CSE1014 - Foundations of AI

Report Paper

A project report titled

Score Aggregation Using Image Recognition

By

19BAI1089	Subramanian Venkittanarayanan
19BAI1098	Aaditya Hemant

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted to

Prof. Vijayalakshmi R

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

June 2021

I. Abstract

In the last year, people have been forced to adjust to a sudden online only format for various jobs, one of the affected areas being education. Our goal with this project is to create a functional system which can take a corrected student marksheet image and retrieve the marks that have been allotted to the student by accurately segmenting the image into relevant portions and only identifying numbers written by the teacher.

Utilising the machine learning algorithm to recognize handwritten numbers, we can obtain the scores assigned to each question, total them, and provide that result as the output. Storing an object of the resultant scores in an appropriate file format would then provide us with quick and easy data access if it is required sometime in the future. In order to train the digit recognizer, we will use the MNIST database of handwritten digits which contains 60,000 training images and 10,000 testing images.

In order to train the handwriting to text model, we will be using “A-Z Handwritten Alphabets in .csv format” which contains 370000+ images of the English alphabet. While scanning the images, we will be looking at specific characteristics to help us differentiate normal numbers from numbers that are to be considered for grading. These may include numbers written using a red pen, or enclosed inside a circle.

Using OpenCV, Keras, and Tensorflow, we hope to achieve a model that can accurately and efficiently identify the marks assigned, and calculate the total score in order to improve the usability and appeal of online learning and examination platforms.

II. Introduction

With the world continuing to feel the consequences caused by the pandemic, there has been a massive shift towards using online media to continue students' education. Classes are conducted through online platforms such as Microsoft Teams, Google Classrooms, Zoom. Students also give their examinations online. This is done by uploading their handwritten answers in the form of pdfs, which teachers can then review, score, and return to the students. Streamlining this process by adding features to help teachers calculate the final score automatically once they have assessed the questions individually would speed up the process of grading students.

Our aim is to create a working model to get marks allotted to a student's answers and store them in an array. We will be giving importance to identifying a mark regardless of where it is on the image, i.e. at the top of the image or on the sides, and whether the number is overlapping other written text. It is not possible to simply use OCR to detect all the numbers in the image as it leads to complications regarding the numbers origins. We make use of OpenCV which is an open source computer vision library. OpenCV provides various features that will assist in image processing and analysis for our project in real time enabling our project to be able to provide quick results from input images. To ensure that we detect only allotted marks and not any numbers part of an answer we must first use document segmentation to separate parts of the document that we need.[https://link.springer.com/referenceworkentry/10.1007%2F978-0-85729-859-1_5#:~:text=The%20most%20common%20way%20is,black-and-white%20image.] [https://link.springer.com/referencework/10.1007/978-0-85729-859-1]

Firstly the image must be segmented into a foreground and background where the foreground only contains features that we are interested in. Secondly we must segregate objects in the foreground based on connected components and shapes. For this purpose we looked at various features that separate marks from numbers appearing in a student's answers, such as different colouring, different font size, outlining or encircling. Colouring here proved to be the best separator and with high contrast in colour we could successfully highlight necessary features. As marks are usually written using red ink, using a red filter to create a colour mask to apply on the image gave the best results. These red filters can be created by applying a colour mask using an HSV representation of colours.

There exists the possibility that a student has written two complete answers in one page resulting in multiple marks. In such cases further object segmentation is required. We will need to detect and split features that belong to one mark from another. Canny Edge detection along with Hough Circle Transforms can be used to separate them. Canny Edge detection uses a multi stage algorithm to group pixels in the same direction and detect edges. Hough Circle Transform is a feature extraction technique used for detecting circles in images. This is done by first detecting the various edges and then finding possible radii around points of the edges. If marks are encircled we can use it to easily segregate encircled features. Once circles are detected we can use the generated coordinates to split them up and pass it into a function that can check if it is of appropriate form. This process would form various circles some of which were irrelevant. One characteristic feature of the circles that best covered the required mark was that their coordinates would closely resemble a square. By retaining only these squared segments we could get what we need.

