Applications of Artificial Intelligence

EAI 6010, CRN 70749

Professor Vladimir Shapiro

Module 4: Assignment Week 4 - Vision AI Applications

Submitted By - Richa Umesh Rambhia

Vision AI Applications - Image Processing & MNIST Classification

---

Table of Contents

Introduction

Introduction to Image Processing in Python

An NCSU Libraries Workshop

Speaker: Nian Xiong

> This workshop provides an introduction to basic image processing techniques using the OpenCV computer vision library and some standard data analysis libraries in Python. Knowledge of image processing and Python programming is not required for this workshop, but will help.

The source of this notebook is located at [https://github.com/xn2333/OpenCV/blob/master/Seminar_Image_Processing_in_Python.ipynb (https://github.com/xn2333/OpenCV/blob/master/Seminar_Image_Processing_in_Python.ipynb)

Useful intro about [Colab (https://colab.research.google.com/notebooks/welcome.ipynb)

Useful intro about [OpenCV (https://opencv.org/)

Image Processing deals with the transformation of an image into a digital form and then performing operations and functions on the same in order to obtain some information from the image. [1] Image processing is used widely in many applications that requires certain fixed number of operations which need to be performed at each pixel of the image, and thus the processinf of an image is performed pixel by pixel. The five main types of image processing are *visualization, recognition, sharpening and restoration, pattern recognition, and retrieval.*

Thus, image processing or digital image processing basically deals with the *manipulation of the images* which is done in the form of signals that focuses on the images. The best example of image processing is **Adobe Photoshop** which is a widely used application in the field of image processing. [2] The various steps of image processing include *image acquisition, image preprocessing, image segmentation, feature extraction, statistical analysis, and classification based on a classifier.*

In this assignment, image processing is performed where the images are analyzed after reading and loading the image from the URL and various steps are implemented on the images in order to generate different effects on the original image. Different images are loaded and various operations such as grayscaling technqiues, contour detection, histogram equalization, etc. are performed on the images and are analyzed using the histogram charts and visuals.

Thus, image processing is executed and performed on the images using various techniques and methods which is majorly used these days in desiging and marketing fields and domains. Various other applications of image processing also include *healthcare domain, image reconstruction, face detection, etc.* The transformation and manipulation of the original images is differentiated using the histogram graph for each image which helps in determining the difference in the frequencies of the images.

MNIST Classification Using *Convolutional Neural Network*

- Author: Vladimir Shapiro, adopted from Diego Inácio's
- Notebook: MNIST_classification.ipynb (https://github.com/diegoinacio/computer-vision-notebooks/blob/master/Computer-Vision-Experiments/MNIST_classification.ipynb)

Digit classification using *Convolutional Neural Network*.

Handwritten Digit Recognition is the process which deals with the ability of machines to recognize human handwritten digits. [3] Handwritten digits classification or recognition is one of the use cases for AI in computer vision which uses D**eep Learning algorithms and Neural Networks** to perform the classification task.

The deep learning algorithm and technique that is used for handwritten digits classification is the *Convolutional Neural Network (CNN)*. CNN is used to classify the input automatically which helps in classifying the images and is an end-to-end prediction algorithm. [4] The process of CNN works in a way that it automatically extracts the important and relevant features from the input that is fed to the model. The below image gives a detailed explanation of the working/end-to-end process of a CNN algorithm.

For the handwritten digit classification task in this assignment using CNN algorithm, the dataset used is the **MNIST digits classification dataset** which is provided by the *Keras deep learning API* and has *60,000 training images and 10,000 testing images.* [4] The aim here is to build an accuracte, effective, and efficient CNN model to classify the handwritten digits.

