**Packages used**

import cv2

import numpy as np

import sys, os

import string

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 , Dropout

from keras.models import model\_from\_json

import operator

**Data collect**

Here we create a folder/directory data that contains 2 diff folders test and train .

# Create the directory structure

if not os.path.exists("data"):

os.makedirs("data")

if not os.path.exists("data/train"):

os.makedirs("data/train")

if not os.path.exists("data/test"):

os.makedirs("data/test")

this is used to create folders named A to Z inside the train and test folder so that it can save the respective images

for i in string.ascii\_uppercase:

if not os.path.exists("data/train/" + i):

os.makedirs("data/train/"+i)

if not os.path.exists("data/test/" + i):

os.makedirs("data/test/"+i)

# train or test mode is changed so the images are saved in the respective folders

mode = 'train'

directory = 'data/'+mode+'/'

minValue = 70

cap = cv2.VideoCapture(0) # opens web camera to capture video or images

interrupt = -1

cap.read() returns a bool (True/False). If frame is read correctly, it will be True. So you can check end of the video by checking this return value.

while True:

\_, frame = cap.read()

# Simulating mirror image

frame = cv2.flip(frame, 1)

# Getting count of existing images in each alphabet

count = {

'a': len(os.listdir(directory+"/A")),

…..

'z': len(os.listdir(directory+"/Z"))

}

# Printing the count in each set to the screen

**Syntax:** cv2.putText(image, text to be printed , org, font, fontScale, color, thickness)

**org:** It is the coordinates of the bottom-left corner of the text string in the image. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

cv2.putText(frame, "MODE : "+mode, (10, 50), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

cv2.putText(frame, "IMAGE COUNT", (10,75 ), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

cv2.putText(frame, "a : "+str(count['a']), (10, 100), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

…..

cv2.putText(frame, "z : "+str(count['z']), (10, 350), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# Coordinates of the ROI

The shape of an image is accessed by img.shape. It returns a tuple of the number of rows, columns, and channels (if the image is color):

x1 = int(0.5\*frame.shape[1])

y1 = 10

x2 = frame.shape[1]-10

y2 = int(0.5\*frame.shape[1])

# Drawing the ROI

#ROI is a bounding box created with  cv2.rectangle() method that is used to draw a rectangle on any image

# The increment/decrement by 1 is to compensate for the bounding box

***Syntax:****cv2.rectangle(image, start\_point, end\_point, color, thickness)*

cv2.rectangle(frame, (220-1, 9), (620+1, 419), (255,0,0) ,1)

# Extracting the ROI

roi = frame[10:410, 220:520]

#starts the camera to capture images

cv2.imshow("Frame", frame)

roi = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY) #convert to gray scale

roi = cv2.GaussianBlur(roi,(5,5),2) #convert to gaussian blur

The function used is **cv2.threshold**. First argument is the source image, which **should be a grayscale image**. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value.

cv2.THRESH\_BINARY\_INV styles of thresholding is decided by the fourth parameter of the function

# cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C : threshold value is the weighted sum of neighbourhood values where weights are a gaussian window.

roi = cv2.adaptiveThreshold(roi,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,cv2.THRESH\_BINARY\_INV,11,2)

cv2.imshow("ROI", test\_image)

This function is very important, without this function cv2.imshow() won’t work properly.

**Parameters: cv2.waitkey(wait time in milliseconds)**

Thus if the wait time is entered as 6000, the picture will be displayed for 6s and then get closed (provided you have cv2.destroyAllWindows() in the script). If you use ‘0’ as the parmater then the image will be displayed for infinite time until you press the esc key.

interrupt = cv2.waitKey(10)

if interrupt & 0xFF == 27: # press escape to close the camera

break

# cv2.imwrite(directory+'0/'+str(count['zero'])+'.jpg', roi)

if interrupt & 0xFF == ord('a'):

cv2.imwrite(directory+'A/'+str(count['a'])+'.jpg', roi)

…….

if interrupt & 0xFF == ord('z'):

cv2.imwrite(directory+'Z/'+str(count['z'])+'.jpg', roi)

# cv2.destroyAllWindows()

This method destroys (in other words “closes”) all the windows created using the opencv methods. If you want to close a specific window, then you can pass the window name as the argument within this function

**Parameters: name of a window opened using opencv (not mandatory)**

Missing to provide the cv2.destroyAllWindows() at the end of the script might make the window opened to crash

cap.release()

cv2.destroyAllWindows()

**TRAIN**

# 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 , Dropout

import os

os.environ["CUDA\_VISIBLE\_DEVICES"] = "1"

sz = 128

# Step 1 - Building the CNN

# Initializing the CNN

Sequential groups a linear stack of layers into a tf.keras.Model.

