=42)
test_df
```

Out[14]:

	filename	label
4058	Image_4058.jpg	hugging
1399	Image_1399.jpg	eating
2034	Image_2034.jpg	sitting
3528	Image_3528.jpg	listening_to_music
7383	Image_7383.jpg	hugging
...	...	...
8057	Image_8057.jpg	cycling
12163	Image_12163.jpg	sitting
2108	Image_2108.jpg	hugging
4100	Image_4100.jpg	running
12515	Image_12515.jpg	listening_to_music

2521 rows × 2 columns

In [15]:
```
# Preprocess the data
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

batch_size=32
img_height = 224
img_width = 224
classes = ["sitting","using laptop","hugging",
           "sleeping","drinking","clapping","dancing",
           "cycling","calling","laughing","eating","fighting",
           "listening_to_music","running","texting"
          ]


train_generator = train_datagen.flow_from_dataframe(
    train_df,
    directory=dataset_path,
    x_col='filename',
    y_col='label',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    classes = classes,
    shuffle = True,
    seed = 42
   )
```

```
test_generator = test_datagen.flow_from_dataframe(
    test_df,
    directory = dataset_path,
    x_col='filename',
    y_col='label',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    classes=classes,
    shuffle = False,
)
```

```
Found 9404 validated image filenames belonging to 15 classes.
Found 2356 validated image filenames belonging to 15 classes.
```

```
C:\Users\pk boss\anaconda3\lib\site-packages\keras\src\preprocessing\image.py:1137: UserWarning: Found 1 invalid image filename
(s) in x_col="filename". These filename(s) will be ignored.
  warnings.warn(
```

3. Model Training:

A convolutional neural network (CNN) architecture was employed for human action recognition and after that we used VGG16 model to improve the accuracy.

Code:

```
In [6]: model = Sequential()
        model.add(Conv2D(32, (3,3), activation = 'relu', input_shape=(img_height,img_width,3)))
        model.add(MaxPooling2D(2,2))
        model.add(Conv2D(64, (3,3), activation = 'relu'))
        model.add(MaxPooling2D(2,2))
        model.add(Conv2D(128, (3,3), activation = 'relu'))
        model.add(MaxPooling2D(2,2))
        model.add(Flatten())
        model.add(Dense(128, activation = 'relu'))
        model.add(Dropout(0.5))
        model.add(Dense(len(classes), activation = 'softmax'))
```

```
In [7]: # Compile the model
        model.compile(optimizer=Adam(learning_rate=0.001),
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])
```

```
Epoch 1/10
294/294 [==============================] - 437s 1s/step - loss: 2.5790 - accuracy: 0.1332 - val_loss: 2.4065 - val_accuracy: 0.
1876
Epoch 2/10
294/294 [==============================] - 428s 1s/step - loss: 2.3372 - accuracy: 0.2124 - val_loss: 2.2966 - val_accuracy: 0.
2288
Epoch 3/10
294/294 [==============================] - 442s 2s/step - loss: 2.1493 - accuracy: 0.2870 - val_loss: 2.2176 - val_accuracy: 0.
2661
Epoch 4/10
294/294 [==============================] - 428s 1s/step - loss: 1.9406 - accuracy: 0.3591 - val_loss: 2.1719 - val_accuracy: 0.
2878
Epoch 5/10
294/294 [==============================] - 431s 1s/step - loss: 1.6750 - accuracy: 0.4378 - val_loss: 2.3298 - val_accuracy: 0.
2742
Epoch 6/10
294/294 [==============================] - 433s 1s/step - loss: 1.3936 - accuracy: 0.5289 - val_loss: 2.3600 - val_accuracy: 0.
3103
Epoch 7/10
294/294 [==============================] - 433s 1s/step - loss: 1.1523 - accuracy: 0.6068 - val_loss: 2.4283 - val_accuracy: 0.
3226
```

Using VGG16 Model:

Using CNN model we got accuracy around 60% and now we are applying VGG16 model to improve accuracy.

Code:

```
In [40]: # Load the VGG-16 model with pre-trained ImageNet weights
         from keras.application import VGG16
         vgg16_base = VGG16(weights='imagenet',
                            include_top=False,
                            input_shape=(img_height, img_width, 3))