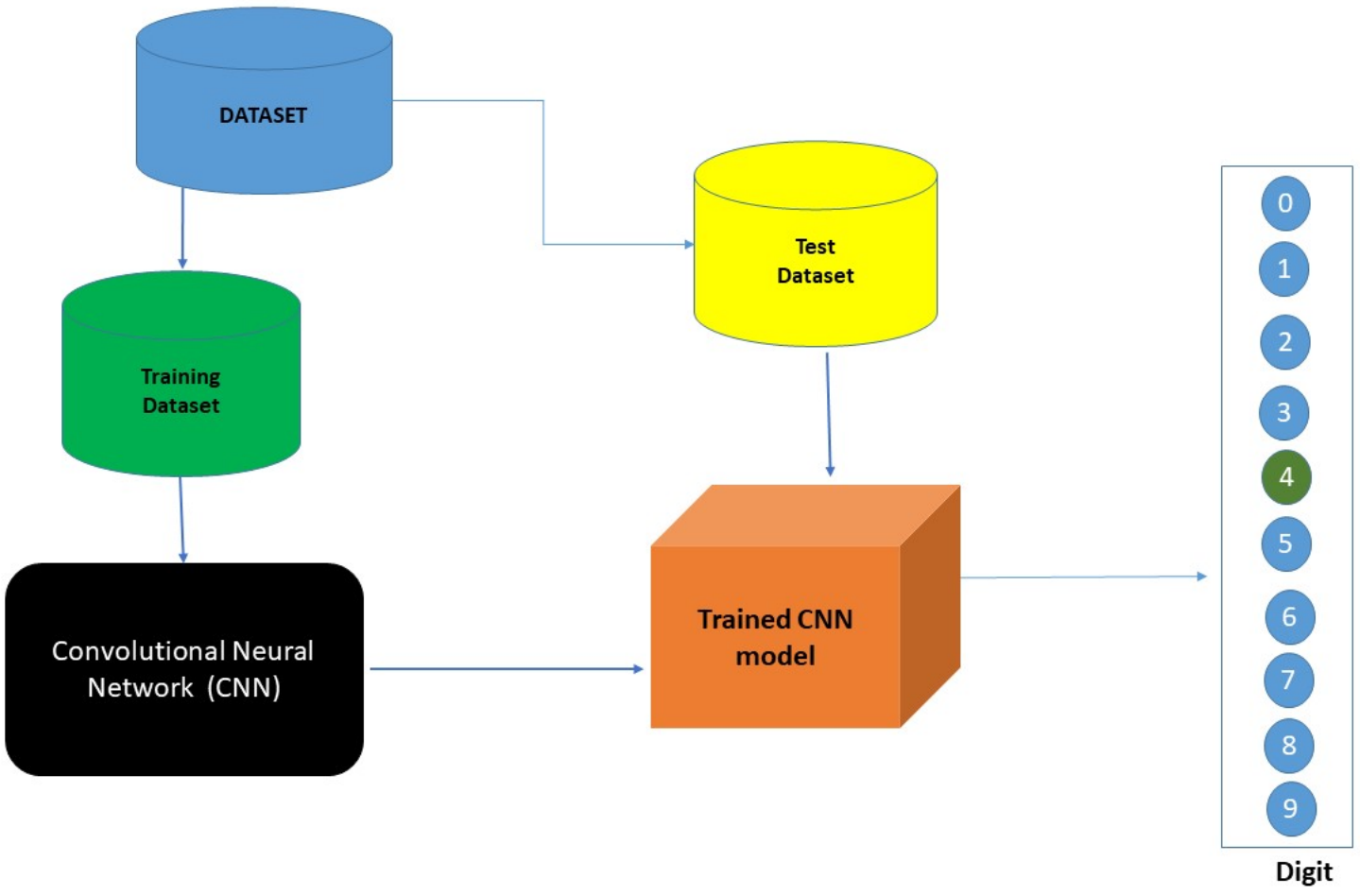

Figure 1. End-to-End Process of CNN [4]

Analysis

Part I: Image Processing

Q1: Load the supplied EAI6010_Image_Processing_Week4.ipynb script to Google Colab (through, e.g., your Google Drive or your GitHub).

A1. Loaded the supplied file to Google Colab.

Q2: Clear all outputs and execute (“Run All”) to make sure it runs correctly, producing all outputs.

A2. Executed the code below.

Step1: Load the Dependencies

> This section loads some required libraries used in this notebook: **numpy**, **pandas**, **cv2**, **skimage**, **PIL**, **matplotlib**

In [ ]:
```
import numpy as np
import pandas as pd
import cv2 as cv
from google.colab.patches import cv2_imshow # for image display
from skimage import io
from PIL import Image
import matplotlib.pylab as plt
```

Step2: Read Image from Urls

> In this step we will read images from urls, and display them using openCV, please note the difference when reading image in RGB and BGR format. The default input color channels are in BGR format for openCV.

RGB?

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))



# Create a list to store the urls of the images
urls = ["https://iiif.lib.ncsu.edu/iiif/0052574/full/800,/0/default.jpg",
        "https://iiif.lib.ncsu.edu/iiif/0016007/full/800,/0/default.jpg",
       "https://placekitten.com/800/571"]
# Read and display the image
# loop over the image URLs, you could store several image urls in the list

for url in urls:
  image = io.imread(url)
  image_2 = cv.cvtColor(image, cv.COLOR_BGR2RGB)
  final_frame = cv.hconcat((image, image_2))
  cv2_imshow(final_frame)
  print('\n')
```

