DIGITAL SIGNAL PROCESSING PROJECT

NUMBER SIGN RECOGNITION USING OPEN-CV AND KERAS (CONVOLUTIONAL NEURAL NETWORK (CNN))

Abstract :

Hand gesture recognition system received great attention in the last few years because of its manifoldness applications and the ability to interact with machines efficiently through human computer interaction. In this paper a survey of recent hand gesture recognition systems is presented. Key issues of hand gesture recognition systems are presented with challenges of gesture systems.

Application : Hand shaking gestures detection using image processing.

Index :

Introduction:

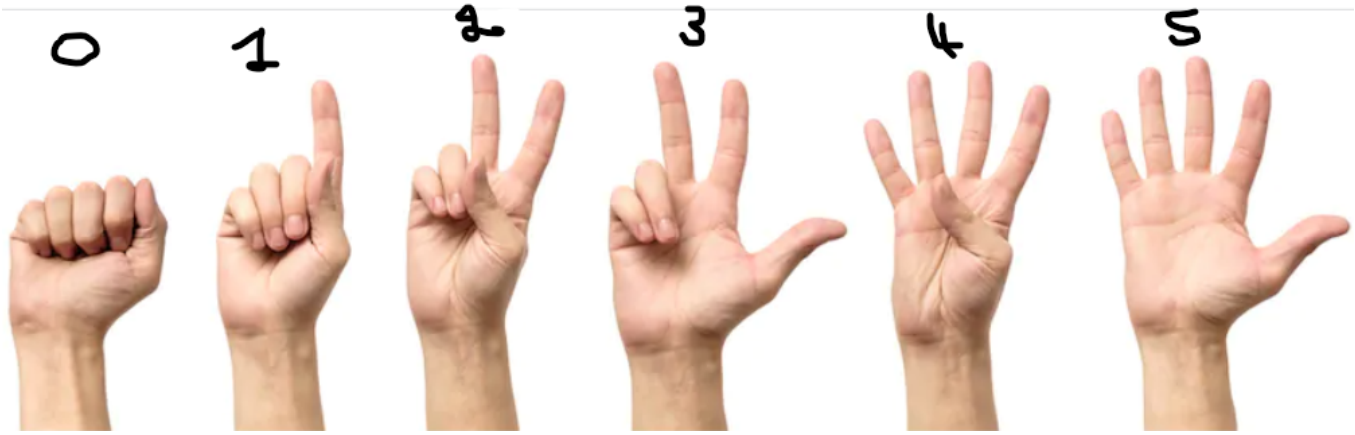
This application has the design and development of a gesture recognition system to recognize finger hand gestures. We developed this solution using the latest deep learning technique called **convolutional neural networks**. This system uses blink detection to initiate the

recognition process, Convex Hull-based hand segmentation with adaptive skin color filtering to segment the hand region, and a convolutional neural network to perform gesture recognition. An ensemble of four convolutional neural networks are trained with a dataset of images for gesture recognition and a feedback unit called head pose estimation is implemented to validate the correctness of predicted gestures. This entire system was developed using Python programming language and other supporting libraries like OpenCV, keras and CNN to perform various image processing and machine learning tasks.

To build a SLR (Sign Language Recognition) we will need three things:

1. Collect Data
2. Model (In this case we will use a CNN)
3. Platform to apply our model (We are going to use OpenCV)

OUR EXACT AIM IS



Our exact aim is to identify the above hand gestures by CNN

<u>Idea for how to implement the project</u> :

- First we save the image in some particular folders like directories.
- By Using a webcam we can take the hand gesture and save it in that folder.

- Then train the code using convolutional neural networks(CNN).
- Then predict the image which we gave a hand gesture to CNN.
- So in python we can import keras libraries and some modules which are useful to this project.
- By using an opencv system you can take any picture through a webcam.
- Now we have to predict the number from the input image. Our model will give outputs as integers .

PROJECT PLANNING & IT'S EXECUTION :

- ❏ We have divide our project into 3 parts
- ❏ Part 1 is to create a directories and capture the image by frames and converting into grayscale and importing to the respective directories by labeling.
- ❏ Part 2is trains the neural network with giving numbers 0,1,2,3,4,5 to respective images with the help of the keyboard on the console.
- ❏ Part 3 is testing the image after some training we must predict by the help of the cnn .

- ❖ We will use CNN (Convolutional Neural Network) to recognise the Number . So We are going to use the keras library in python .
- ❖ Now we will Train code by using CNN.
- ❖ And then predict the number by using CNN.

Requirements of the project :

1) Pc with a webcam support
2) Spyder software(Installed with numpy and open cv libraries)

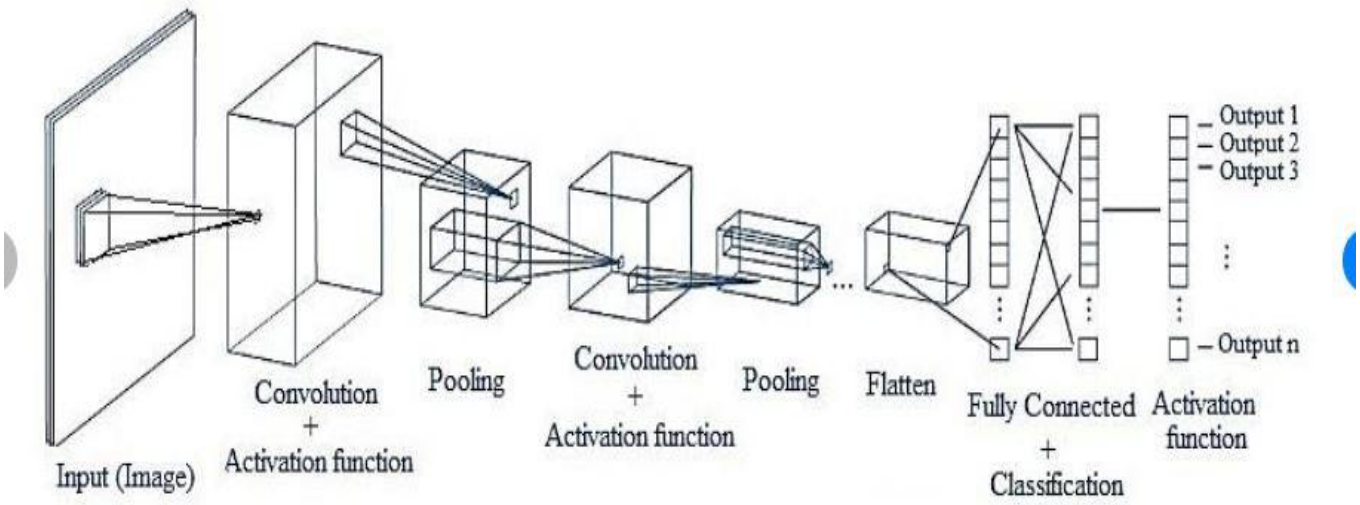
3) Keras libraries

Libraries used in this project:

