This project consists of three python scripts such as AutoRecognize.py, PDF2JPG.py and mnist\_test.py.

Where, main script is AutoRecognize.py.

And PDF2JPG.py is to convert PDF to jpg.

PDF2JPG.py works well on Windows and Ubuntu OS.

mnist\_test.py is also loaded with trained weights and models using keras and Tensorflow.

Let consider AutoRecognize.py.

Please look at line 9 and 10.

pdf\_path = 'PDF/p1.pdf'

folder = 'Images/'

pdf\_path displays the path of the current pdf file.

folder displays the path to store the converted jpg files.

Please look at line 13.

pdf\_jpg = PDFJPGConvert(pdf\_path, folder)

This calls PDFJPGConvert class in PDF2JPG.py.

PDFJPGConvert class converts PDF to jpg file.

class PDFJPGConvert:

def \_\_init\_\_(self, pdf\_path, image\_folder):

self.path = pdf\_path

self.folder = image\_folder

The above script displayes path of pdf and converted jpg files.

Please look at line 14 in AutoRecognize.py.

students = pdf\_jpg.main\_convert()

This calls main\_convert function in PDF2JPG.py.

def main\_convert(self):

This function is to convert PDF to jpg.

if not os.path.exists(self.folder):

os.makedirs(self.folder)

with open(self.path, "rb") as file:

file.seek(0)

pdf = file.read()

start\_mark = b"\xff\xd8"

start\_fix = 0

end\_mark = b"\xff\xd9"

end\_fix = 2

i = 0

n\_jpg = 0

while True:

i\_stream = pdf.find(b"stream", i)

if i\_stream < 0:

break

i\_start = pdf.find(start\_mark, i\_stream, i\_stream + 20)

if i\_start < 0:

i = i\_stream + 20

continue

i\_end = pdf.find(b"endstream", i\_start)

if i\_end < 0:

raise Exception("Didn't find end of stream!")

i\_end = pdf.find(end\_mark, i\_end - 20)

if i\_end < 0:

raise Exception("Didn't find end of JPG!")

i\_start += start\_fix

i\_end += end\_fix

print("JPG %d from %d to %d" % (n\_jpg, i\_start, i\_end))

jpg = pdf[i\_start:i\_end]

with open(self.folder + "jpg%d.jpg" % n\_jpg, "wb") as jpgfile:

jpgfile.write(jpg)

n\_jpg += 1

i = i\_end

print(n\_jpg)

Through the above step, pdf converted jpg files and converted jpg files stored in image folder.

The below script is only to crop the specific region such as R.A.Number and Answer.

# -------------------- Read all images in the specified folder -------------------------

images = []

types = '\*.jpg'

for files in types:

images.extend(glob.glob(os.path.join(self.folder, 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)]

images.sort(key=natural\_keys)

**Where, all images are loaded and sorted alphabetically.**

# -------------------- Move the front and back images to their database --------------------

j = 0

for i in range(1, len(images)):

if i % 2 != 0:

j += 1

new\_path = "ImageDB/" + str(j)

if not os.path.exists(new\_path):

os.makedirs(new\_path)

shutil.move(images[i], new\_path)

# -------------------- Crop RA\_Number, Answer images -----------------------------

for k in range(j):

path = "ImageDB/" + str(k+1)

# Select the first image

image1 = os.listdir(path)[0]

img1 = cv2.imread(path + '/' + image1)

ra\_img = img1[1400:2460, 3070:3213, :]

cv2.imwrite(os.path.join(path, 'RA\_Number.png'), ra\_img)

ans1\_img = img1[670:1070, 200:850, :]

cv2.imwrite(os.path.join(path, 'Answer1.png'), ans1\_img)

# Select the second image

image2 = os.listdir(path)[2]

img2 = cv2.imread(path + '/' + image2)

ans2\_img = img2[3947:4324, 2535:3210, :]

cv2.imwrite(os.path.join(path, 'Answer2.png'), ans2\_img)

return j

**Through the above steps, all images are converted jpg files, cropped the specific regions and stored to the specific directory.**

**In the AutoRecognize.py, line 57 will be executed as the next step.**

**This part is important in the project.**

def main():

loaded\_model = mnist\_test.model()

The above script calles mnist\_test function.

**loaded\_model loads the trained weights and model using Keras and Tensorflow.**

for i in range(students):

**This part works preprocessing and recognition steps for each image.**

image\_list = ['RA\_Number.png', 'Answer1.png', 'Answer2.png']

for jj in range(3):

**This part is to read, preprocess and recognize for each specific region such as R.A and Answers.**

image\_path = 'ImageDB/' + str(i + 1) + '/' + image\_list[jj]

**This shows the path stored the specific region image.**

image = cv2.imread(image\_path)

**Read the image**

image = rotate(image, 90)

**Rotate the image.**

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

**Convert rgb to gray image.**

# Normalize and threshold image

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

**Convert gray to binary image.**

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

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

**This part converts the specific region to black color.**

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

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

# morphology operation

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

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

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

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

**In the above steps, we remove the noise and extract the letters.**

# remove border

closing = clear\_border(closing, 5)

**Remove the border.**

# 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

**This is to extract each letter.**

# 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

**Crop the letter’s area.**

# 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])

letter\_pos = []

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

# Get bounding box

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

**Crop the each letter.**

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

letter\_pos.append(h)

**Recognize the letters**

**Let consider the cropped area.**

if (k > 0) and (h < 0.7 \* letter\_pos[0]):

**If it satisfied the above condition, it is comma.**

ans1 = '.'

else:

**Else, we must recognize the extracted letter.**

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

new\_image = roi.flatten()

**In order to match with the trained model, reshape the image to the following:**

new\_image = new\_image.reshape(1, 1, 28, 14)

**Recognize the extracted letter using trained model and weights.**

ans1 = loaded\_model.predict(new\_image)

ans1 = ans1.tolist()

ans1 = ans1[0].index(max(ans1[0]))

**Display the recognized letter on the image.**

cv2.putText(image, str(ans1), (x, y),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

cv2.imshow('img', image)

cv2.waitKey(0)