Once images were segregated into only the most necessary and relevant parts we could pass them into a trained model to detect the number itself. For our purposes we will make use of three independently trained model. One trained on the Mnist Handwritten numbers dataset, another handwritten numbers dataset obtained from kaggle and a third custom made dataset. The custom made dataset is generated so that it includes encircled numbers. This way we can account for the various different types of ways our features may be extracted. We will get predictions for our extracted features from each of these models and combine their results using softmax averages or max voting to get an output. This output can then be stored into any required format and used as and when necessary.

III. Methodology

The basic process flow involved in our project consists of:

1. Digit recognition model creation & training
2. Importing the image and applying filters
3. Performing circle detection
4. Fine-tuning the image before passing it through the model
5. Obtaining the prediction and adjusting the model/image accordingly

In order to approach the problem of digit recognition on an answer sheet, we must first analyse the possible obstacles our model may face at any phase.

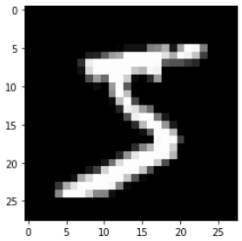
For creating and training a handwritten digit recognition model, we have the readily available, widely used MNIST dataset [http://yann.lecun.com/exdb/mnist/], which consists of 60,000 images, each 28x28 in size, containing a handwritten number from 0 to 9. However, one drawback of this model is that we will need to shrink our input images to 28x28 resolution, which can result in loss of detail, worsening our predictions.

Our primary method to differentiate between the text and numeric values a student has written and the actual number representing the marks awarded by the teacher involves using a combination of red-filtering and circle detection. We felt that this idea aligned with how grading is done normally, through the use of a red pen. By also encircling the score, we can improve the detection of the mark considerably. However, the problems here arise from the contrasts between the intensity of red pixels in the image, which may be a consequence of varying lighting conditions, ink thickness, or picture quality.

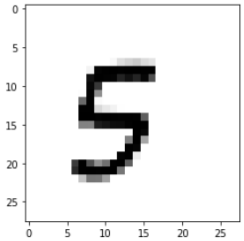
Finally, when it comes to passing the image of a cropped digit to our model, we again need to address some issues. One being the presence of the circle that was used to identify the number. If by chance, some part of the number coincides with the circle itself, we cannot risk cropping it out. Moreover, the image has to be shrunk, so we need to deal with the loss in quality as well. The MNIST model alone may not be sufficient, since here, the images contain an encircled digit, as opposed to just a number, which may trick the model into skewed results.

In order to solve these problems, we attempted to implement multiple solutions, and explored different methods and their viabilities. To improve prediction accuracy as a whole, and overcome the disadvantages of purely using the MNIST model, we decided we would create two more models, trained on slightly

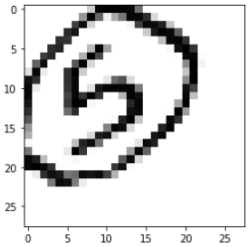
different datasets. One of these models would be trained on a dataset found on Kaggle (https://www.kaggle.com/jcprogjava/handwritten-digits-dataset-not-in-mnist), which consisted of 107,730 handwritten digits, in the same 28x28 format. Finally, we also created a custom dataset (tool found at https://github.com/JC-ProgJava/ Handwritten-Digit-Dataset), which had similar handwritten digits, but enclosed inside a circle. We generated close to 3000 images for each digit, and used them to train a third model that specified in predicting digits with circles around them.



(a) MNIST



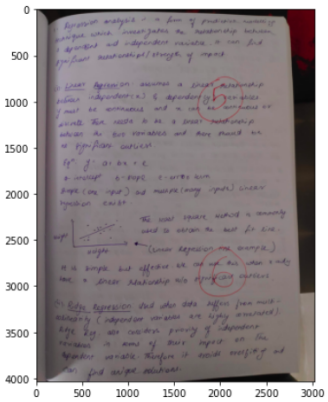
(b) Not-in-MNIST

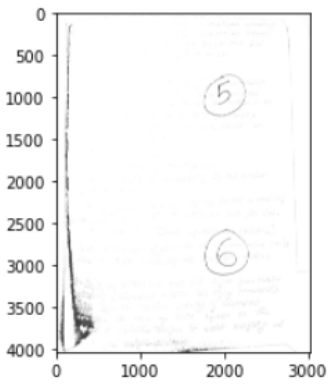


(c) Custom

The models were made using the keras library. They are simple perceptron models, consisting of convolution layers, flattening layers and a dense layer for the output.