1. A great way to use deep learning to classify images is to build a convolutional neural network (CNN).
2. Keras library
3. Open cv library
4. Numpy library
5. Os library

Importance of Libraries and Modules :

Convolutional Neural networks :

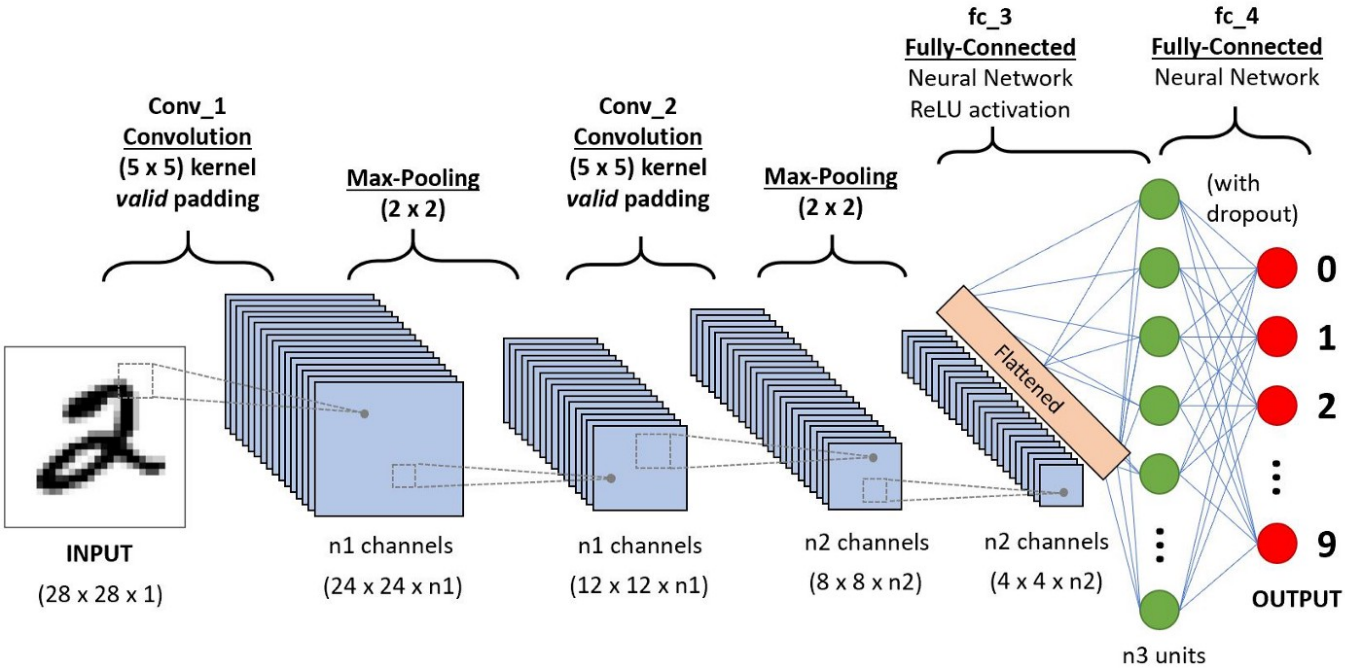


The basic structure of convolution neural networks

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

The pre-processing required in a ConvNet is much lower as compared to other classification algorithms.

While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



A CNN sequence to classify handwritten digits

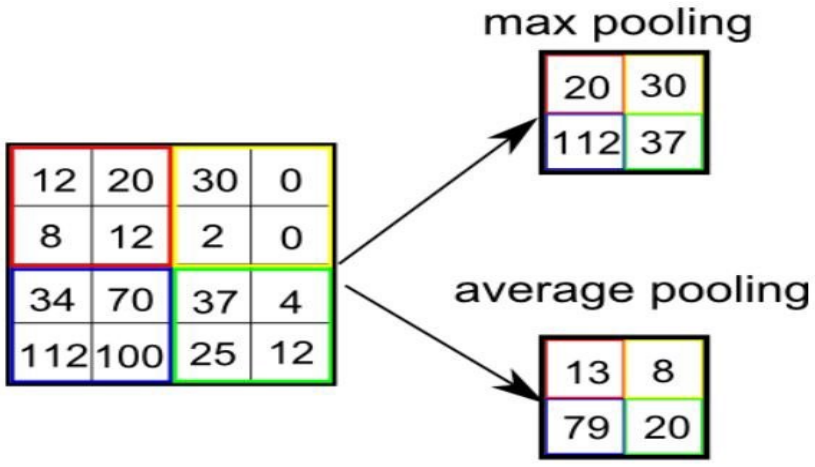
The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

Pooling Layer :

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training the model.

There are two types of Pooling: **Max Pooling** and **Average Pooling**. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling** performs a lot better than **Average Pooling**.