```
<IPython.core.display.Javascript object>
```

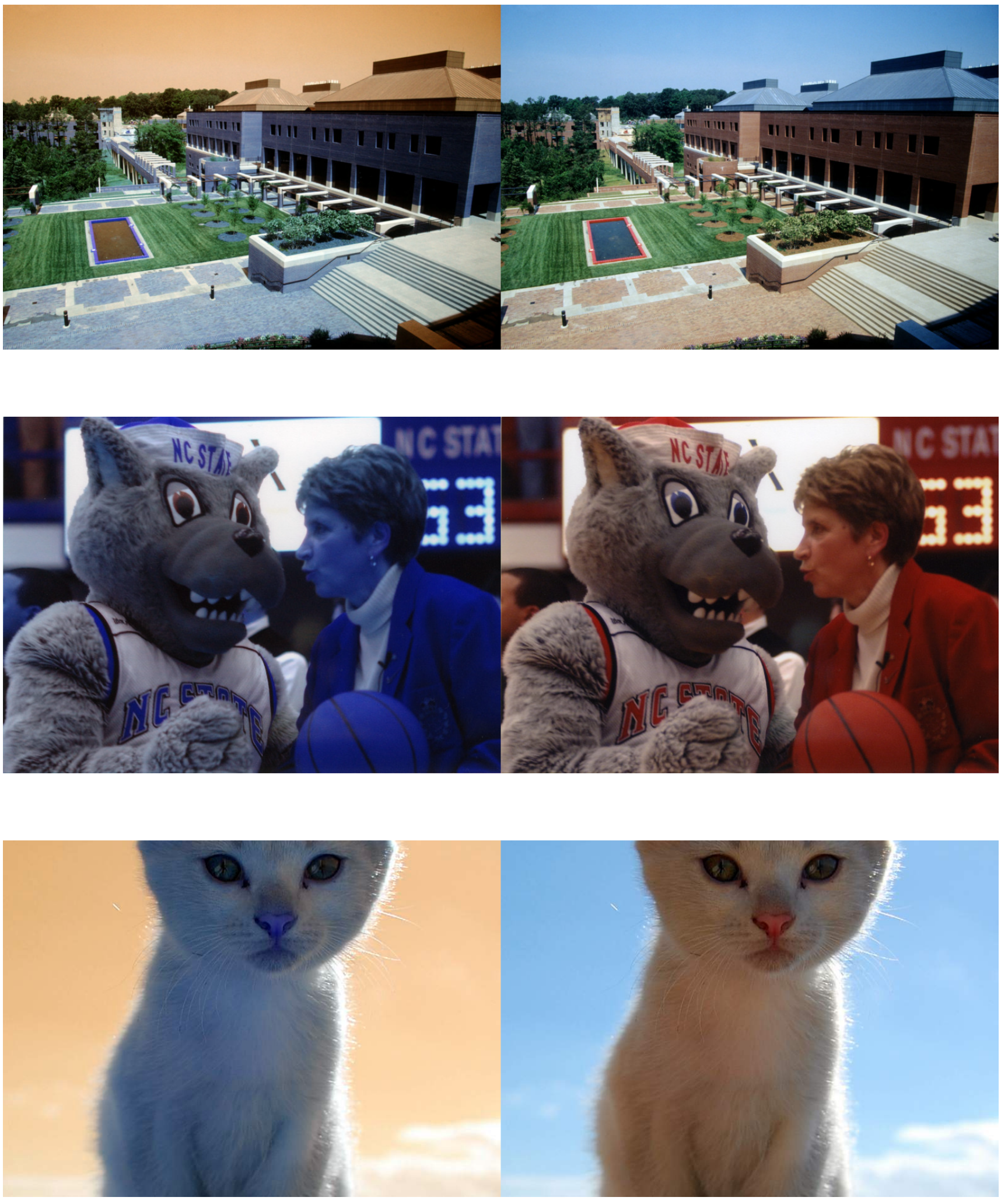

Figure 2. Images loaded through URL

TODO #1: Read an image from a URL and display it

Image source examples:

Place Kitten (https://placekitten.com/) - use the base Place Kitten URL followed by a width and height separated by backslashes "/". For example, use the URL `https://placekitten.com/500/300` to fetch a cat image with a width of 500px and height of 300px.

NC State University Libraries Special Collections (https://d.lib.ncsu.edu/collections/catalog) - browse the site to find an image thumbnail. Right-click on the thumbnail and select "Copy Image Address". The address will look like this: `https://iiif.lib.ncsu.edu/iiif/0051230/square/300,/0/default.jpg` . Replace the word "square" with the word "full" and replace "300" with "800" to access the full image at a width of 800px.

Google Image search (https://www.google.com/imghp?hl=en) - search for an image. Left-click one of the returned images, then right-click on the full image, and then select "Copy Image Address".

Q3: Uncomment the code in the cell beginning with ## TODO: LOAD IMAGE, and load your portrait or another desired picture. Run the rest of the script with it instead of the default image in the supplied script.

A3. Uncommented the code, loaded a desired picture and executed the rest of the script.

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))


## TODO: LOAD IMAGE
url = "https://www.anthropics.com/portraitpro/img/page-images/homepage/v21/new-features-B.jpg"
myImg = io.imread(url)
cv2_imshow(cv.cvtColor(myImg, cv.COLOR_BGR2RGB))
```

```
<IPython.core.display.Javascript object>
```