Sequential provides training and inference features on this model

In **line 1**, we’ve imported Sequential from keras.models, to initialise our neural network model as a sequential network. There are two basic ways of initialising a neural network, either by a sequence of layers or as a graph.

In **line 2**, we’ve imported Conv2D from keras.layers, this is to perform the convolution operation i.e the first step of a CNN, on the training images. Since we are working on images here, which a basically 2 Dimensional arrays, we’re using Convolution 2-D, you may have to use Convolution 3-D while dealing with videos, where the third dimension will be time.

In **line 3**, we’ve imported MaxPooling2D from keras.layers, which is used for pooling operation, that is the step — 2 in the process of building a cnn. For building this particular neural network, we are using a Maxpooling function, there exist different types of pooling operations like Min Pooling, Mean Pooling, etc. Here in MaxPooling we need the maximum value pixel from the respective region of interest.

In **line 4**, we’ve imported Flatten from keras.layers, which is used for Flattening. Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector.

And finally in **line 5**, we’ve imported Dense from keras.layers, which is used to perform the full connection of the neural network, which is the step 4 in the process of building a CNN.

 create an object of the sequential class

classifier = Sequential()

https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

# First convolution layer and pooling

The Conv2D function is taking 4 arguments, the first is the number of filters i.e 32 here, the second argument is the shape each filter is going to be i.e 3x3 here, the third is the input shape and the type of image(RGB or Black and White)of each image i.e the input image our CNN is going to be taking is of a 128x128 resolution and “1” stands for Black and White, , the fourth argument is the activation function we want to use, here ‘relu’ stands for a rectifier function.

classifier.add(Convolution2D(32, (3, 3), input\_shape=(sz, sz, 1), activation='relu'))

we need to perform pooling operation on the resultant feature maps we get after the convolution operation is done on an image. The primary aim of a pooling operation is to reduce the size of the images as much as possible. the key thing to understand here is that we are trying to reduce the total number of nodes for the upcoming layers.

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)))

# Flattening the layers

classifier.add(Flatten())

# Adding a fully connected layer

Dropout is applied between the two hidden layers and between the last hidden layer and the output layer. Again a dropout rate of 40% is used as is a weight constraint on those layers.

classifier.add(Dense(units=128, activation='relu'))

classifier.add(Dropout(0.40))

classifier.add(Dense(units=96, activation='relu'))

classifier.add(Dropout(0.40))

classifier.add(Dense(units=64, activation='relu'))

classifier.add(Dense(units=26, activation='softmax')) # softmax for more than 2

# 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

classifier.summary()

# 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=(sz, sz),

batch\_size=10,

color\_mode='grayscale',

class\_mode='categorical')

test\_set = test\_datagen.flow\_from\_directory('data/test',

target\_size=(sz , sz),

batch\_size=10,

color\_mode='grayscale',

class\_mode='categorical')

classifier.fit\_generator(

training\_set,

steps\_per\_epoch=3200, # No of images in training set #3181

epochs=5,

validation\_data=test\_set,

validation\_steps=1270)# No of images in test set #1263

# Saving the model

model\_json = classifier.to\_json()