         # Freeze the VGG16 base layers to use their weights
         for layer in vgg16_base.layers:
             layer.trainable = False

         flatten_layer = Flatten()(vgg16_base.output)
         dense_layer = Dense(256, activation = 'relu')(flatten_layer)
         output_layer = Dense(15, activation = 'softmax')(dense_layer)

         vgg_model = Model(inputs = vgg16_base.input, outputs = output_layer)

         # Train the model
         epochs = 10
         steps_per_epoch = len(train_generator)
         validation_steps = len(test_generator)

         history = vgg_model.fit(train_generator,
                                 epochs=epochs,
                                 steps_per_epoch=steps_per_epoch,
                                 validation_data=test_generator,
                                 validation_steps=validation_steps,
                                 callbacks=[early_stopping])
```

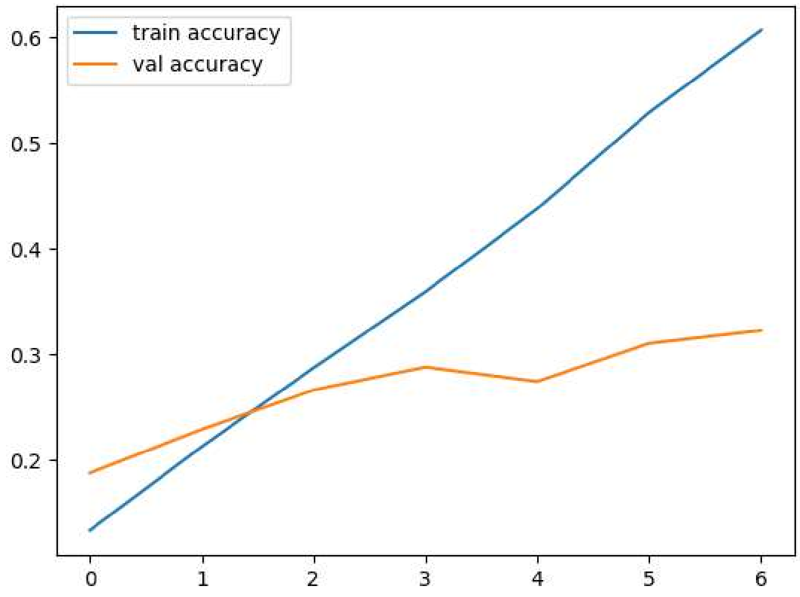
Output:

```
Epoch 1/10
294/294 [==============================] - 132s 448ms/step - loss: 0.9446 - accuracy: 0.6965 - val_loss: 1.4542 - val_accuracy: 0.5514
Epoch 2/10
294/294 [==============================] - 136s 461ms/step - loss: 0.8535 - accuracy: 0.7200 - val_loss: 1.4246 - val_accuracy: 0.5789
Epoch 3/10
294/294 [==============================] - 134s 457ms/step - loss: 0.7511 - accuracy: 0.7534 - val_loss: 1.4208 - val_accuracy: 0.5696
Epoch 4/10
294/294 [==============================] - 133s 453ms/step - loss: 0.6867 - accuracy: 0.7725 - val_loss: 1.5978 - val_accuracy: 0.5522
Epoch 5/10
294/294 [==============================] - 132s 448ms/step - loss: 0.6234 - accuracy: 0.7972 - val_loss: 1.7269 - val_accuracy: 0.5365
Epoch 6/10
294/294 [==============================] - 132s 448ms/step - loss: 0.5735 - accuracy: 0.8106 - val_loss: 1.5812 - val_accuracy: 0.5679
Epoch 7/10
294/294 [==============================] - 134s 457ms/step - loss: 0.5486 - accuracy: 0.8186 - val_loss: 1.7044 - val_accuracy: 0.5577
Epoch 8/10
294/294 [==============================] - 133s 453ms/step - loss: 0.4676 - accuracy: 0.8469 - val_loss: 1.7726 - val_accuracy: 0.5501
```

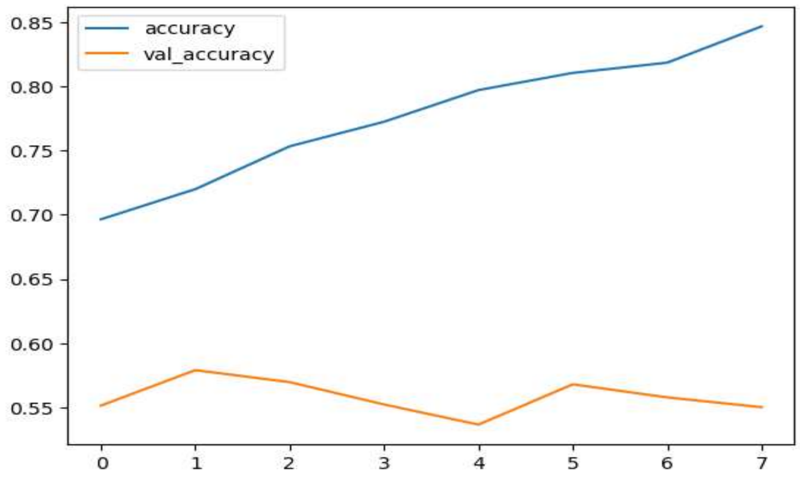
4. Model Evaluation:

The trained model achieved an accuracy of over 80% by using VGG16 model and accuracy of 60% on CNN model.

CNN model accuracy(60%)-



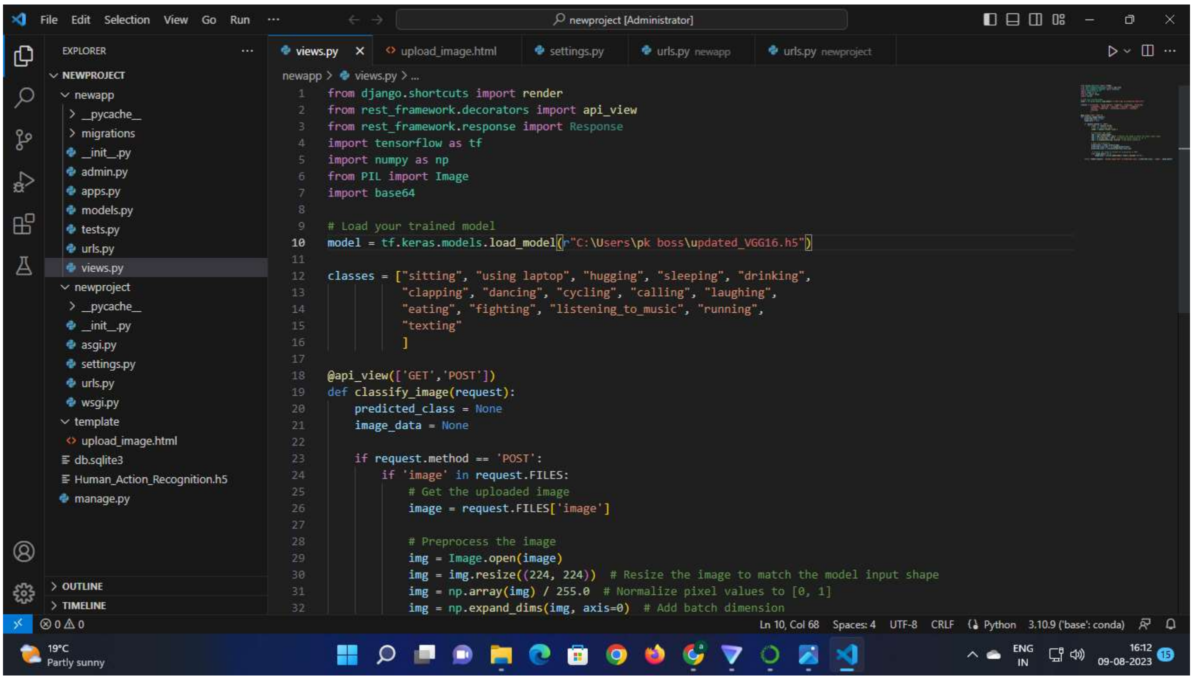
VGG16 model accuracy(84%)-



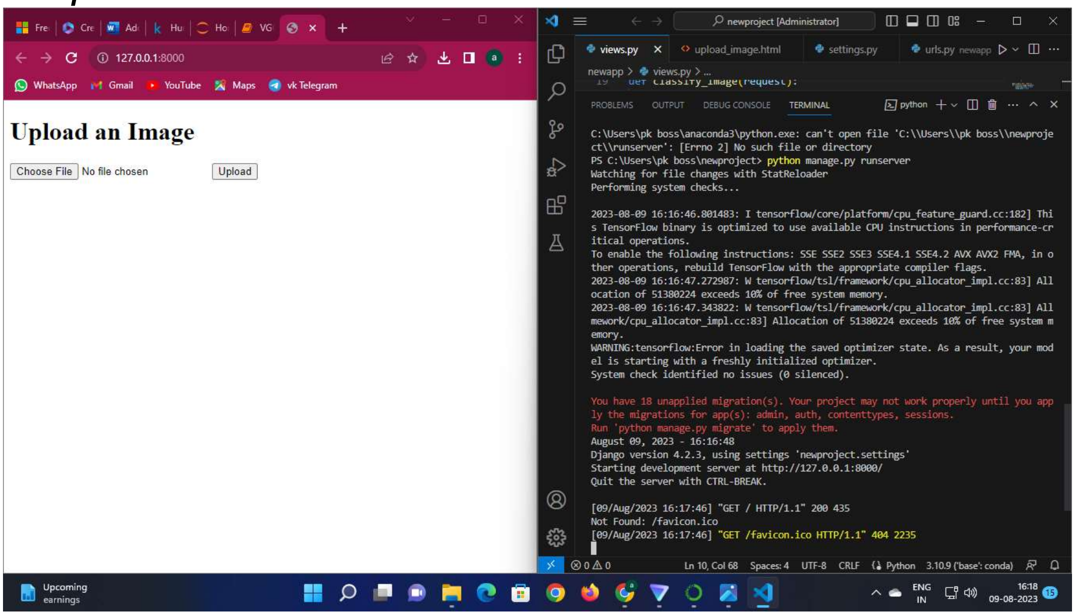
5. Model Saving and Inference Pipeline:

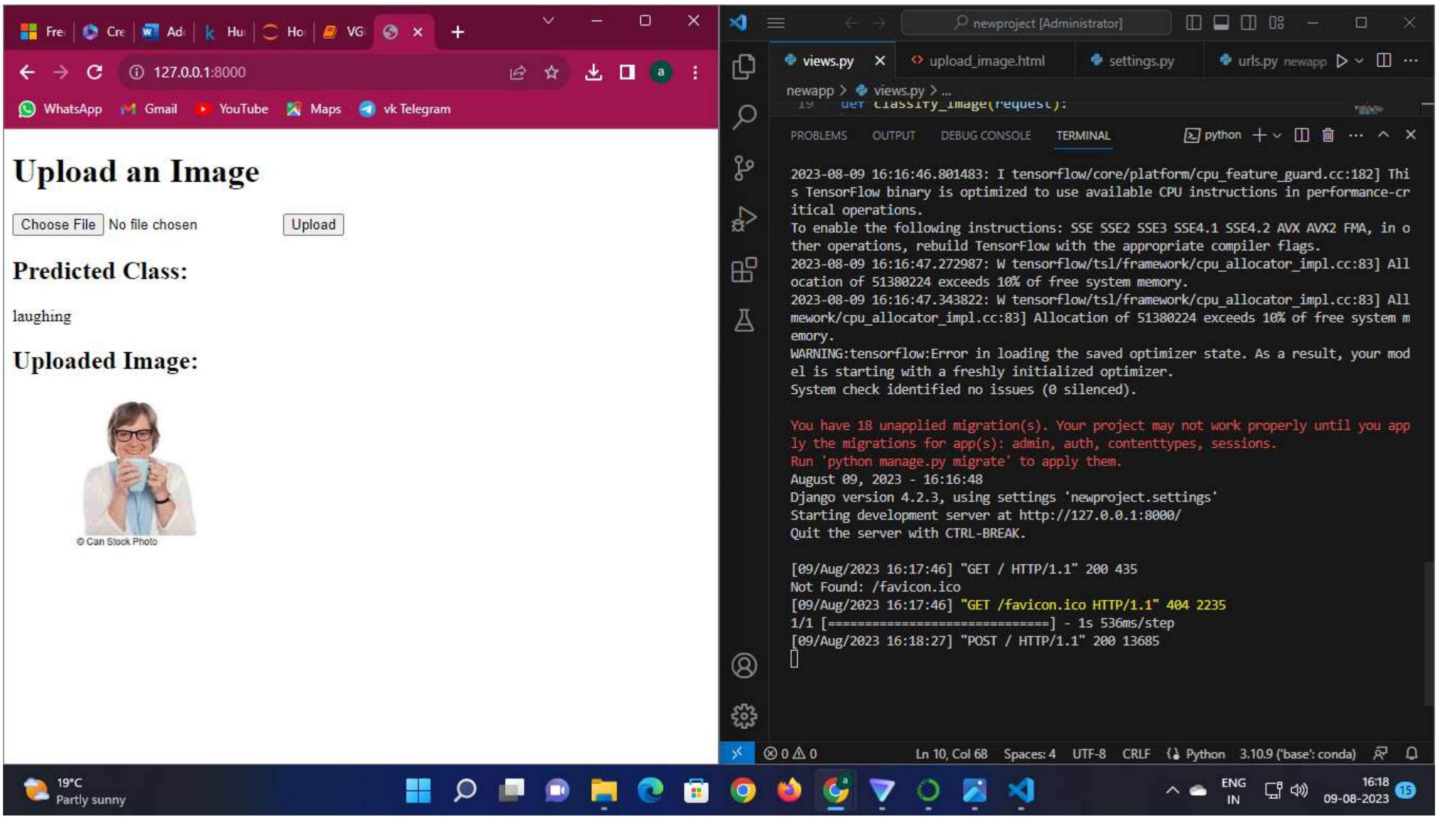