Figure 3. Reading image from an URL

Step3: Image Contours and Histograms

In [ ]:
```
# Check the image matrix data type (could know the bit depth of the image)
print("Image Matrix Data Type:", myImg.dtype)
# Check the height of image
print("Height of Image:", myImg.shape[0])
# Check the width of image
print("Width of Image:", myImg.shape[1])
# Check the number of channels of the image
print("Number of Channels of Image:", myImg.shape[2])
```

```
Image Matrix Data Type: uint8
Height of Image: 665
Width of Image: 1000
Number of Channels of Image: 3
```

Generate Histogram of color image and grayscale image

Sometimes you want to enhance the contrast in your image or expand the contrast in a particular region while sacrificing the detail in colors that don't vary much, or don't matter. A good tool to find interesting regions is the histogram. To create a histogram of our image data, we use the matplot.pylab `hist()` function.

More info: Histogram (https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_table_of_contents_histograms/py_table_of_contents_histogram

Display the histogram of all the pixels in the color image

In [ ]:
```
plt.hist(myImg.ravel(),bins = 256, range = [0,256])
plt.show()
```

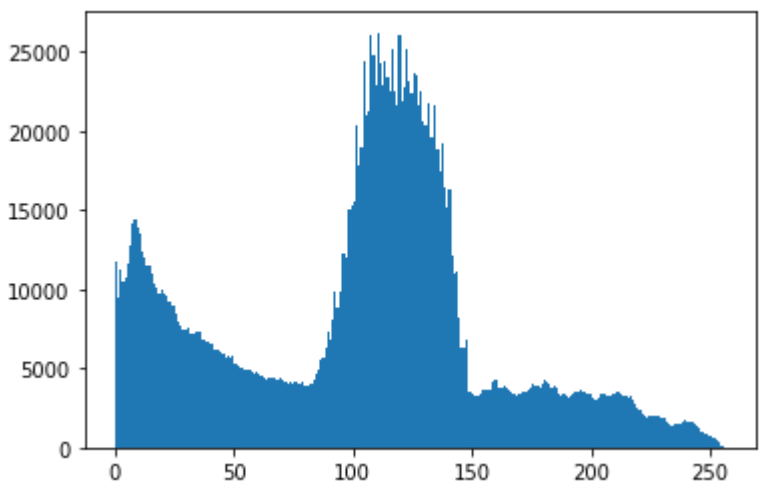

Figure 4. Histogram of all pixels

Display the histogram of R, G, B channel

In [ ]:
```
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv.calcHist([myImg],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

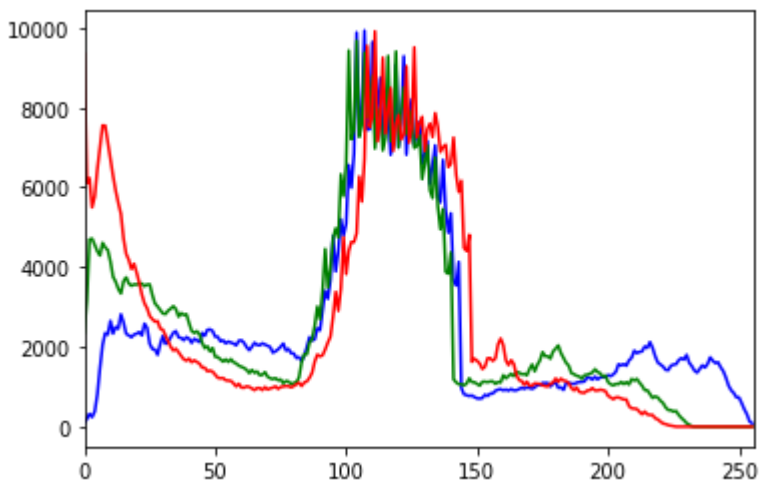

Figure 5. Histogram of R,G,B channel

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))


gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
cv2_imshow(gray_image)
```

```
<IPython.core.display.Javascript object>
```

Figure 6. Gray picture of original image

In [ ]:
```
# Plot the histogram of the gray image. We could observe that the frequency of
# the image hist has decreased ~ 1/3 of the histogram of color image
plt.hist(gray_image.ravel(),bins = 256, range = [0, 256])
plt.show()
```