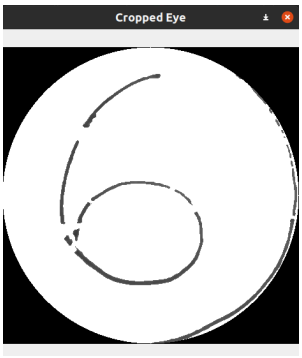
Images can easily be imported using cv2. We then applied a red filter on the image that would capture red values within a specified range (123, 57, 57) to (220,57,57) with a threshold of 35, in order to ensure we capture red text/digits regardless of the pixel intensity. A mask was then applied to extract only the required entities, and then the masked image was converted to grayscale.





(a) Before Image (b) After filtering and mask

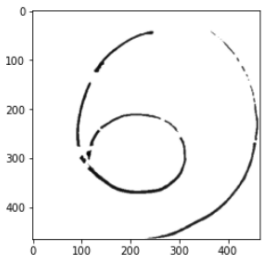
Now, we can see that some of the text is still present in the filtered image. This is where the Hough circle transform comes in handy. We can use cv.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]]), which enables us to find circles in a grayscale image. The hyperparameters here were set based upon trial and error, in order to produce the best set of consistent results.



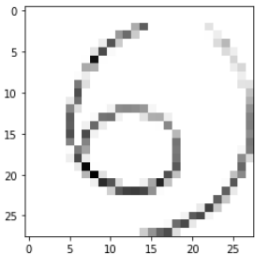
(a) ‘6’ identified

(b) ‘5’ identified

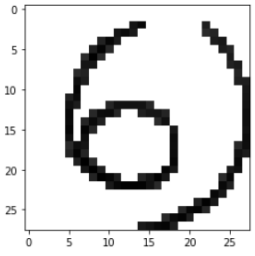
The last step involves just loading the models, and resizing the images before inputting them. However, we still need to address the loss in quality after resizing. This is where we can use cv2.equalizeHist(). Histogram equalization improves the contrast present in an image. This is done through identifying the intensity ranges of pixels in an image, and expanding them in order to apply higher contrast. Doing this helps counter the effects of resolution loss due to resizing.



(a) Original Image



(b) Resized Image



(c) After equalizeHist()

After comparing the results of all 3 models, we found that the pure MNIST model and the model trained on our custom dataset performed reasonably well,

predicting the digit correctly a fair amount of times. The non-MNIST dataset model performed poorly, often producing incorrect results. We collect the softmax values from each of the models and take a weighted sum of them such that model of MNIST which produced better results on average has a higher weight of 0.5 while others have weights of 0.25. From their combined value we can obtain the final result.

Overall, our results were limited by the time-frame provided. With additional modifications, such as fine-tuning the histogram and ensuring hough circle transform grabs only the required portion of the image, we may be able to increase the accuracy of our predictions even further.