The trained model was saved and integrated into a Django-based inference pipeline.

Code:
Django project -



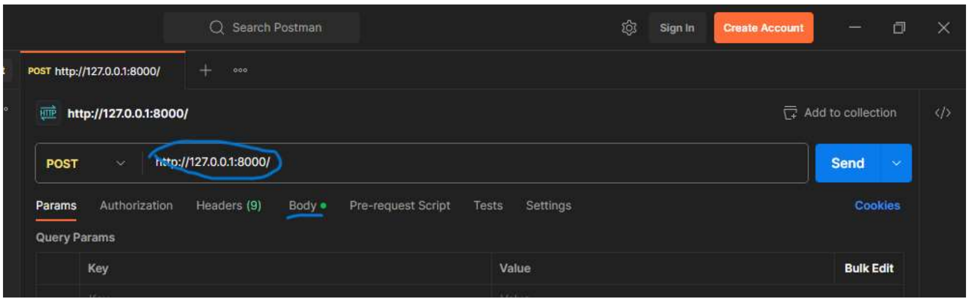
Output:



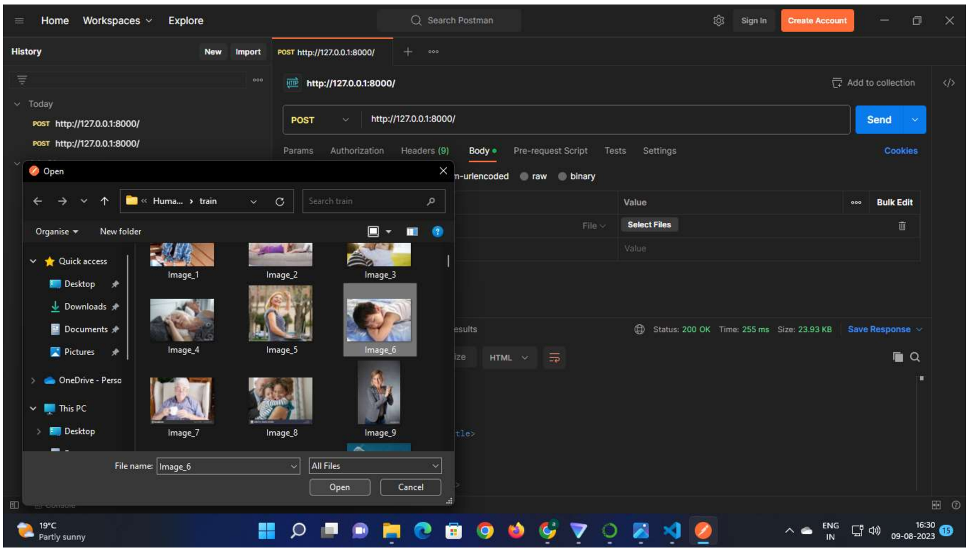

127.0.0.1:8000
WhatsApp
Gmail
YouTube
Maps
vk Telegram
Upload an Image
Choose File
No file chosen
Upload
Predicted Class:
laughing
Uploaded Image:
© Can Stock Photo
newproject [Administrator]
views.py
upload_image.html
settings.py
urls.py newapp
newapp > views.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
python
2023-08-09 16:16:46.801483: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-08-09 16:16:47.272987: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 51380224 exceeds 10% of free system memory.