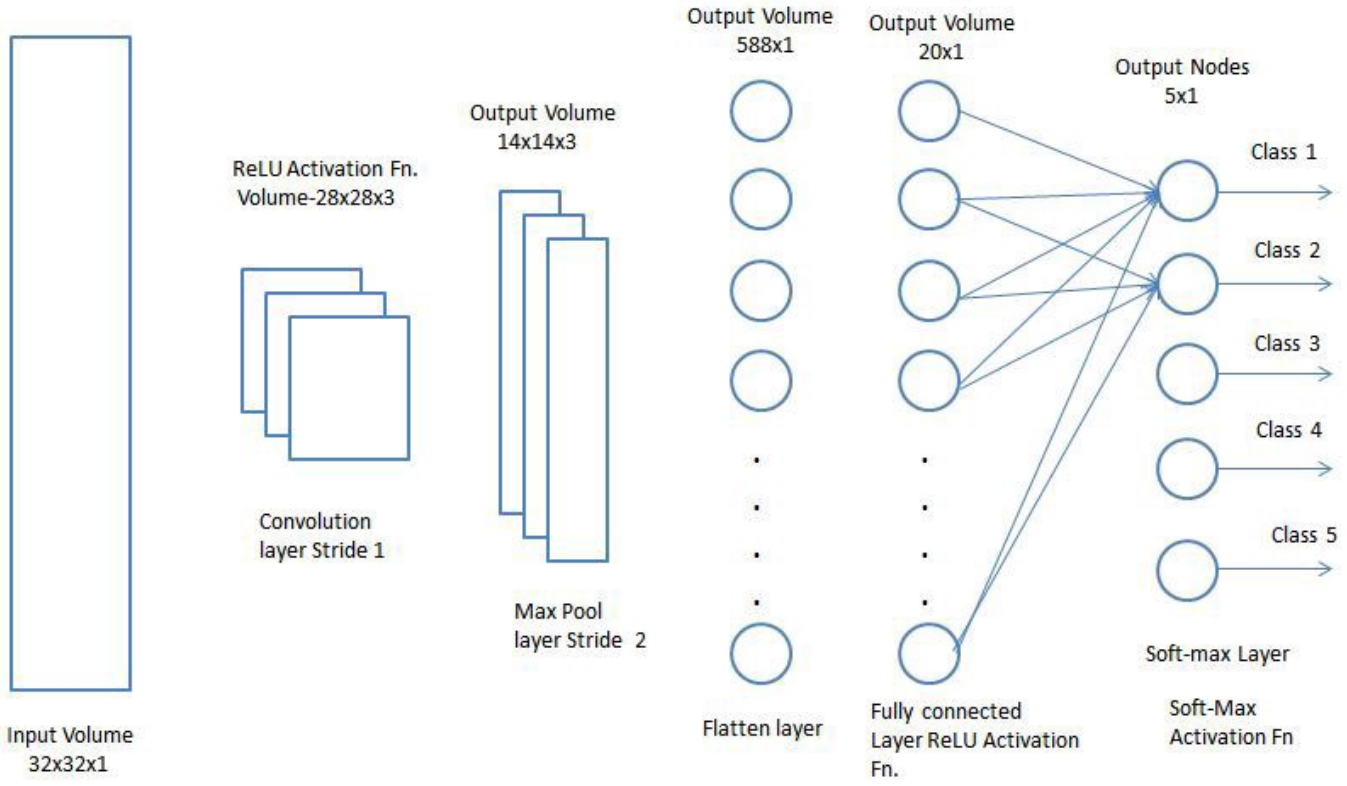


The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number

of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

Classification — Fully Connected Layer (FC Layer)



Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique.

Keras Library :

KERAS is an Open Source Neural Network library written in Python that runs on top of Theano or Tensorflow. It is designed to be modular, fast and easy to use. It was developed by François Chollet, a Google engineer.

Keras doesn't handle low-level computation. Instead, it uses another library to do it, called the "Backend. So Keras is a high-level API wrapper for the low-level API, capable of running on top of TensorFlow, CNTK, or Theano.

Keras High-Level API handles the way we make models, defining layers, or set up multiple input-output models. In this level, Keras also compiles our model with loss and optimizer functions, training process with fit function. Keras doesn't handle Low-Level API such as making the computational graph, making tensors or other variables because it has been handled by the "backend" engine.

Keras was developed and maintained by François Chollet, a Google engineer using four guiding principles:

- **Modularity**: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
- **Minimalism**: The library provides just enough to achieve an outcome, no frills and maximizing readability.
- **Extensibility**: New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
- **Python**: No separate model files with custom file formats. Everything is native Python.

Advantages of Keras

Fast Deployment and Easy to understand

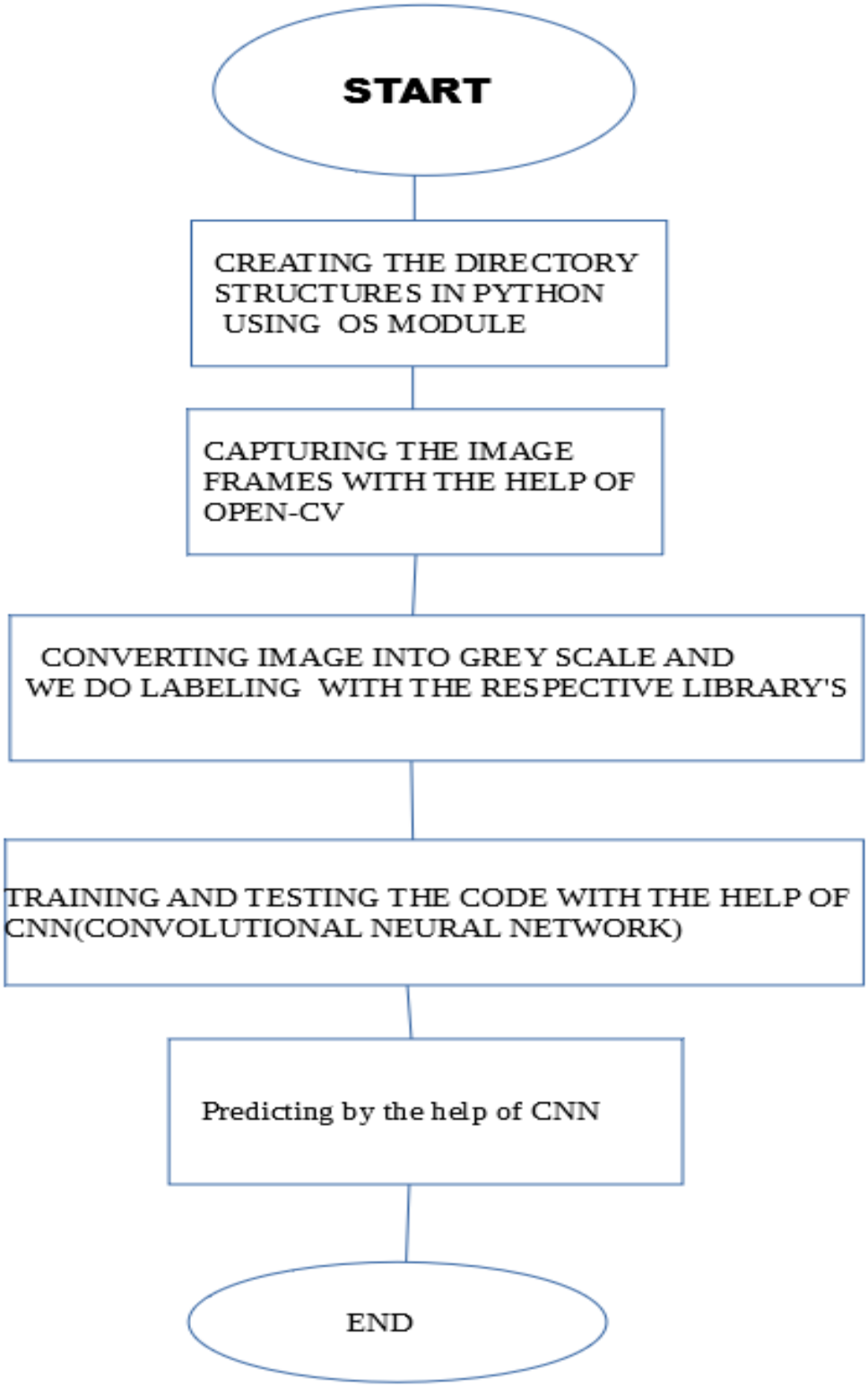
Keras is very quick to make a network model. If you want to make a simple network model with a few lines, Keras

