A gesture based tool for sterile browsing of radiology images CNN and open cv

(Gesture based Automation Tool)

Group -20

- I AJAY KUMAR REDDY – 18BCE7183
- ATLURI NIKHIL RAM – 18BCE7144
- K MANI VAMSI KUMAR – 18BCE7331
- ANURAG SOMANI – 19BCY10090
- VIKANKSH GAUTAM – 19BAI10024

Introduction

- Humans are able to recognize body and sign language easily. This is possible due to the combination of vision and synaptic interactions that were formed along brain development.

- In this project Gesture based Desktop automation, First the model is trained pre trained on the images of different hand gestures, such as showing numbers with fingers as 1,2,3,4.

- This model uses the integrated webcam to capture the video frame. The image of the gesture captured in the video frame is compared with the pre trained model and the gesture is identified.

- If the gesture predicts is 1 then images is blurred; 2 then image is resized; 3 then images is rotated etc

Project objectives

- Know fundamental concepts and techniques of Convolutional Neural Network (CNN)
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data pre-processing techniques.
- Know how to build a web application using Flask framework.

Image pre-processing

In this we will be improving the data that supresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation etc.

Import the image data generator library

```
In [1]: from keras.preprocessing.image import ImageDataGenerator
        Using TensorFlow backend.
```

Image data augmentation is a technique that can be used t artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class

Configure image data generator class

Image Data Argumentation

```
In [4]: train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
        test_datagen=ImageDataGenerator(rescale=1./255)
```

- ImageDataGenerator class is instantiated and the configuration for the types of data augmentation
- There are five main types of data augmentation techniques
  1) Image shifts via the width_shift_range and height_shift_range arguments
  2) Image flips via the horizontal_flip and vertical_flip arguments
  3) Image rotations via the rotation_range arguments
  4) Image brightness via the brightness_range arguments
  5) Image zoom via the zoom_range argument

Applying image data generator functionality to trainset and to testset

Loading our data and performing data Argumentation

```
In [5]: x_train = train_datagen.flow_from_directory(r"C:\Dataset\train",target_size = (64,64),batch_size = 5, color_mode='grayscale', cl
        x_test= test_datagen.flow_from_directory(r"C:\Dataset\test",target_size = (64,64),batch_size = 5, color_mode = 'grayscale',class_

        Found 594 images belonging to 6 classes.
        Found 30 images belonging to 6 classes.
```

This function will return batches of images from the subdirectories 0,1,2,3,4,5 together with labels 0 to 5 {‘0’ : 0, ‘1’ : 1, ‘2’ : 2, ‘3’ : 3, ‘4’ : 4, ‘5’ : 5 }

Model Building

Time to build our Convolutional Neural Networking which contains a input layer along with convolution, maxpooling and finally an output layer

Importing the model building libraries

In [1]:
```
import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

from tensorflow.keras.layers import Dense,Flatten

from tensorflow.keras.layers import Conv2D,MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

```
Using TensorFlow backend.
```

Initializing the model

```
In [9]: # Initializing the CNN
        classifier = Sequential()
```

Sequential model is a linear stack of layers.

You can create a sequential model by passing a list of layer instances to the constructor from keras models import Sequential from keras

Adding CNN Layers

```
# First convolution layer and pooling
classifier.add(Conv2D(32, (3, 3), input_shape=(64, 64, 1), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Second convolution layer and pooling
classifier.add(Conv2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening the layers
classifier.add(Flatten())
```

We are adding a convolution layer with activation function as “relu” and with a small filter size (3,3) and number of filters (32) followed by a max pooling layer

Maxpool layer is used to downsample the input

Flatten layer flattens the output

Adding Dense Layers

- Dense layer is deeply connected neural network layer. It is most common and frequently used layer.
- Understanding the model is very important phase to properly use it for training and prediction purposes.
- Keras provides a simple method, summary to get the full information about the models and its layers.

```
In [10]: # Adding a fully connected layer
         classifier.add(Dense(units=128, activation='relu'))
         classifier.add(Dense(units=6, activation='softmax')) # softmax for more than 2
```

```
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)
_________________________________________________________________
flatten (Flatten)            (None, 6272)
_________________________________________________________________
dense (Dense)                (None, 128)
_________________________________________________________________
dense_1 (Dense)              (None, 6)
=================================================================
Total params: 813,286
Trainable params: 813,286
Non-trainable params: 0
_________________________________________________________________
```

Configure the learning process

The compilation is the final step in creating the model. Once the compilation is done, we can move on to the training process.

Optimization is an important process which optimize the input weights by comparing the prediction and the loss function

Metrics is used to evaluate the performance of our model. It is similar to loss function, but not used in training process

```
In [12]: classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Train the model

Fit_generator functions are used to train a deep learning neural network

Arguments

Steps_per_epoch : It specifies the total number of steps taken from the generator as soon as one epoch is finished and next epoch has started.