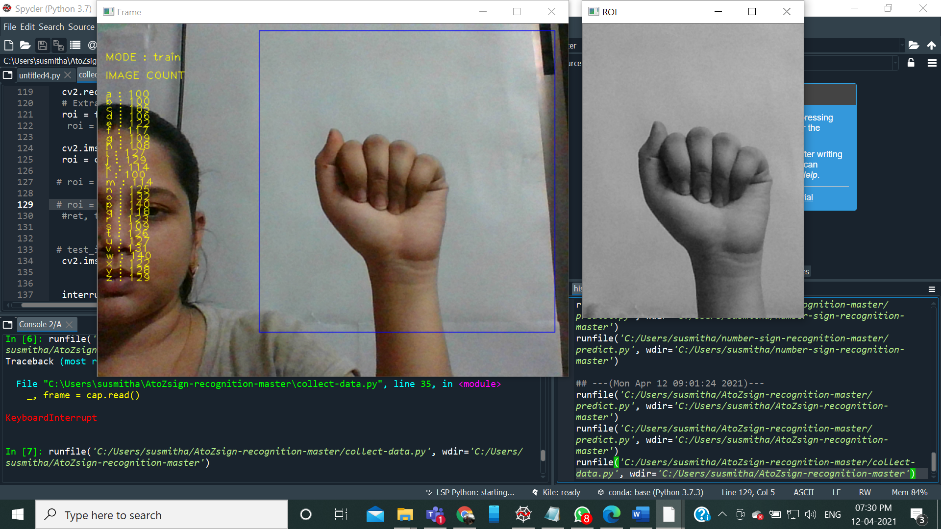
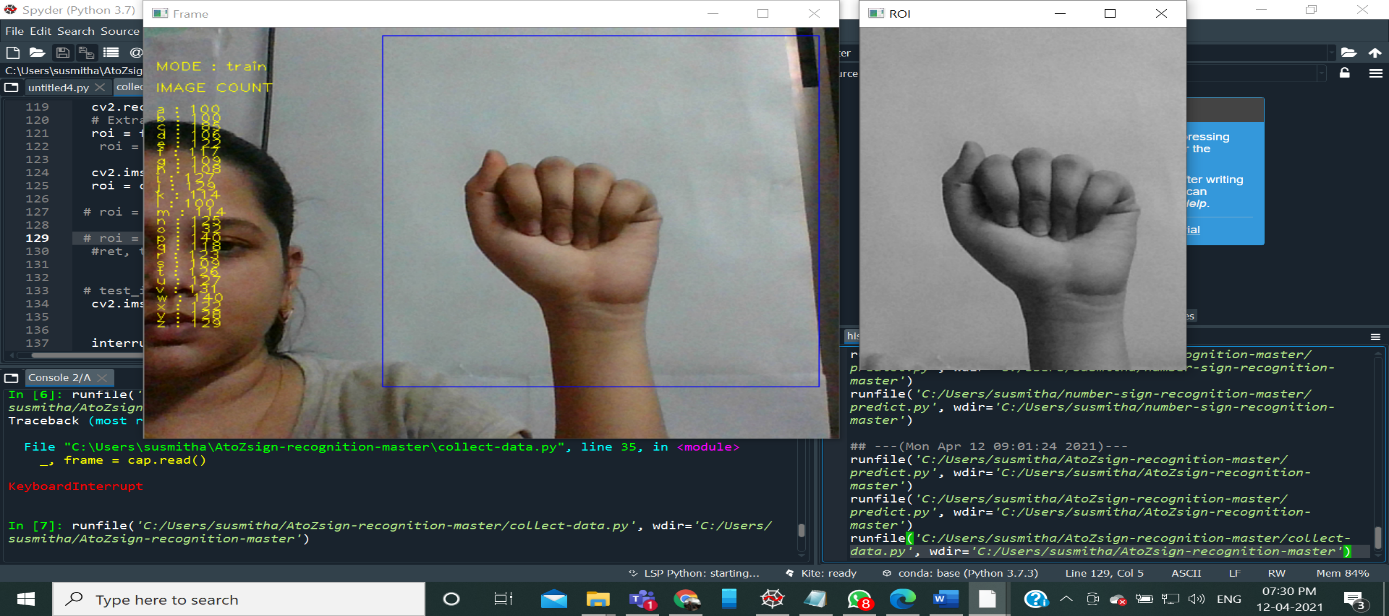
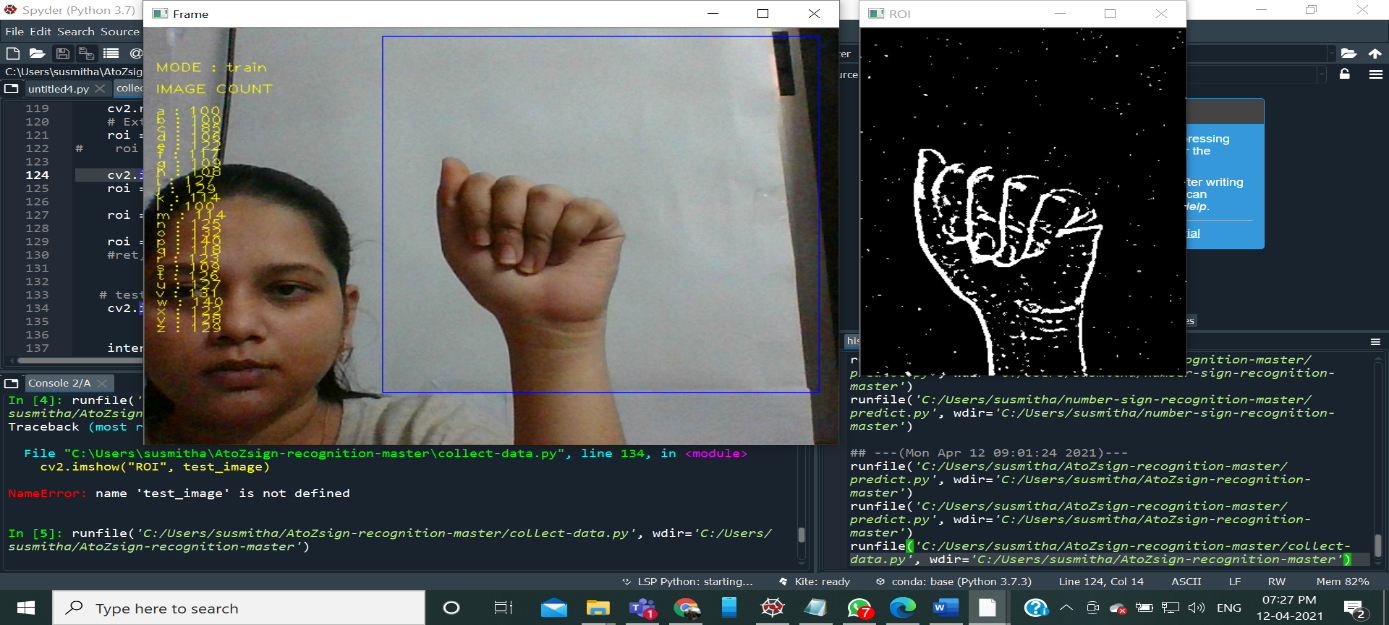
with open("model-bw.json", "w") as json\_file:

json\_file.write(model\_json)

print('Model Saved')

classifier.save\_weights('model-bw.h5')

print('Weights saved')

**COLOUR IMAGE (RGB) GRAY SCALE GAUSSIAN BLUR (B&W)**

**PREDICT:**

**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: 'A', 1: 'B', 2: 'C', 3:'D', 4: 'E', 5: 'F', 6: 'G' , 7:'H', 8:'I' , 9:'J' ,**

**10:'K' , 11:'L',12: 'M', 13: 'N', 14: 'O', 15:'P', 16:'Q', 17:'R', 18:'S',19:'T',**

**20:'U', 21:'V', 22:'W', 23: 'X', 24:'Y', 25:'Z'}**

**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, (220-1, 9), (620+1, 419), (255,0,0) ,1)**

**# Extracting the ROI**

**roi = frame[10:410, 220:520]**

**# Resizing the ROI so it can be fed to the model for prediction**

**roi = cv2.resize(roi, (128, 128))**

**roi = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)**

**roi = cv2.GaussianBlur(roi,(5,5),2)**

**roi = cv2.adaptiveThreshold(roi,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,cv2.THRESH\_BINARY\_INV,11,2)**

**#ret, test\_image = cv2.threshold(roi, 120, 255, cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)**

**cv2.imshow("ROI", roi)**

**# Batch of 1**

**result = loaded\_model.predict(roi.reshape(1, 128, 128, 1))**

**prediction = {'A': result[0][0],**

**'B': result[0][1],**

**'C': result[0][2],**

**'D': result[0][3],**

**'E': result[0][4],**

**'F': result[0][5],**

**'G': result[0][6],**

**'H': result[0][7],**

**'I': result[0][8],**

**'J': result[0][9],**

**'K': result[0][10],**

**'L': result[0][11],**

**'M': result[0][12],**

**'N': result[0][13],**

**'O': result[0][14],**

**'P': result[0][15],**

**'Q': result[0][16],**

**'R': result[0][17],**

**'S': result[0][18],**

**'T': result[0][19],**

**'U': result[0][20],**

**'V': result[0][21],**

**'W': result[0][22],**

**'X': result[0][23],**

**'Y': result[0][24],**

**'Z': result[0][25]}**

**# 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.putText(frame, prediction[0][0], (10, 120),cv2.FONT\_HERSHEY\_SIMPLEX, 2, (0,0,255), 1, cv2.LINE\_AA)**

**cv2.imshow("Frame", frame)**

**interrupt = cv2.waitKey(10)**

**if interrupt & 0xFF == 27: # esc key**

**break**

**cap.release()**

**cv2.destroyAllWindows()**

**reff**

The Sequential from keras.modelsis used to initialise our neural network model as a sequential network.

Conv2D from keras.layers performs the convolution operation i.e the first step of a CNN, on the training images. Since we are working on images here, which a basically 2 Dimensional arrays, we’re using Convolution 2-D

MaxPooling2D from keras.layers, is used for pooling operation, step — 2 in the process of building a cnn. For building this particular neural network, we are using a Maxpooling function, because we need the maximum value pixel from the respective region of interest.

Flatten from keras.layers, which is used for Flattening, it is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector.

And finally Dense from keras.layers is used to perform the full connection of the neural network, which is the step 4 in the process of building a CNN.