Disadvantages of Keras

Cannot handle low-level API

Keras only handles high-level API which runs on top of other frameworks or backend engines such as Tensorflow, Theano, or CNTK. So it's not very useful if you want to make your own abstract layer for your research purposes because Keras already have pre-configured layers.

TENTATIVE FLOW CHART OF THE PROJECT :



Project Code:

Collect data:

```
#importing  the following modules and libraries are:

import cv2
import numpy as np
import os

# Create the directory structure

if not os.path.exists("data"):
    os.makedirs("data")                    # to create directory/file
    os.makedirs("data/train")
    os.makedirs("data/test")


    os.makedirs("data/train/0")
    os.makedirs("data/train/1")
    os.makedirs("data/train/2")
    os.makedirs("data/train/3")
    os.makedirs("data/train/4")
    os.makedirs("data/train/5")


    os.makedirs("data/test/0")
    os.makedirs("data/test/1")
    os.makedirs("data/test/2")
    os.makedirs("data/test/3")
    os.makedirs("data/test/4")
    os.makedirs("data/test/5")


mode = 'train'                             # Train or test
```

```
directory = 'data/'+mode+'/'                #It saves to corresponding path


cap = cv2.VideoCapture(0)                   #Capturing video through webcam

while True:
    _, frame = cap.read()                   # To view the captured frame

      frame = cv2.flip(frame, 1)            # Simulating mirror image




    count = {'zero': len(os.listdir(directory+"/0")),      # Getting count of existing
images
            'one': len(os.listdir(directory+"/1")),
            'two':  len(os.listdir(directory+"/2")),
            'three':  len(os.listdir(directory+"/3")),
            'four':  len(os.listdir(directory+"/4")),
            'five':  len(os.listdir(directory+"/5"))}

   # Printing the count in each set to the screen
   # Using cv2.putText() method
   #cv2.putText() method is used to draw a text string on any image
    cv2.putText(frame, "MODE : "+mode, (10, 50), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "IMAGE COUNT", (10, 100), cv2.FONT_HERSHEY_PLAIN,
1 , (0,255,255), 1)
    cv2.putText(frame, "ZERO : "+str(count['zero']), (10, 120),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "ONE : "+str(count['one']), (10, 140),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "TWO : "+str(count['two']), (10, 160),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "THREE : "+str(count['three']), (10, 180),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "FOUR : "+str(count['four']), (10, 200),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
```

```
    cv2.putText(frame, "FIVE : "+str(count['five']), (10, 220),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

#Defining the ROI(Region of Interest)
    x1 = int(0.5*frame.shape[1])              # Coordinates of the ROI
    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]
    roi = cv2.resize(roi, (64, 64))  #Resize the image 64x64

    cv2.imshow("Frame", frame)

    #_, mask = cv2.threshold(mask, 200, 255, cv2.THRESH_BINARY)
    #kernel = np.ones((1, 1), np.uint8)
    #img = cv2.dilate(mask, kernel, iterations=1)
    #img = cv2.erode(mask, kernel, iterations=1)
    # do the processing after capturing the image!
    roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)  # converting the captured
image from ROI into Grayscale
    _, roi = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)  # In RGB 0 for
white and 255 corresponds to Black
    cv2.imshow("ROI", roi)

    interrupt = cv2.waitKey(10)   #Keyboard Interrupt
#It sends the ROI to correct folder by corresponding  pressing  the labeled key
    if interrupt & 0xFF == 27: # when we press esc key  it breaks the loop it directly
goes to release and it closes all windows open
        break
    if interrupt & 0xFF == ord('0'):
        cv2.imwrite(directory+'0/'+str(count['zero'])+'.jpg', roi)
    if interrupt & 0xFF == ord('1'):
        cv2.imwrite(directory+'1/'+str(count['one'])+'.jpg', roi)
    if interrupt & 0xFF == ord('2'):
```

```
        cv2.imwrite(directory+'2/'+str(count['two'])+'.jpg', roi)
    if interrupt &oxFF == ord('3'):
        cv2.imwrite(directory+'3/'+str(count['three'])+'.jpg', roi)
    if interrupt & oxFF == ord('4'):
        cv2.imwrite(directory+'4/'+str(count['four'])+'.jpg', roi)
    if interrupt & oxFF == ord('5'):
        cv2.imwrite(directory+'5/'+str(count['five'])+'.jpg', roi)

cap.release()
cv2.destroyAllWindows()
"""
d = "old-data/test/0"
newd = "data/test/0"
for walk in os.walk(d):
    for file in walk[2]:
        roi = cv2.imread(d+"/"+file)
        roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
        _, mask = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
        cv2.imwrite(newd+"/"+file, mask)
"""
```

An explanation for the above code:

How It works:

1. We have to create a window to take the input from our webcam by importing Opencv in python .
2. Now we created six categories(directories) like zero,One,Two,Three,Four,Five and define different modes like train mode or test mode.
3. Here in train mode ,we will show our hand with some fingers open.
4. Then Create ROI to take the picture from a webcam.
5. Then after capturing the image through a webcam ,then press the corresponding number of fingers.

6. Then it goes directly into the corresponding folders which were created before.
7. Image count is there and it increases with adding different sample images for every numeric digit.
8. By giving input from the keyboard then it decides to save in which folder.