Epochs : an integer and number of epochs we want to train our model for

Validation_data can be either

- An inputs and targets list
- A generator
- An inputs, targets and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

```
In [13]: classifier.fit_generator(
             generator=x_train,steps_per_epoch = len(x_train),
             epochs=50, validation_data=x_test,validation_steps = len(x_test))
```

Save the model

```
In [14]: classifier.save('gesture.h5')

In [15]: model_json = classifier.to_json()
         with open("model-bw.json", "w") as json_file:
             json_file.write(model_json)
```

- The model is saved with .h5 extension.
- An H5 file is a data file saved in the Hierarchical Data Format (HDF)
- It contains multidimensional arrays of scientific data

Test the model

Evaluation is a process during development of the model to check whether the model is best fir for the given problem and corresponding data

```
In [24]: index=['0','1','2','3','4','5']
         result=str(index[pred[0]])
         result

Out[24]: '3'
```

```
In [23]: img = image.load_img(r"C:\Manisha\3.jpg",grayscale=True,target_size= (64,64))
         x = image.img_to_array(img)
         x = np.expand_dims(x,axis = 0)
         pred = model.predict_classes(x)
         pred

Out[23]: array([3], dtype=int64)
```

```
In [16]: from tensorflow.keras.models import load_model
         from keras.preprocessing import image
         model = load_model("gesture.h5")
```

Application building

After the model is trained in this particular milestone, we will be building our flask application which will be running in out local browser with a user interface

Create html pages

We use HTML to create the front end part of the web page

Here, we created 3 HTML pages – home.html, intro.html and index6.html

Home.html displays the home page

Intro.html displays the introduction about the hand gesture recognition

Index6.html accepts the input from the user and predicts the values.

We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages

Build Python code

Importing Libraries

```
from flask import Flask,render_template,request
# Flask-It is our framework which we are going to use to run/serve our application.
#request-for accessing file which was uploaded by the user on our application.
import operator
import cv2 # opencv library
from tensorflow.keras.models import load_model#to load our trained model
import os
from werkzeug.utils import secure_filename
```

Creating our flask applications and loading our model

```
app = Flask(__name__,template_folder="templates") # initializing a flask app
# Loading the model
model=load_model('gesture.h5')
print("Loaded model from disk")
```

Routing to the html page

```
@app.route('/')# route to display the home page
def home():
    return render_template('home.html')#rendering the home page


@app.route('/intro') # routes to the intro page
def intro():
    return render_template('intro.html')#rendering the intro page

@app.route('/image1',methods=['GET','POST'])# routes to the index html
def image1():
    return render_template("index6.html")
```

Code

Getting our input and storing it

```
if request.method == 'POST':
    print("inside image")
    f = request.files['image']

    basepath = os.path.dirname(__file__)
    file_path = os.path.join(basepath, 'uploads', secure_filename(f.filename))
    f.save(file_path)
```

Grab the frames from the web cam

```
cap = cv2.VideoCapture(0)
while True:
    _, frame = cap.read() #capturing the video frame values
    # Simulating mirror image
    frame = cv2.flip(frame, 1)
```

Creating ROI

```
# Got this from collect-data.py
# Coordinates of the ROI
x1 = int(0.5*frame.shape[1])
y1 = 10
x2 = frame.shape[1]-10
y2 = int(0.5*frame.shape[1])
# Drawing the ROI
# The increment/decrement by 1 is to compensate for the bounding box
cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
# Extracting the ROI
roi = frame[y1:y2, x1:x2]

# Resizing the ROI so it can be fed to the model for prediction
roi = cv2.resize(roi, (64, 64))
roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
_, test_image = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
cv2.imshow("test", test_image)
```

Code

Predicting our results

```
result = model.predict(test_image.reshape(1, 64, 64, 1))
prediction = {'ZERO': result[0][0],
              'ONE': result[0][1],
              'TWO': result[0][2],
              'THREE': result[0][3],
              'FOUR': result[0][4],
              'FIVE': result[0][5]}
# Sorting based on top prediction
prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)

# Displaying the predictions
cv2.putText(frame, prediction[0][0], (10, 120), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.imshow("Frame", frame)
```