IV. Implementation

```
# # Importing Libraries

# In[1]:

import cv2
import numpy as np
import matplotlib.pyplot as plt
import imutils
from keras.datasets import mnist
from keras.layers import Dense, Flatten, Dropout
from keras.layers.convolutional import Conv2D
from keras.models import Sequential
from keras.utils import to_categorical
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix as cm, classification_report
as cr
```

```
from tensorflow import keras
import warnings
warnings.filterwarnings("ignore")

# # Part 1: Model Creation
# ## MNIST Model
# Download and split the MNIST dataset into test and training sets.
# In[2]:
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# In[3]:
plt.imshow(X_train[0], cmap="gray")
plt.show()
print (y_train[0])
# In[4]:
X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)

# Reshaped data from (60000,28,28) to (60000,28,28,1) to be able to use in
our model.
# In[5]:
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
# Converted labels to categorical format so that our model can predict
these values categorically instead of continuously.
# In[6]:
## Declare the model
model = Sequential()
## Declare the layers
layer_1 = Conv2D(32, kernel_size=3, activation='relu', input_shape=(28,
28, 1))
layer_2 = Conv2D(64, kernel_size=3, activation='relu')
layer_3 = Flatten()
layer_4 = Dense(10, activation='softmax')

## Add the layers to the model
model.add(layer_1)
model.add(layer_2)
model.add(layer_3)
model.add(layer_4)
```

```
# Created the model with 4 layers. The first layer is a simple convolution
layer that takes in inputs of shape (28,28,1) with the "1" indicating a
greyscale image.
#
# The flatten layer allows us to link the convolution layer to the dense
layer to obtain an output

# In[7]:
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# In[8]:
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2)
# In[9]:
model.save("digit")
# ## Custom Model without Encircled Digits
# In[2]:
x = []
y = []
# In[3]:
for dir1 in os.listdir('dataset2'):
   if dir1!=".DS_Store":
       for file in os.listdir(os.path.join('dataset2', dir1)):
           image_path= os.path.join('dataset2', dir1,  file)
           image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
           img = 255 - image[:, :, 3]
           x.append(img)
           y.append(int(dir1))
# Here, we are appending images into our list 'x' and their corresponding
labels into our list 'y'
# In[13]:
plt.imshow(x[40000], cmap="gray")
# In[12]:
x = np.array(x)
y = np.array(y)
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.15,
random_state=101)
# We convert the lists into numpy arrays and then perform train-test-split
to use in our model.
# In[13]:
xtrain = xtrain.reshape(91570,28,28,1)
```

```
xtest = xtest.reshape(16160,28,28,1)
ytrain = to_categorical(ytrain)
ytest = to_categorical(ytest)
# Before the arrays can be used in our model, we need to reshape them to
the order of (size, 28,28,1) to feed our model grayscale images.
# In[14]:
## Declare the model
model2 = Sequential()
## Declare the layers
layer_1 = Conv2D(32, kernel_size=3, activation='relu', input_shape=(28,
28,1))
layer_2 = Conv2D(16, kernel_size=3, activation='relu')
layer_3 = Flatten()
layer_4 = Dense(10, activation='softmax')
## Add the layers to the model
model2.add(layer_1)
model2.add(layer_2)
model2.add(Dropout(0.2))
model2.add(layer_3)
model2.add(layer_4)
# The model here is similar to the one used to train the MNIST dataset.
# In[15]:
model2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# In[16]:
model2.fit(xtrain, ytrain, validation_data=(xtest, ytest), epochs=2)
# In[17]:
model2.save("digit2")
# ## Custom Model with Encircled Digits
# This model is trained on numbers that are encircled. We perform the same
steps as we did with the prrevious model in order to create our datasets
and train the model.
# In[23]:
x = []
y = []
image = cv2.imread('circle_dataset/datadown/5/20250.png',
cv2.IMREAD_UNCHANGED)
img = 255 - image[:, :, 3]
plt.imshow(image)
# In[26]:
```

```
for dir1 in os.listdir('circle_dataset/datadown(2)'):
       for file in os.listdir(os.path.join('circle_dataset/datadown',
dir1)):
           image_path= os.path.join('circle_dataset/datadown', dir1,
file)
           image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
           img = 255 - image[:, :, 3]
           x.append(img)
           y.append(int(dir1))


# In[27]:

x = np.array(x)
y = np.array(y)


# In[28]:


xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.15,
random_state=101)
print(xtrain.shape, xtest.shape)


# In[30]:


xtrain = xtrain.reshape(74358,28,28,1)
xtest = xtest.reshape(13122,28,28,1)
ytrain = to_categorical(ytrain)
ytest = to_categorical(ytest)


# In[31]:


## Declare the model
model3 = Sequential()
```

```
## Declare the layers
layer_1 = Conv2D(32, kernel_size=3, activation='relu', input_shape=(28,
28,1))
layer_2 = Conv2D(16, kernel_size=3, activation='relu')
layer_3 = Conv2D(12, kernel_size=3, activation ='relu')
layer_4 = Flatten()
layer_5 = Dense(10, activation='softmax')

## Add the layers to the model
model3.add(layer_1)
model3.add(layer_2)
model3.add(Dropout(0.2))
model3.add(layer_3)
model3.add(Dropout(0.2))
model3.add(layer_4)
