**In this tutorial, I will explain how to train the handwritten digits in python.**

**There are many algorithms such as CNN, Tensorflow keras, SVM, KNN and etc to train the model.**

**I have used deep learning algorithm-Tensorflow Keras to train the model.**

**In order to train the model, we must make the training data. (Pre-processing Step)**

**Here, let consider how to make the training data.**

**Firstly, I have prepared R.A. number images in images folder to make the training data.**

**There are many images in the image folder.**

**Read all images in the images folder.**

**And sort their name with respect to the alphabetical order.**

**Based on the sorted name, create image list.**

**Please look at the following:**

types = '\*.png'

image\_list = []

path = "images/"

for files in types:

image\_list.extend(glob.glob(os.path.join(path, files)))

def atoi(text):

return int(text) if text.isdigit() else text

def natural\_keys(text):

return [atoi(c) for c in re.split('(\d+)', text)]

image\_list.sort(key=natural\_keys)

**In the next step, we must extract letters in each image.**

**For it, convert rgb image to binary one.**

image = cv2.imread(image\_list[i])

im = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Normalize and threshold image

res, img = cv2.threshold(im, 150, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

**The next step shows how to remove the border line and other noises.**

# Fill everything that is the same colour (black) as top-left corner with white

cv2.floodFill(img, None, (0, 0), 255)

# Fill everything that is the same colour (white) as top-left corner with black

cv2.floodFill(img, None, (0, 0), 0)

kernel = np.ones((3, 3), np.uint8)

dilate = cv2.dilate(img, kernel, iterations=1)

kernel = np.ones((3, 3), np.uint8)

opening = cv2.morphologyEx(dilate, cv2.MORPH\_OPEN, kernel)

closing = cv2.morphologyEx(opening, cv2.MORPH\_CLOSE, kernel)

**In the next step, it shows how to extract the letters.**

# find all your connected components (white blobs in your image)

nb\_components, output, stats, centroids = cv2.connectedComponentsWithStats(closing, connectivity=8)

sizes = stats[1:, -1]

nb\_components = nb\_components - 1

# minimum size of particles we want to keep (number of pixels)

# here, it's a fixed value, but you can set it as you want, eg the mean of the sizes or whatever

min\_size = 120

# your answer image

temp = np.zeros(output.shape, dtype=np.uint8)

# for every component in the image, you keep it only if it's above min\_size

for j in range(0, nb\_components):

if sizes[j] >= min\_size:

temp[output == j + 1] = 255

# find contours

\_, contours, hierarchy = cv2.findContours(temp.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# sort contours

sorted\_contours = sorted(contours, key=lambda ctr: cv2.boundingRect(ctr)[0])

for k, ctr in enumerate(sorted\_contours):

Num += 1

# Get bounding box

x, y, w, h = cv2.boundingRect(ctr)

letter = temp[y:y+h, x:x+w]

roi = cv2.resize(letter, (14, 28), cv2.INTER\_AREA)

data.append([roi])

Where, data is the training data.

**Based on the extracted letter images, please create labels.txt.**

**Here, we must manually make the label.**

**Please look at train\_labels.txt in the labels folder.**

**We have prepared the training data from the above step.**

**In the next step, it shows how to train the numbers using tensorflow keras.**

**Please create the training and test data to train and validate the model.**

train\_label\_data = open("labels/train\_labels.txt", "r")

trainList = []

for line in train\_label\_data:

trainList.append(line)

labels = [x for v in trainList for x in v.rstrip().split(" ")]

test\_split = 0.1

idx = int(len(data) \* (1 - test\_split))

X\_train, y\_train = np.array(data), np.array(labels)

X\_test, y\_test = np.array(data[idx:]), np.array(labels[idx:])

# --------------------------- Training handwritten digits using mnist data set -------------------

# reshape to be [samples][pixels][width][height]

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

**It is necessary to normalize the image in order to improve the accuracy.**

**So, I have normalized the image.**

# normalize inputs from 0-255 to 0-1

X\_train = X\_train / 255

X\_test = X\_test / 255

# one hot encode outputs

y\_train = np\_utils.to\_categorical(y\_train)

y\_test = np\_utils.to\_categorical(y\_test)

num\_classes = y\_test.shape[1]

**Create Tensorflow keras model.**

# define the larger model

def create\_model():

# create model

models = Sequential()

models.add(Conv2D(30, (5, 5), input\_shape=(1, 28, 14), activation='relu'))

models.add(MaxPooling2D(pool\_size=(2, 2)))

models.add(Conv2D(15, (3, 3), activation='relu'))

models.add(MaxPooling2D(pool\_size=(2, 2)))

models.add(Dropout(0.2))

models.add(Flatten())

models.add(Dense(128, activation='relu'))

models.add(Dense(50, activation='relu'))

models.add(Dense(num\_classes, activation='softmax'))

# Compile model

models.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

return models

**Train the data using the created model.**

# build the model

model = create\_model()

**# Fit the model**

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=300, batch\_size=200, verbose=2)

**Evaluate the model**

# Final evaluation of the model

scores = model.evaluate(X\_test, y\_test, verbose=0)

print("Large CNN Error: %.2f%%" % (100-scores[1]\*100))

**Save the model and weights to json and h5 files.**

# Exciting! we have trained our model

# Saving model weights for later use

model.save("model.h5")

print("model weights saved in model.h5 file")

# Saving model information in .json file

model\_json = model.to\_json()

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

json\_file.write(model\_json)

print("model saved as model.json file")