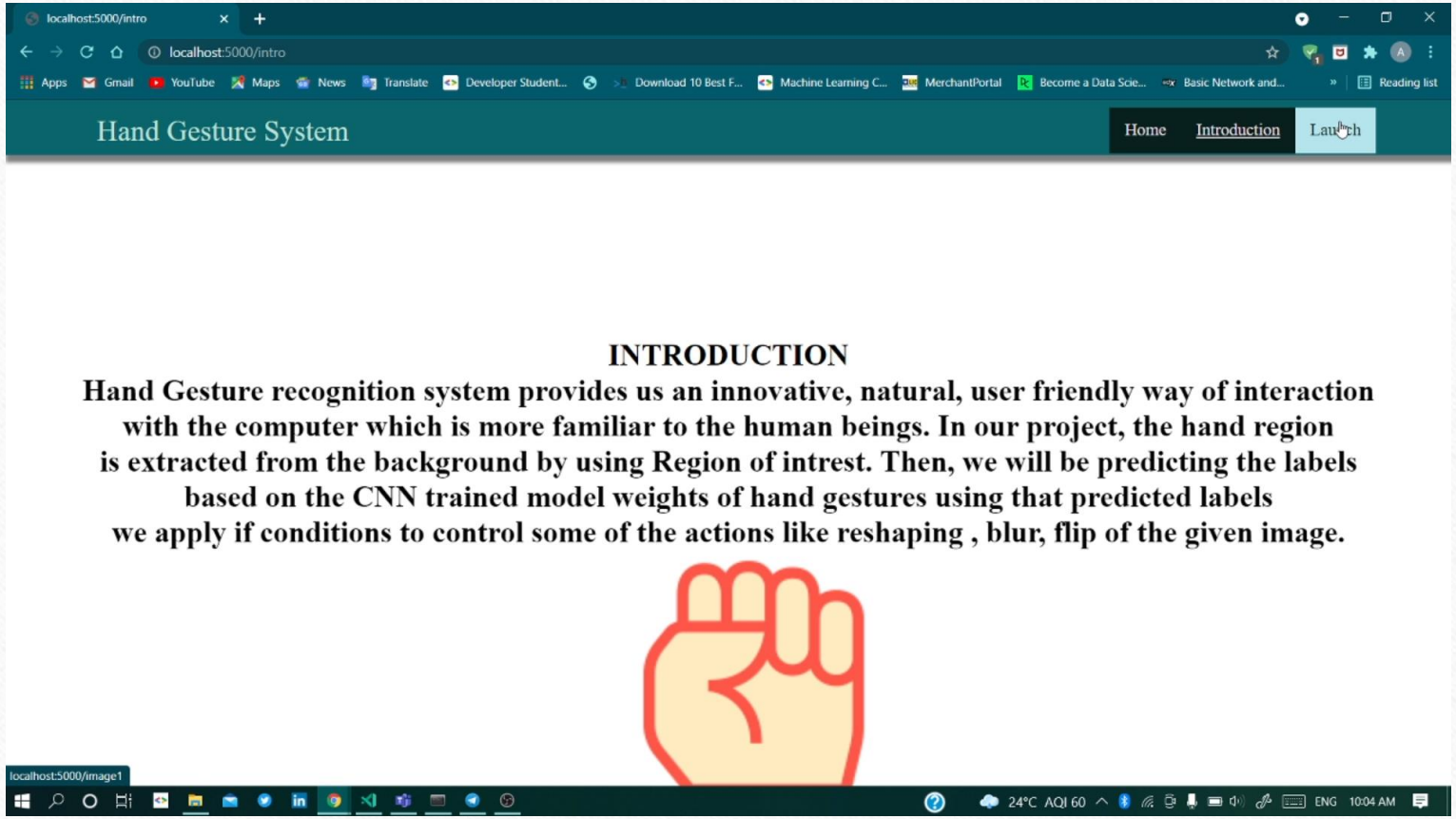
```
            #loading an image
            image1=cv2.imread(file_path)
            if prediction[0][0]=='ONE':

                resized = cv2.resize(image1, (200, 200))
                cv2.imshow("Fixed Resizing", resized)
                key=cv2.waitKey(3000)

                if (key & 0xFF) == ord("1"):
                    cv2.destroyWindow("Fixed Resizing")

            elif prediction[0][0]=='ZERO':

                cv2.rectangle(image1, (480, 170), (650, 420), (0, 0, 255), 2)
                cv2.imshow("Rectangle", image1)
                cv2.waitKey(0)
                key=cv2.waitKey(3000)
                if (key & 0xFF) == ord("0"):
                    cv2.destroyWindow("Rectangle")

            elif prediction[0][0]=='TWO':
                (h, w, d) = image1.shape
                center = (w // 2, h // 2)
                M = cv2.getRotationMatrix2D(center, -45, 1.0)
                rotated = cv2.warpAffine(image1, M, (w, h))
                cv2.imshow("OpenCV Rotation", rotated)
                key=cv2.waitKey(3000)
                if (key & 0xFF) == ord("2"):
                    cv2.destroyWindow("OpenCV Rotation")

            elif prediction[0][0]=='THREE':
                blurred = cv2.GaussianBlur(image1, (11, 11), 0)
                cv2.imshow("Blurred", blurred)
                key=cv2.waitKey(3000)
                if (key & 0xFF) == ord("3"):
                    cv2.destroyWindow("Blurred")
            else:
                continue


            interrupt = cv2.waitKey(10)
            if interrupt & 0xFF == 27: # esc key
                break


        cap.release()
        cv2.destroyAllWindows()
    return render_template("home.html")
```