Train data:

Code:

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense

# Step 1 - Building the CNN

# Initializing the CNN
classifier = Sequential() #adding layers into it

# First convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), input_shape=(64, 64, 1), activation='relu'))
#We have a black and white image
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolution layer and pooling
classifier.add(Convolution2D(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)))  #Actually it is 2D array

# Flattening the layers
```

```
classifier.add(Flatten())    #2D array to 1D array

# Adding a fully connected layer
classifier.add(Dense(units=128, activation='relu')) #units is the no of neurons
classifier.add(Dense(units=6, activation='softmax')) # softmax for more than 2
# here 6 is the no of class ,those are 0,1,2,3,4,5.
# Compiling the CNN

classifier.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) # categorical_crossentropy for more than 2



# Step 2 - Preparing the train/test data and training the model

# Code copied from - https://keras.io/preprocessing/image/
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory('data/train',
                                                 target_size=(64, 64),
                                                 batch_size=5,
                                                 color_mode='grayscale',
                                                 class_mode='categorical')

test_set = test_datagen.flow_from_directory('data/test',
                                            target_size=(64, 64),
                                            batch_size=5,
                                            color_mode='grayscale',
                                            class_mode='categorical')
classifier.fit_generator(
        training_set,
```

```
    steps_per_epoch=600, # No of images in training set
    epochs=10,
    validation_data=test_set,
    validation_steps=30)# No of images in test set


# Saving the model
model_json = classifier.to_json()
with open("model-bw.json", "w") as json_file:
  json_file.write(model_json)
classifier.save_weights('model-bw.h5')
```

An explanation for the above code:

1. Importing the Keras library in Python makes it pretty simple to build a CNN.
2. The model type that we will be using is Sequential. Sequential is the easiest way to build a model in Keras. It allows you to build a model layer by layer.
3. We use the 'add()' function to add layers to our model.
4. Classifier Is a neural network adding the convolutional layers into it.
5. Our Convolutional layer having 32 filters and (3,3) is the size of the convolution window.
6. Add a second layer without input shape and use a flatten function to change the 2D array to 1D array. Here we use 1D array.
7. As you can observe, like any other CNN our model consists of a couple of Conv2D and MaxPooling layers followed by some fully connected layers (Dense).
8. Next, prepare to train , This is taken from keras documentation.It basically takes the image and add some shear to it and add zoom to it.Then take the images and feed it to neural networks.
9. Next step to training the model using the fit_generator method then passing the training set .
10. steps_per_epoch=600 is the no of images in the training set. Here given epochs is 10 .
11. After training is started we can save the model file and also save the weights.
12. Then these models and weights are loaded into the predict code for predicting.

Predict data:

Code:

```
import numpy as np
from keras.models import model_from_json
import operator
import cv2
import sys, os

# Loading the model
json_file = open("model-bw.json", "r")
model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(model_json)
# load weights into new model
loaded_model.load_weights("model-bw.h5")
print("Loaded model from disk")

cap = cv2.VideoCapture(0)

# Category dictionary
categories = {0: 'ZERO', 1: 'ONE', 2: 'TWO', 3: 'THREE', 4: 'FOUR', 5: 'FIVE'}

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # 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)
    # Batch of 1
    result = loaded_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)