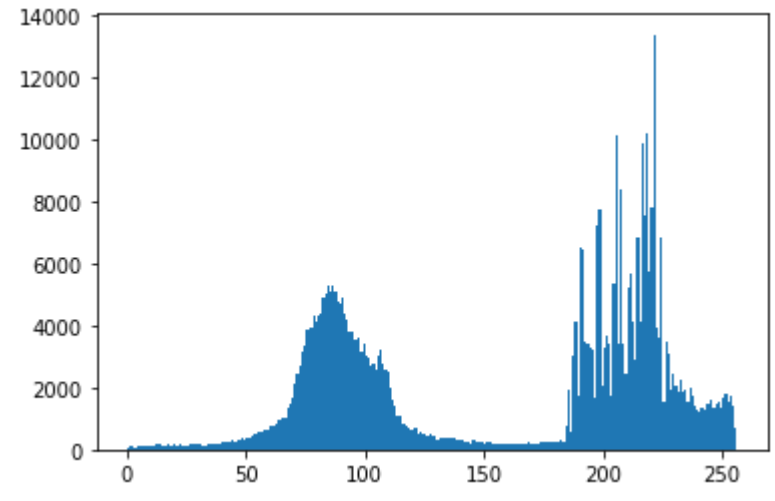

Figure 7. Histogram of gray image

TODO # 2: DISPLAY THE GRAYSCALE OF YOUR COLOR IMAGE AND GENERATE HISTOGRAM

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))


myGrayImg = cv.cvtColor(myImg, cv.COLOR_BGR2GRAY)
cv2_imshow(myGrayImg)
```

```
<IPython.core.display.Javascript object>
```

Figure 8. Grayscale of color image

In [ ]:
```
# Plot the histogram of the gray image. We could observe that the frequency of
# the image hist has decreased ~ 1/3 of the histogram of color image
plt.hist(myGrayImg.ravel(),bins = 256, range = [0, 256])
plt.show()
```

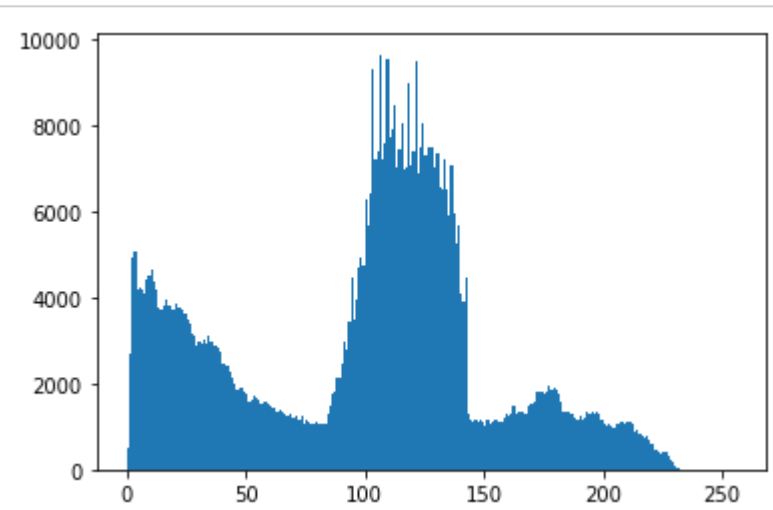

Figure 9. Historgram of gray image

Find image contour of the grayscale image

Method 1: Use the matplotlib. contour

More Info: [matplotlib contour (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.contour.html)]

Contour of original picture.

```
In [ ]: plt.contour(gray_image, origin = "image")
```

```
Out[12]: <matplotlib.contour.QuadContourSet at 0x7fb59a106390>
```

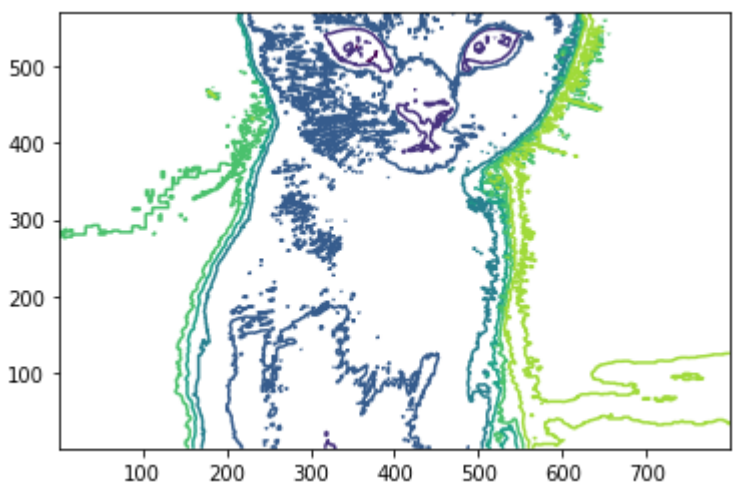

Figure 10. Contour of original image

Method 2: Use the openCV lib

More info: [Contour (https://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html)]

```
In [ ]: # Set threshold for the contour detection
ret, thresh = cv.threshold(gray_image,150,255,0)
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(image, contours, -1, (0, 255, 0), 3)
plt.imshow(image)
```