model3.add(layer_5)


# In[32]:


model3.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# In[33]:
model3.fit(xtrain, ytrain, validation_data=(xtest, ytest), epochs=2)
# In[34]:
model3.save("digit3")
# # Part 2: Extracting Red Entities from the Image
# First, we will read the image to be scanned.
# In[14]:
original_img = cv2.imread('0003.jpg')
img = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,7))
plt.imshow(img)


# Next, we will apply a red threshold filter on the image, to scan for
pixels in a given rgb range.

# In[15]:
```

```
t = 35 #threshold acceptance
red_thresh = cv2.inRange(img, np.array([123 - t, 57 - t, 57 - t]),
np.array([220 + t, 57 + t, 57 + t]))


# Finally, we will create a mask over the original image, to bring out
only the pixels in the previously defined RGB range.

# In[16]:


combined_mask = red_thresh
combined_mask_inv = 255 - combined_mask


# In[17]:


combined_mask_rgb = cv2.cvtColor(combined_mask_inv, cv2.COLOR_GRAY2RGB)
img = cv2.max(img, combined_mask_rgb)


# In order to pass grayscale images through our model, we will convert the
masked image to a grayscaled version of itself.

# In[18]:


img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
plt.imshow(img, cmap="gray")


# # Part 3: Extracting Circles from the Image

# Once the image has been red-filtered, we need to extract the circular
entities from the image. We do this as an additional step, incase the
```

```
red-filtering produces inconsistencies in the image, such as bringing out
random lines that are not red, or part of the numbers we require.

# For this, we will be using Hough Circle Transform to identify circles in
the image.

# In[19]:


#Mask to crop circles
height,width = img.shape
mask = np.zeros((height,width), np.uint8)


# In[20]:


#Applying Hough Circle Transform
img = cv2.medianBlur(img,5)
circles =
cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,200,param1=100,param2=30,minRadi
us=120,maxRadius=250)


# In[21]:


# To Idenitify Circles in the Image
for i in circles[0,:]:
   i[2]=i[2]+4
   # Draw on mask
   cv2.circle(mask,(i[0],i[1]),int(i[2]),(255,255,255),thickness=-1)


# In[22]:


#to show masked images
masked_data = cv2.bitwise_and(img, img, mask=mask)
```

```
# Apply Threshold
_,thresh = cv2.threshold(mask,1,255,cv2.THRESH_BINARY)

# Find Contour
contours =
cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
crop = []
j = 0
for i in contours[0]:
   x,y,w,h = cv2.boundingRect(i)


# Crop masked_data
   crop.append(masked_data[y:y+h,x:x+w])

#Code to close Window
   cv2.imshow('Cropped Eye',crop[j])
   cv2.waitKey(0)
   cv2.destroyAllWindows()
   j+=1


# Here, we can also see that applying Hough Transform causes the detection
of non-circular objects at times. However, these objects are rarely equal
in both height and width, and thus, this can be used as a way of
distingusihing images with and without circled numbers.

# # Part 4: Digit Recognition

# First we will load our digit recognition models.

# In[95]:


model1 = keras.models.load_model("digit")
model2 = keras.models.load_model("digit2")
model3 = keras.models.load_model("digit3")


# In[98]:
```

```
grey = crop[1]
image = grey
size = len(grey)
for i in range(size):
   for j in range(size):
       if (grey[i][j]==0):
           grey[i][j] = 255


# Here, we have simply converted the black pixels outside the cropped
circle images to white pixels, as to clean up our image before passing it
through the model.

# In[99]:


preprocessed_digits = []
plt.imshow(grey, cmap="gray")
plt.show()
resized_digit=cv2.equalizeHist(grey)
# Resizing that digit to (18, 18)
resized_digit =  imutils.resize(resized_digit, width=28, height=28)

# Adding the preprocessed digit to the list of preprocessed digits
preprocessed_digits.append(resized_digit)

print("\n\n\n----------------Contoured Image--------------------")
np.set_printoptions(threshold=np.inf)
plt.imshow(resized_digit, cmap="gray")
inp = np.array(preprocessed_digits)


# We have resized the original cropped image to a 28x28 size image, in
order for us to be able to pass it through our model.
# collect outputs into the pred variable
pred = []
pred_count = 0
```

```
# ### Model 1

# In[100]:


for digit in preprocessed_digits:
   digit2=cv2.equalizeHist(digit)
   #digit2 = digit
   prediction = model1.predict(digit2.reshape(1, 28, 28, 1))

   print ("\n\n---------------------------------------\n\n")
   print ("=========PREDICTION============ \n\n")
   plt.imshow(digit2.reshape(28, 28), cmap="gray")
   plt.show()
   print("\n\nFinal Output: {}".format(np.argmax(prediction)))


   print ("\nPrediction (Softmax) from the neural network:\n\n
{}".format(prediction))
   pred.append( [i * 0.5 for i in prediction] )
   pred_count += pred_count



   hard_maxed_prediction = np.zeros(prediction.shape)
   hard_maxed_prediction[0][np.argmax(prediction)] = 1
   print ("\n\nHard-maxed form of the prediction: \n\n
{}".format(hard_maxed_prediction))
   print ("\n\n---------------------------------------\n\n")


# ### Model 2

# In[105]:


for digit in preprocessed_digits:
   #digit2=cv2.equalizeHist(digit)
   digit2 = digit
   prediction = model2.predict(digit2.reshape(1, 28, 28, 1))
```