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


cap.release()
cv2.destroyAllWindows()
```

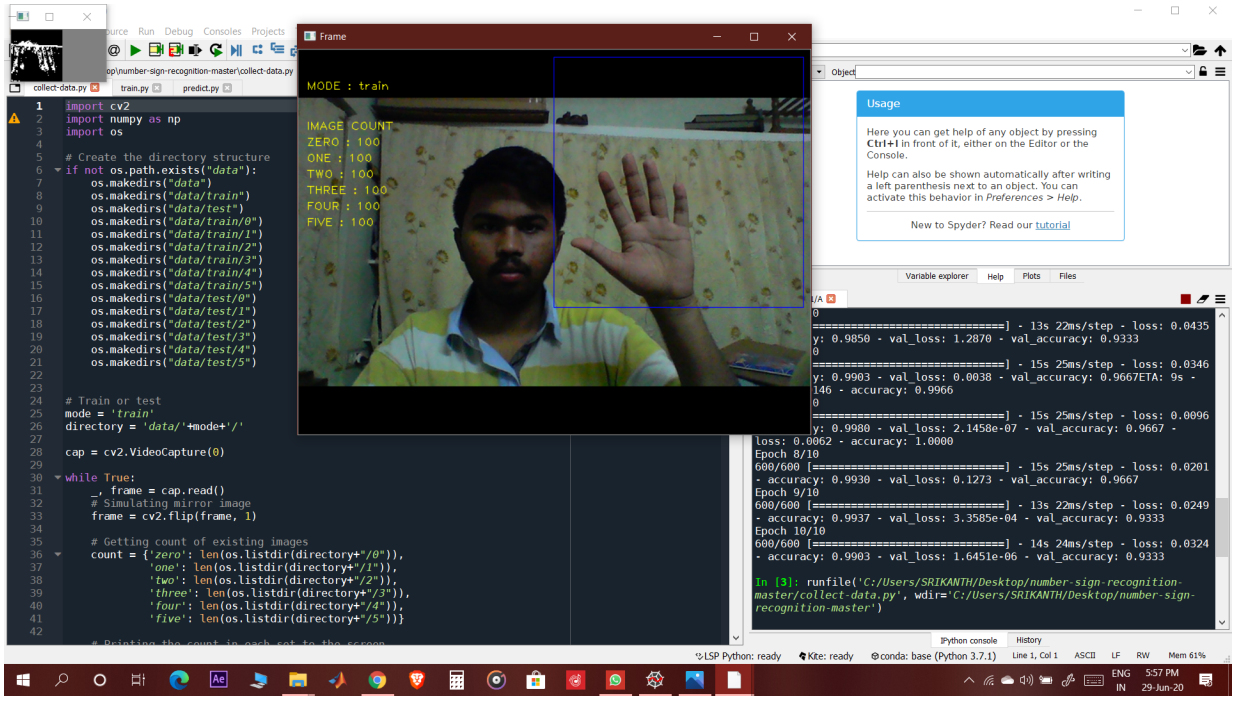
An explanation for the above code:

How it works :

1. First import all the libraries that we need here and then load the model .
2. loaded_model = model_from_json(model_json) is using the model from json method in keras.models
3. Then we load our weights into a new model .
4. Like collect data , capture the object and define the frame with coordinates and it makes like a rectangle to take the image.
5. Then take an image from the (ROI) to the model for prediction .
6. Then extract and resize and convert it into black and white ,then threshold the data like collect data .Because prediction based on train data images .

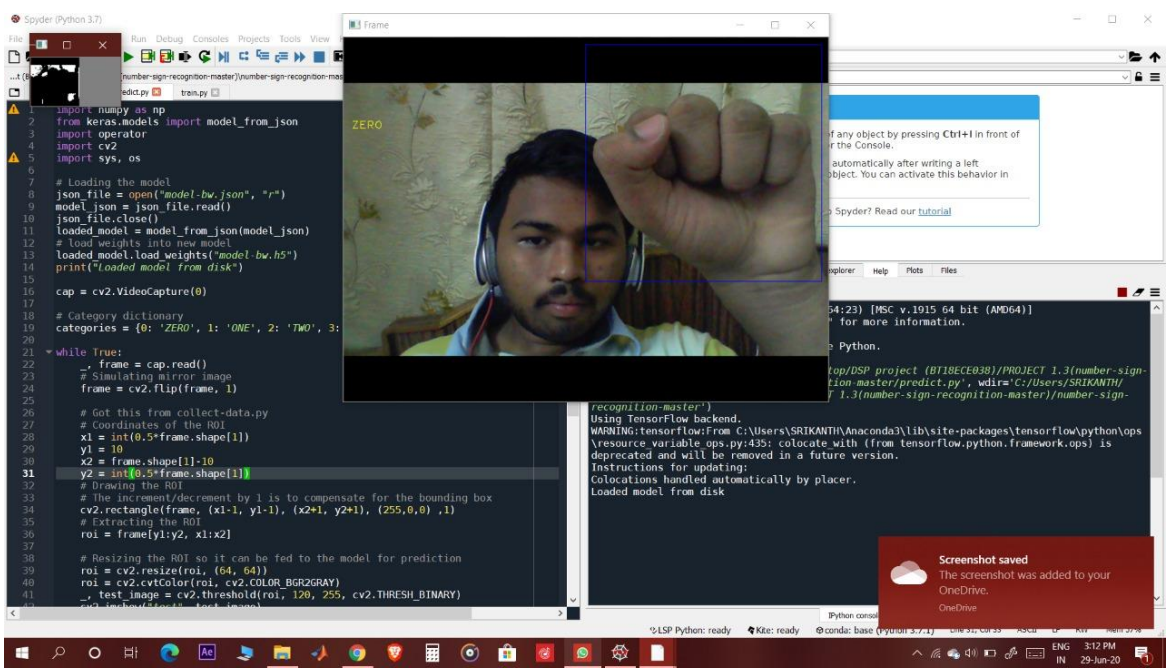
Project Testing :

Train :



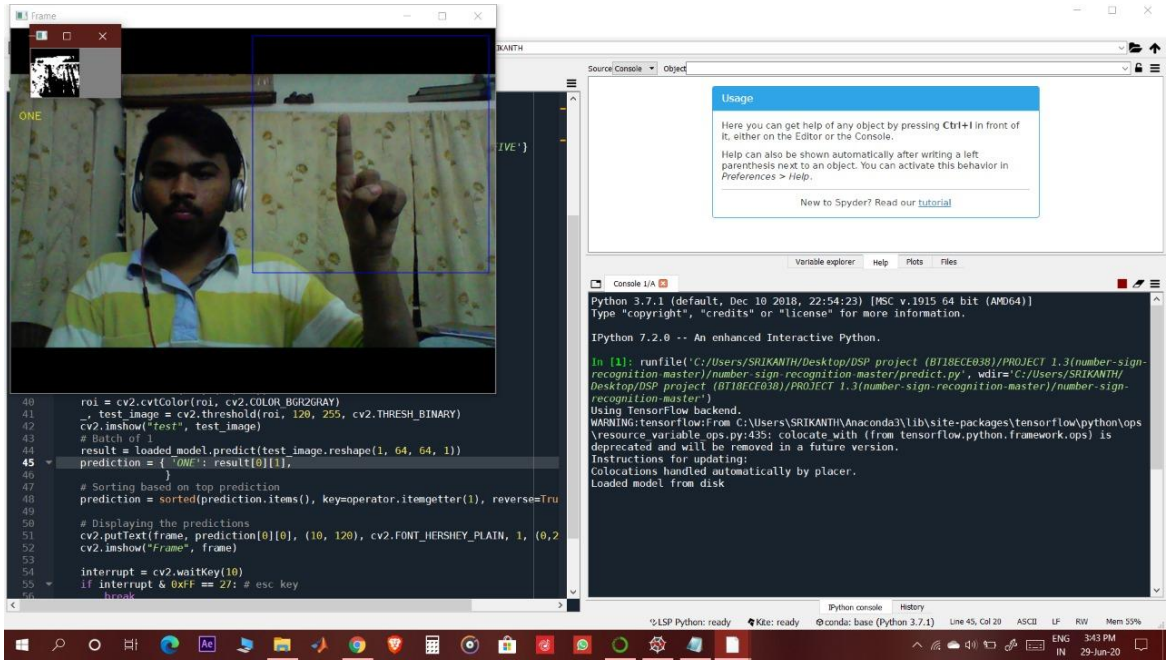
Predict :

For zero:

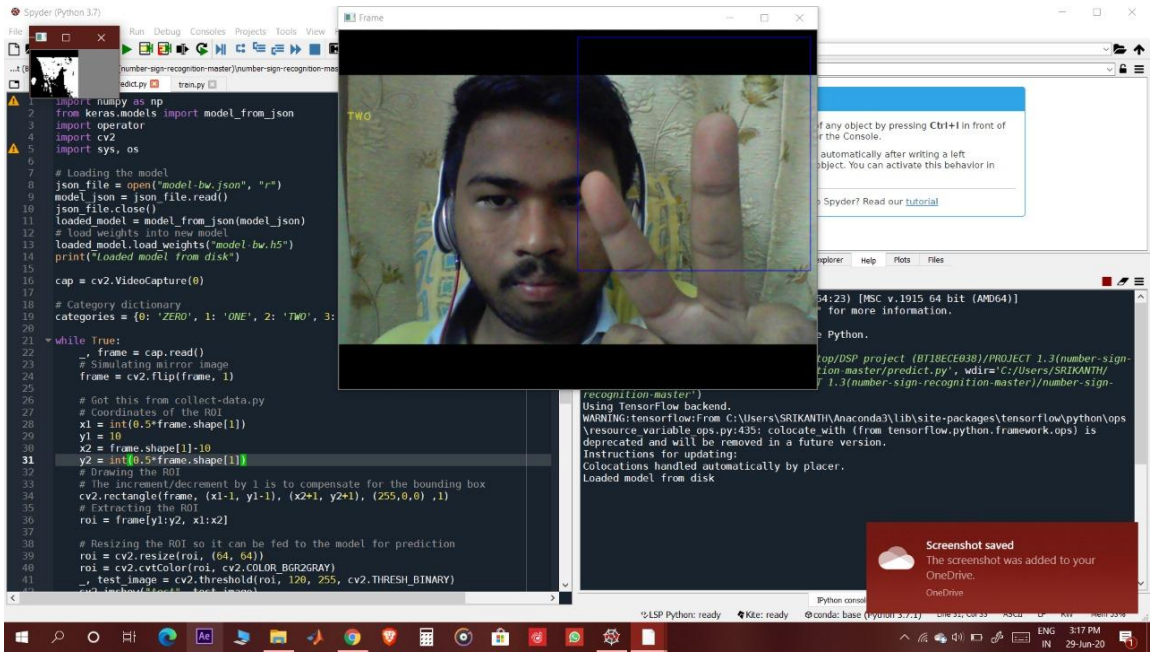


➢ We can see the output at top left corner as ZERO

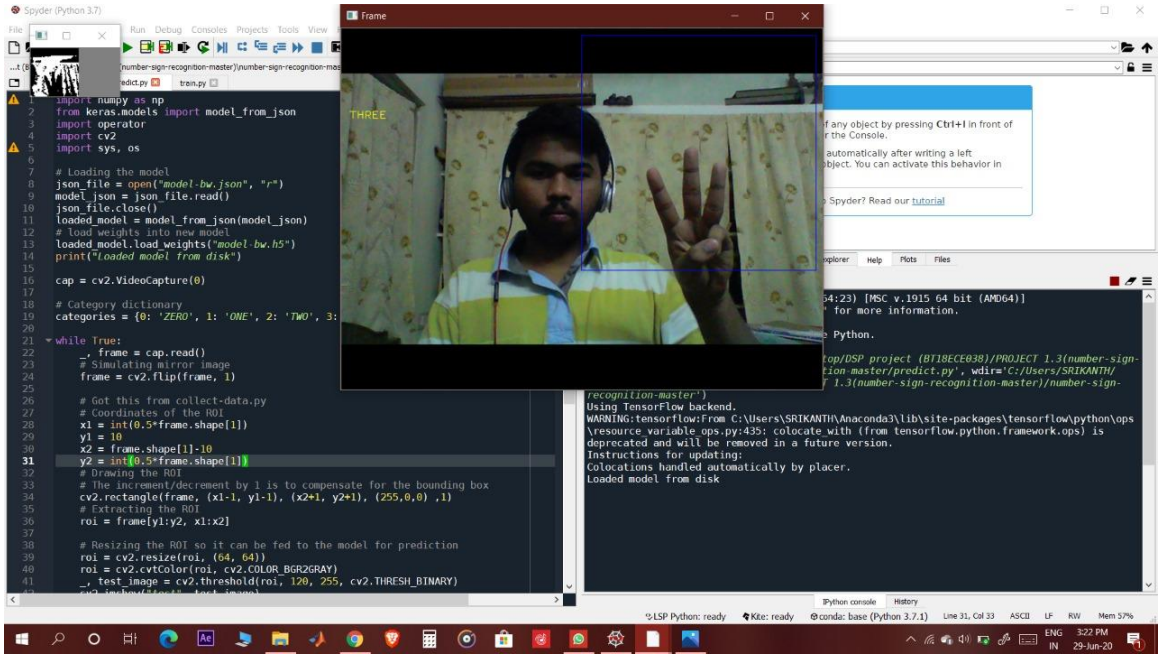
For One :



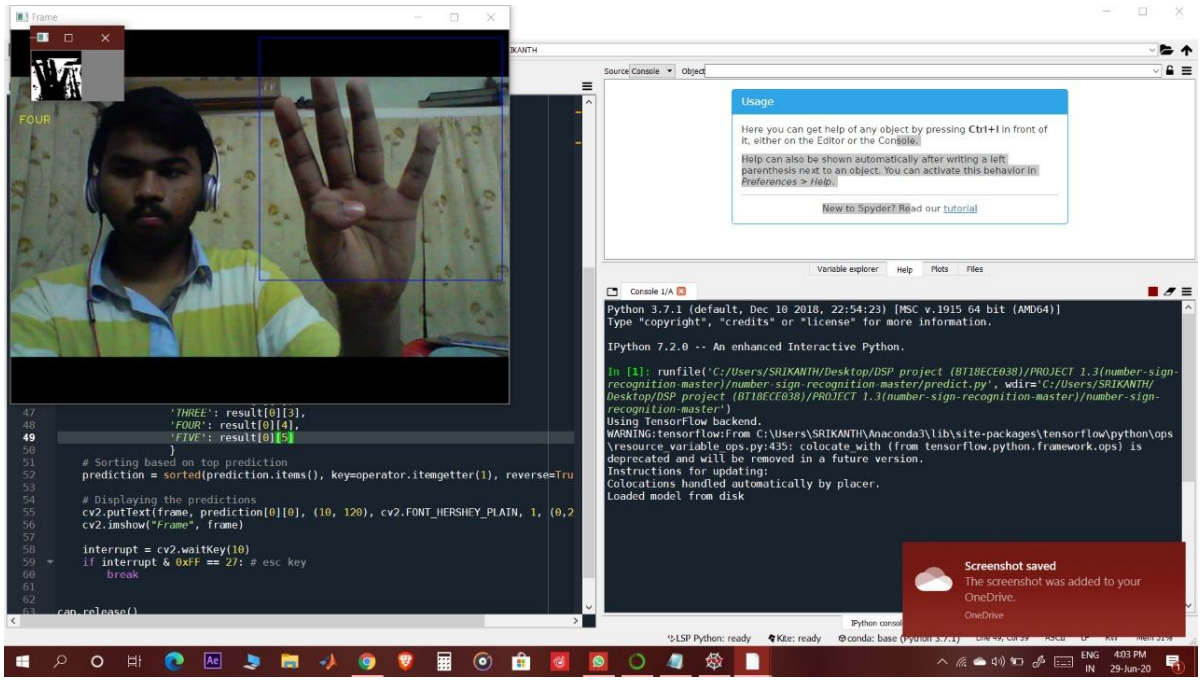
For Two:


Frame
TWO
Screenshot saved
The screenshot was added to your OneDrive.
OneDrive

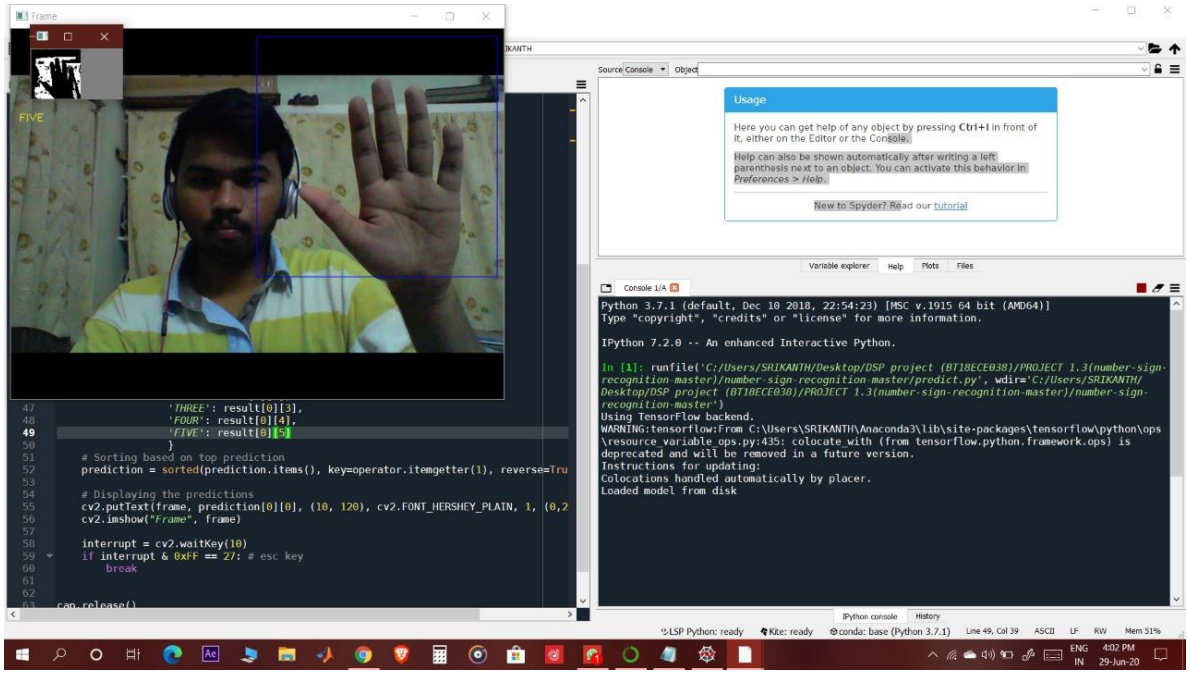
For Three:


Frame
THREE

For Four :



For Five:



- **Small demo video link of my project :**

  https://drive.google.com/file/d/1vBRUSgI7Nmj0kpRNKbUIHuXv3hYxjjKE/view?usp=sharing

Problems that we faced while doing the project :

1) The current problems we faced were unable to access our web cam and some web cam problem issues .
2) Some problem libraries importing issues.
3) And also we feel some difficulty installing keras library in spyder software
4) We are able to run the code without any errors but output is not accurate as we want after that to avoid this problem we have to train and test the code more times.