```
Out[13]: <matplotlib.image.AxesImage at 0x7fb599d86410>
```

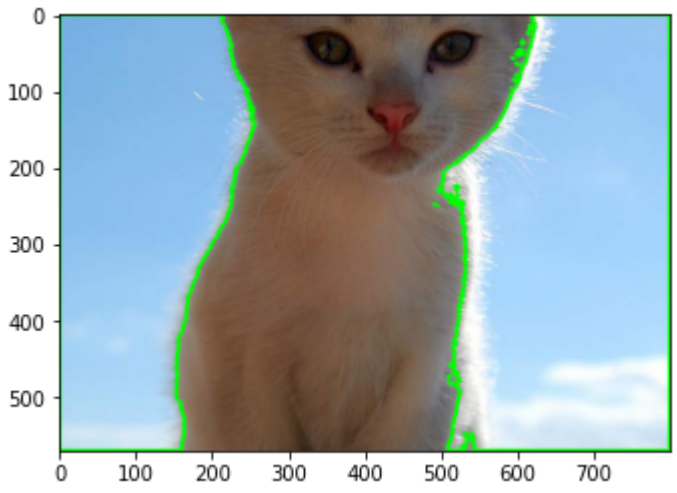

Figure 11. Contour detection of original image

TODO # 3: Find the contour of your own image

```
In [ ]: # Your code
plt.contour(myGrayImg, origin = "image")
```

```
Out[79]: <matplotlib.contour.QuadContourSet at 0x7fb599193710>
```

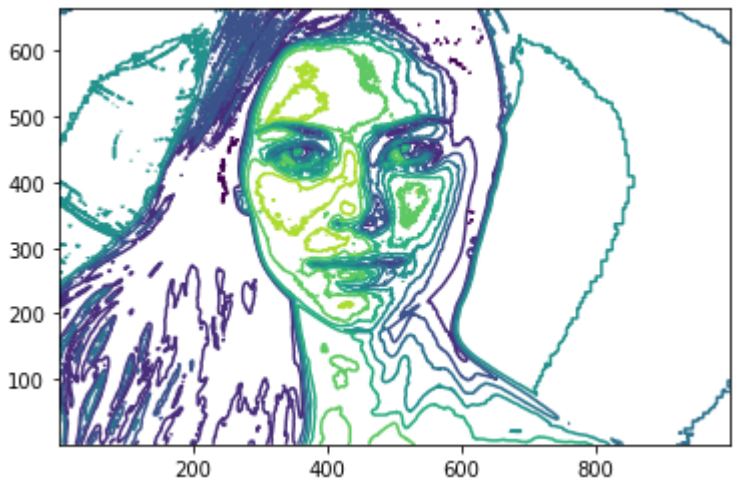

Figure 12. Contour of image

In [ ]:
```
# Set threshold for the contour detection
ret, thresh = cv.threshold(myGrayImg, 125,255,0)
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(myImg, contours, -1, (0, 255, 0), 3)
plt.imshow(myImg)
```

Out[80]: <matplotlib.image.AxesImage at 0x7fb59889e590>

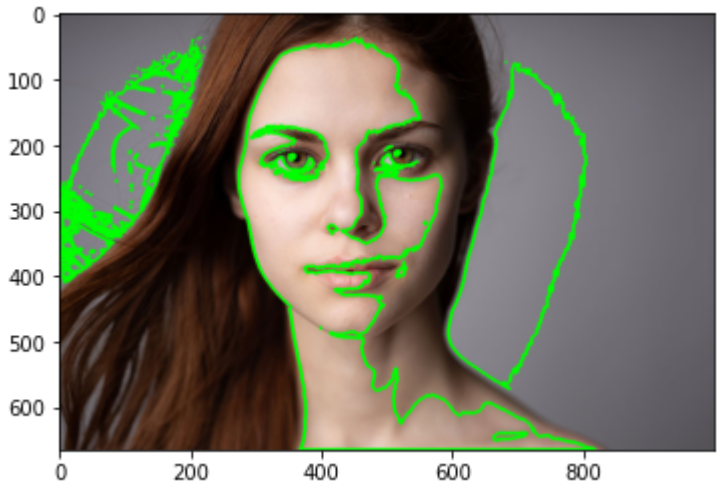

Figure 13. Contour detection of image

Step4: Grayscale Transform and Histogram Equalization

Grayscale Transformation

This section provides some examples of conducting mathematical transformations of the grayscale image

Grayscale Transformation of original picture.

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))


# This is an inverse operation of the grayscale image, you could see that the
# bright pixels become dark, and the dark pixels become bright
im2 = 255 - gray_image
cv2_imshow(im2)
```

<IPython.core.display.Javascript object>

Figure 14. Grayscale transformation of original image

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))


# Another transform of the image, after adding a constant,
# all the pixels become brighter and a hazing-like effect of the image is generated
im3 = (100.0/255)*gray_image + 100
cv2_imshow(im3)
```

```
<IPython.core.display.Javascript object>
```

Figure 15. Grayscale transformation of original image after adding constant

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))


# The lightness level of the gray_image decreases after this step
im4 = 255.0*(gray_image/255.0)**2
cv2_imshow(im4)
```

```
<IPython.core.display.Javascript object>
```