2023-08-09 16:16:47.343822: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allmework/cpu_allocator_impl.cc:83] Allocation of 51380224 exceeds 10% of free system memory.
WARNING:tensorflow:Error in loading the saved optimizer state. As a result, your model is starting with a freshly initialized optimizer.
System check identified no issues (0 silenced).
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
August 09, 2023 - 16:16:48
Django version 4.2.3, using settings 'newproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[09/Aug/2023 16:17:46] "GET / HTTP/1.1" 200 435
Not Found: /favicon.ico
[09/Aug/2023 16:17:46] "GET /favicon.ico HTTP/1.1" 404 2235
1/1 [==============================] - 1s 536ms/step
[09/Aug/2023 16:18:27] "POST / HTTP/1.1" 200 13685
Ln 10, Col 68
Spaces: 4
UTF-8
CRLF
Python
3.10.9 ('base': conda)
19°C
Partly sunny
ENG
IN
16:18
09-08-2023

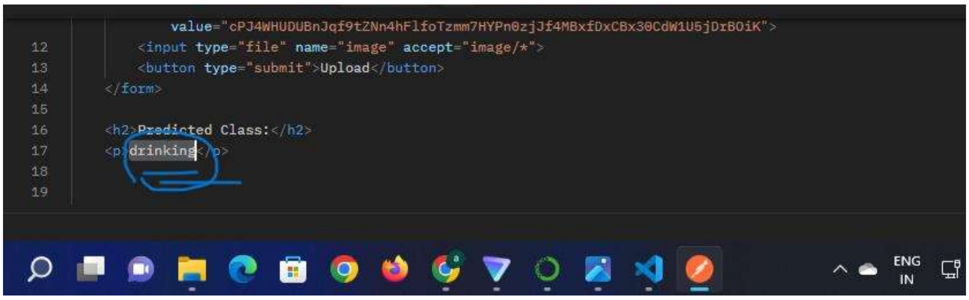
6. Testing the Pipeline: *Testing the pipeline was facilitated through Postman. The step-by-step guide allows users to upload images and receive predictions.*

Entering the running url server and then select body



Click on choose file.



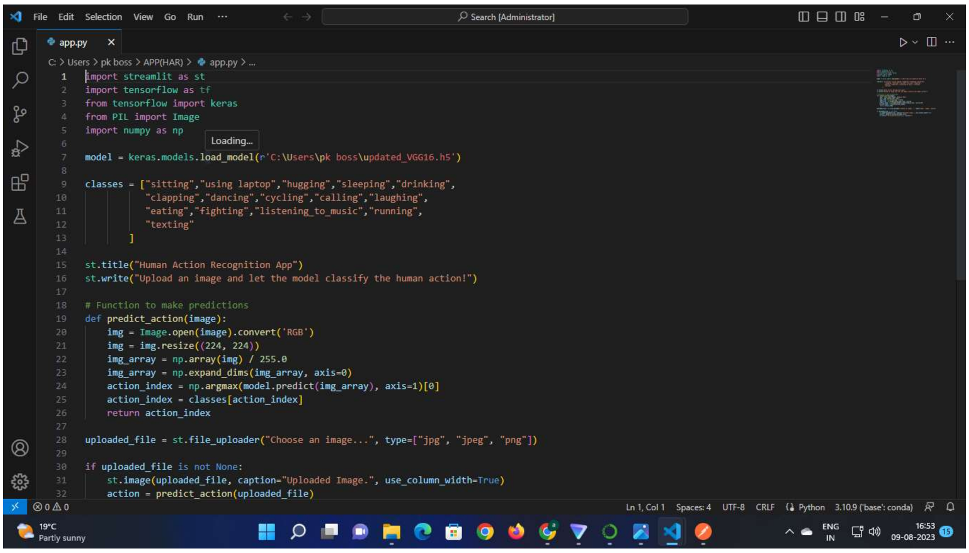


Predicted class is “drinking”

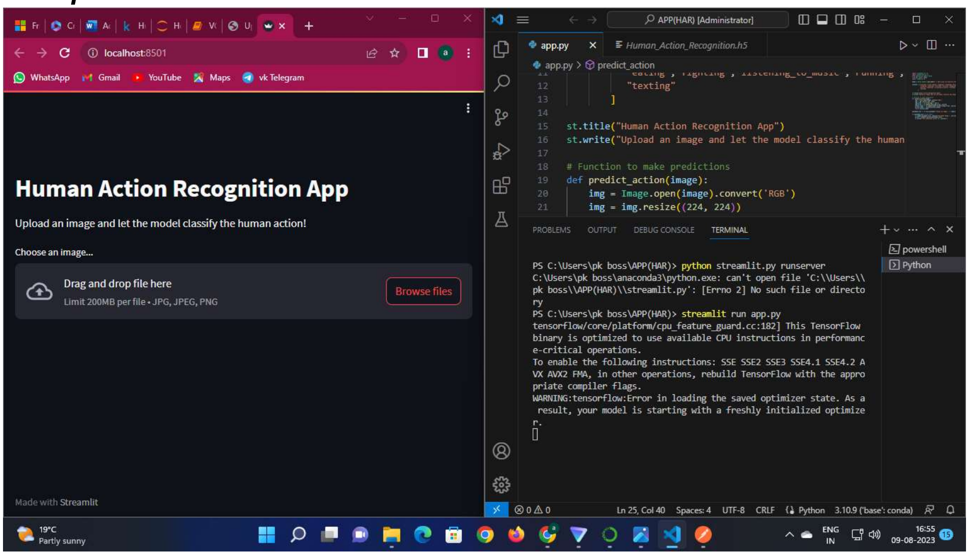
7. Creating the Streamlit User Interface:

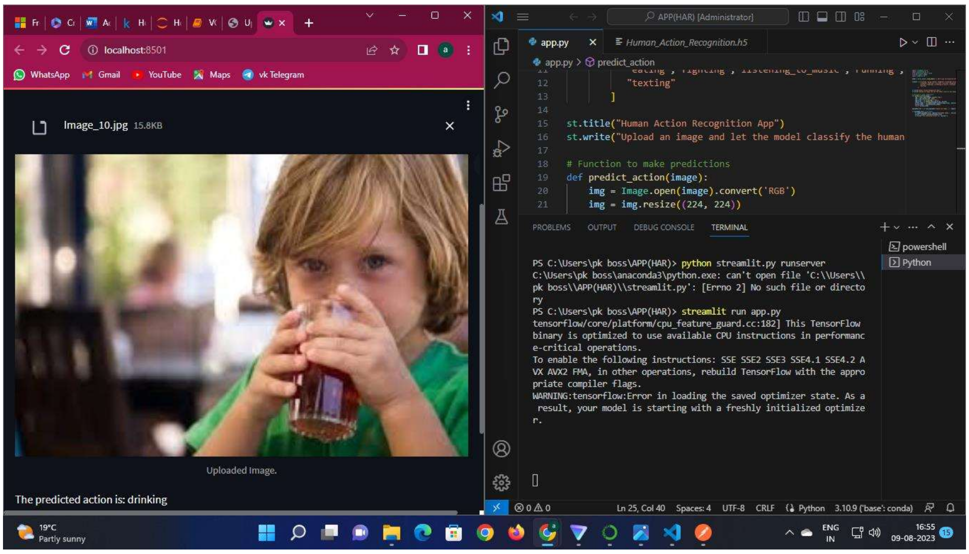
A user-friendly Streamlit UI was developed to provide an interactive platform for users to upload images and receive real-time predictions.

Code:



Output:





Link for my github - https://github.com/AkashSingh2002/Human_Action_Recognition_Using_VGG16