5) Finally now we are done with project without any errors and good accuracy

Applications areas of hand gestures system :

Hand gestures recognition system has been applied for different applications on different domains, as mentioned in including; sign language translation, virtual environments, smart surveillance, robot control, medical systems etc. overview of some hand gesture application areas are listed below

A. Sign Language Recognition:

Since the sign language is used for interpreting and explanations of a certain subject during the conversation, it has received special attention A lot of systems have been proposed to recognize gestures using different types of sign languages For example recognized American Sign Language ASL using boundary histogram, MLP neural network

and dynamic programming matching. recognized Japanese sign language JSL using Recurrent Neural Network, 42 alphabet and 10 words. recognized Arabic Sign language ArSL using two different types of Neural Network, Partially and Fully Recurrent neural Network.

B. Robot Control:

Controlling the robot using gestures considered as one of the interesting applications in this field proposed a system that uses the numbering to count the five fingers for controlling a robot using hand pose signs. The orders are given to the robot to perform a particular task where each sign has a specific meaning and represents different functions. For example, “one”means “move forward”, “five” means “stop”, and so on.

C. Graphic Editor Control:

Graphic editor control system requires the hand gesture to be tracked and located as a preprocessing operation used 12 dynamic gestures for drawing and editing graphic system. Shapes for drawing are; triangle, rectangular, circle, arc, horizontal and vertical line for drawing, and commands for editing graphic systems are; copy, delete, move, swap, undo, and close.