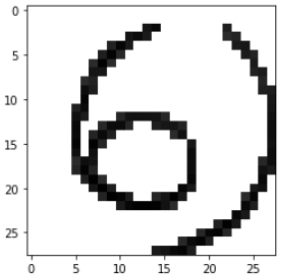
```
   print ("\n\n---------------------------------------\n\n")
   print ("=========PREDICTION============ \n\n")
   plt.imshow(digit2.reshape(28, 28), cmap="gray")
   plt.show()
   print("\n\nFinal Output: {}".format(np.argmax(prediction)))


   print ("\nPrediction (Softmax) from the neural network:\n\n
{}".format(prediction))
   pred.append( [i * 0.25 for i in prediction] )
   pred_count += pred_count


   hard_maxed_prediction = np.zeros(prediction.shape)
   hard_maxed_prediction[0][np.argmax(prediction)] = 1
   print ("\n\nHard-maxed form of the prediction: \n\n
{}".format(hard_maxed_prediction))
   print ("\n\n---------------------------------------\n\n")


# ### Model 3

# In[103]:


for digit in preprocessed_digits:
   digit2=cv2.equalizeHist(digit)
   #digit2 = digit
   prediction = model3.predict(digit2.reshape(1, 28, 28, 1))

   print ("\n\n---------------------------------------\n\n")
   print ("=========PREDICTION============ \n\n")
   plt.imshow(digit2.reshape(28, 28), cmap="gray")
   plt.show()
   print("\n\nFinal Output: {}".format(np.argmax(prediction)))
   pred.append( [i * 0.25 for i in prediction] )
   pred_count += pred_count
```

```
   print ("\nPrediction (Softmax) from the neural network:\n\n
{}".format(prediction))


   hard_maxed_prediction = np.zeros(prediction.shape)
   hard_maxed_prediction[0][np.argmax(prediction)] = 1
   print ("\n\nHard-maxed form of the prediction: \n\n
{}".format(hard_maxed_prediction))
   print ("\n\n----------------------------------------\n\n")

# In[ ]:
#Predict from combined result
pred_f = [0,0,0,0,0,0,0,0,0,0]
n = len(pred)
for i in range(n):
    pred_f += pred[i]/n
final_o = np.argmax(pred_f)
```