RUNNING THE APPLICATION

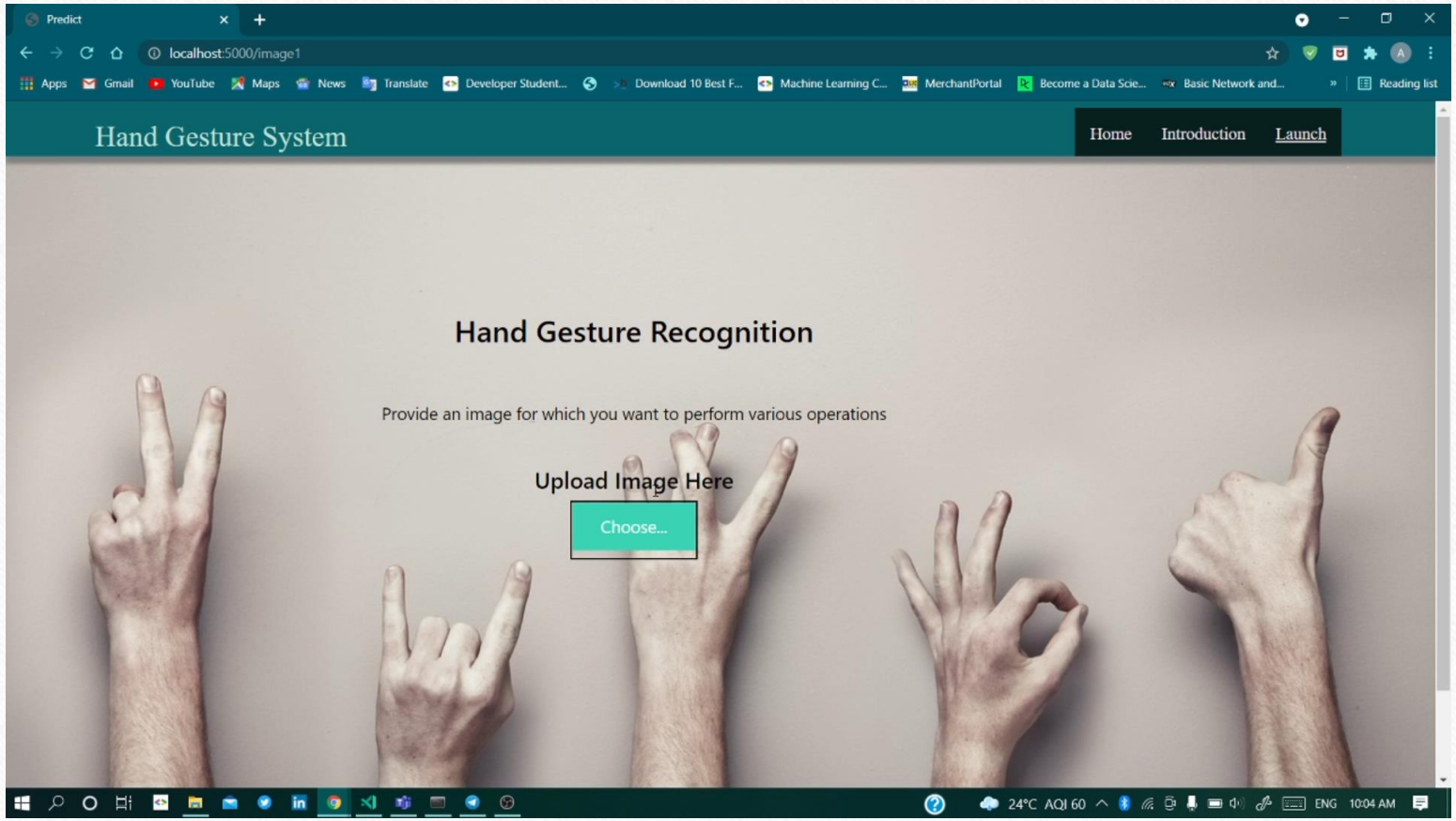
Home page


localhost:5000
HAND GESTURE RECOGNITION
Home
Introduction
Launch
HAND GESTURE RECOGNITION
OF
RADIOLOGY IMAGES
THROUGH
STERILE BROWSING

Introduction page



Prediction page



Thank You