D. Virtual Environments (VEs):

One of the popular applications in gesture recognition systems is virtual environments VEs, especially for communication media systems provided 3D pointing gesture recognition for natural human computer Interaction HCI in a real-time from binocular views. The proposed system is accurate and independent of user characteristics and environmental changes .

E. Numbers Recognition:

Another recent application of hand gesture is recognizing numbers. proposed an automatic system that could isolate and recognize a meaningful gesture from hand motion of Arabic numbers from 0 to 9 in a real time system using HMM.

Real life applications of our project :

Television Control:

Smart gestures control TV which was brought to the market in 2013 by Samsung hub. Hand postures and gestures are used for controlling the Television device In a set of hand gestures are used to control the TV activities, such as turning the TV on and off, increasing and decreasing the volume, muting the sound, and changing the channel using open and close hands .

REFERENCE OF THIS PROJECT :

We have learn about some basics of image processing and how it will work in python with the help of this research papers

Link1:-https://drive.google.com/file/d/1SVu_wV9Llhk3S4pYeFX7hFwTiO37D3eO/view?usp=drivesdk

Link2:-https://drive.google.com/file/d/1SZbwQGotFJG0ntwInkXhB0nFjP6YnXbF/view?usp=drivesdk

Link3:https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Link4:https://keras.io/

Link5:https://machinelearningmastery.com/introduction-python-deep-learning-library-keras/

Submitted by

Gokul kumar(BT18ECE038)

Praveen kumar(BT18ECE054)