Figure 16. Grayscale transformation of original image where lightness level decreases

TODO #4: Try some mathematical operations on your image

In [ ]:
```
## Implement your code here


from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))



# This is an inverse operation of the grayscale image, you could see that the
# bright pixels become dark, and the dark pixels become bright
im20 = 255 - myGrayImg
cv2_imshow(im20)
```

```
<IPython.core.display.Javascript object>
```

Figure 17. Grayscale transformation of image

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))



# Another transform of the image, after adding a constant,
# all the pixels become brighter and a hazing-like effect of the image is generated
im30 = (100.0/255)*myGrayImg + 100
cv2_imshow(im30)
```

```
<IPython.core.display.Javascript object>
```

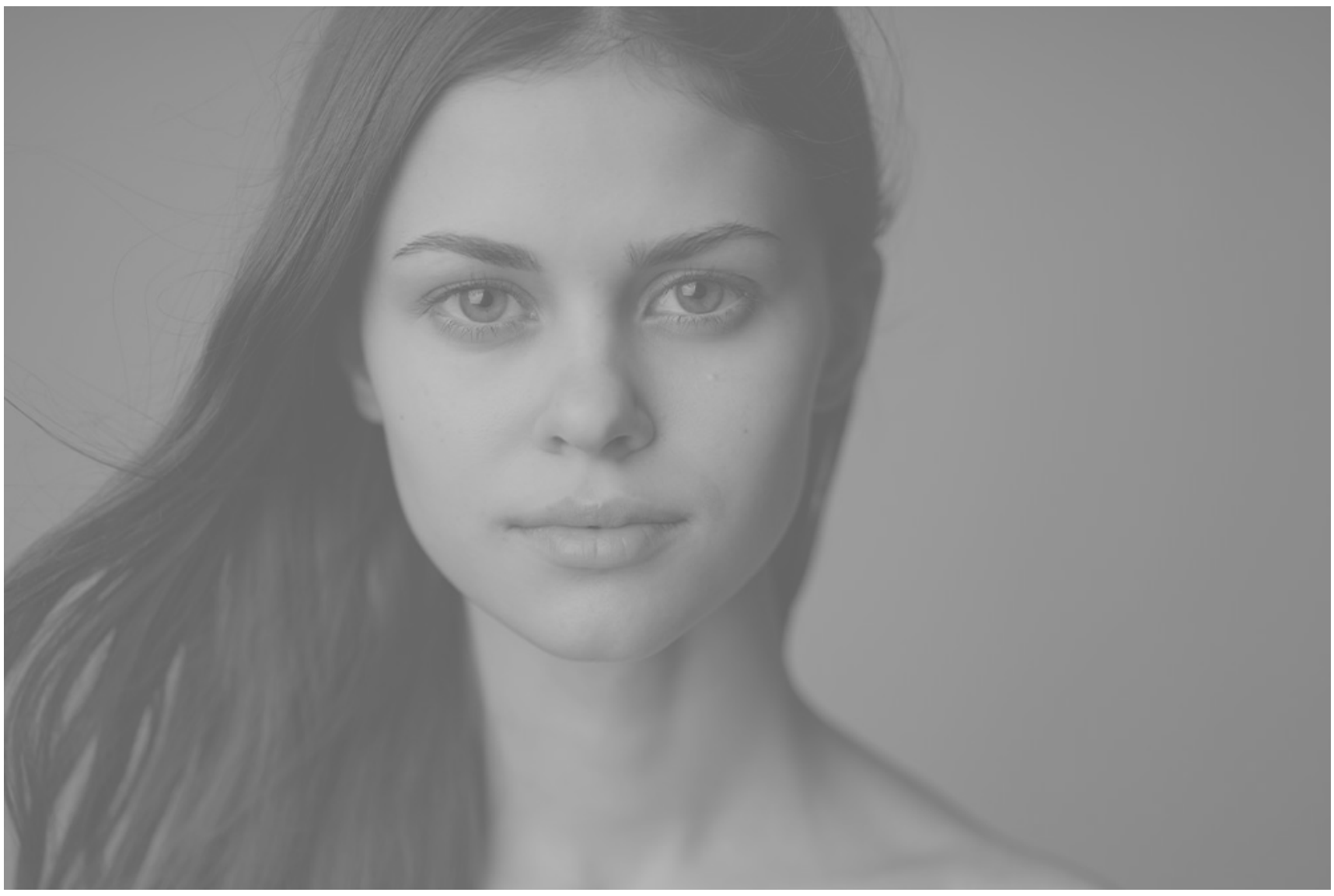

Figure 18. Grayscale transformation of image after adding a constant

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))


# The lightness level of the myGrayImg decreases after this step
im40 = 255.0*(myGrayImg/255.0)**2
cv2_imshow(im40)
```

```
<IPython.core.display.Javascript object>
```

Figure 19. Grayscale transformation of image where lightness level decreases

Histogram Equalization

This section demonstrates histogram equalization on a dark image. This transform flattens the gray-level histogram so that all intensities are as equally common as possible. The transform function is a cumulative distribution function (cdf) of the pixel values in the image (normalized to map the range of pixel values to the desired range). This example uses image 4 (im4).