V. Result

Model 1

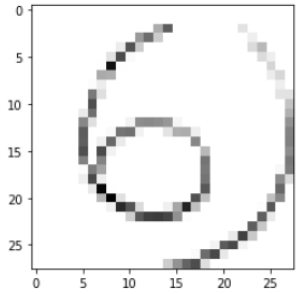
```
=========PREDICTION============
```




```
Final Output: 6

Prediction (Softmax) from the neural network:

 [[1.6190547e-02 1.2721592e-06 1.4404309e-04 3.0059405e-04 9.6848281e-03
  1.6645813e-01 7.7948207e-01 4.9597948e-05 2.6097750e-02 1.5912225e-03]]


Hard-maxed form of the prediction:

 [[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

Model 2

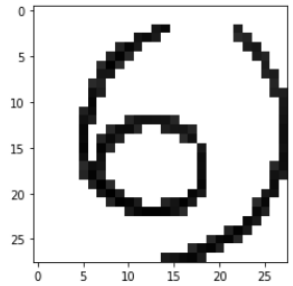
```
=========PREDICTION============
```



```
Final Output: 0

Prediction (Softmax) from the neural network:

 [[0.24160798 0.21271734 0.0723645  0.11439162 0.03695168 0.07204394
  0.10686465 0.03902509 0.08457685 0.01945628]]


Hard-maxed form of the prediction:

 [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Model 3

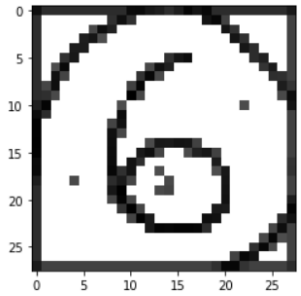
```
=========PREDICTION============
```



```
Final Output: 5

Prediction (Softmax) from the neural network:

 [[1.0749969e-04 8.2470510e-09 6.4078820e-05 4.5867577e-11 5.3301639e-09
  9.8728406e-01 4.4985742e-07 1.2543932e-02 1.1959494e-08 2.1091030e-11]]


Hard-maxed form of the prediction:

 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
```

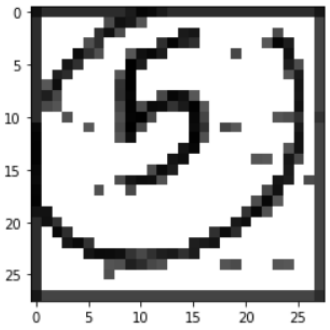
Final Result

```
=========PREDICTION============
```



```
Final Output: 6

Prediction (Softmax) from the neural network:

 [[5.3250435e-05 0.0000000e+00 8.9849094e-36 0.0000000e+00 0.0000000e+00
  2.5446657e-21 9.9994671e-01 0.0000000e+00 5.9479178e-34 0.0000000e+00]]


Hard-maxed form of the prediction:

 [[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

```
=========PREDICTION============
```



```
Final Output: 5

Prediction (Softmax) from the neural network:

 [[1.7565277e-07 0.0000000e+00 1.9394616e-22 1.2296227e-13 9.6880522e-36
  9.9790347e-01 9.8562975e-33 0.0000000e+00 2.0964162e-03 1.0901179e-19]]


Hard-maxed form of the prediction:

 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
```

```
In [39]: final_o

Out[39]: [6, 5]
```

VI. Conclusion

Our project is successfully able to segregate parts of the document image, find relevant areas and identify the marks given to the student. Three models were trained to determine the number from the image. A custom dataset was created of encircled digits to be used for training one of the models. Model was tested with images of different numbers of marks and at different colour intensities.The model is able to identify marks provided it satisfies the requirement that it be written in red colour and is encircled. The project has some future scope in expanding the features that can be used to segment the marks such as using significant font size difference or selecting based on surrounding empty spaces. One obstacle that has persisted is that half and cut off circles that remain in the image can negatively influence the results, if this issue could be resolved the confidence in the results would significantly increase. Another feature that can be worked on is creating an interface to upload an image and get the final result from it. As we have made use of OpenCV and Tensorflow, we can produce results real time thereby increasing functionalities.