In [ ]:
```
from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))



# function of histogram equalization
def histeq(im, nbr_bins = 256):
  """ Histogram equalization of a grayscale image.  """
  # get the image histogram
  imhist, bins = np.histogram(im.flatten(), nbr_bins, [0, 256])
  cdf = imhist.cumsum() # cumulative distribution function
  cdf = imhist.max()*cdf/cdf.max()  #normalize
  cdf_mask = np.ma.masked_equal(cdf, 0)
  cdf_mask = (cdf_mask - cdf_mask.min())*255/(cdf_mask.max()-cdf_mask.min())
  cdf = np.ma.filled(cdf_mask,0).astype('uint8')
  return cdf[im.astype('uint8')]

# apply the function on your dark image to increase the contrast
# we could observe that the contrast of the black background has increased
im5 = histeq(im4)
cv2_imshow(im5)
```

```
<IPython.core.display.Javascript object>
```

Figure 20. Histogram equalization of the original image

In [ ]:
```
# Extra: try to visualize the histogram of the image after histogram equalization
# Before histogram equalization
plt.hist(im4.ravel(),bins = 256, range = [0, 256])
plt.show()
```

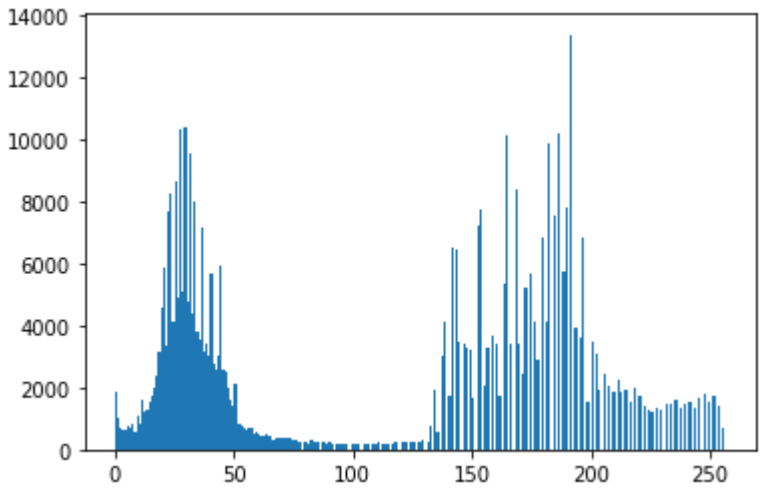

Figure 21. Histogram graph before histogram equalization of the original image

In [ ]:
```
# After histogram equalization
plt.hist(im5.ravel(),bins = 256, range = [0, 256])
plt.show()
```

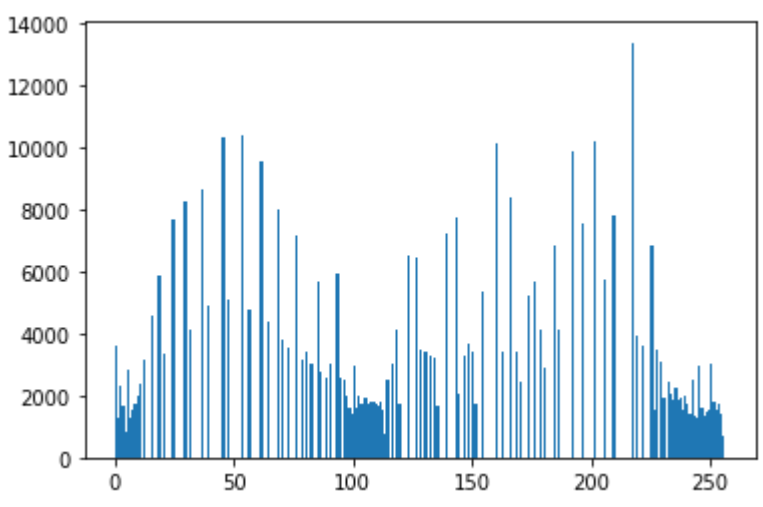

Figure 22. Histogram graph after histogram equalization of the original image

TODO # 5: Do a histogram equalization on your grayscale image

In [ ]:
```
## TODO: IMPLEMENT YOUR CODE HERE
## CODE INSTRUCTION

from IPython.display import HTML, Javascript
# Disable vertical scrolling by setting a massive max-height
display(Javascript('google.colab.output.setIframeHeight(-1, true, {maxHeight: 100000})'))



# function of histogram equalization
def histeq(im, nbr_bins = 256):
  """ Histogram equalization of a grayscale image.  """
  # get the image histogram
  imhist, bins = np.histogram(im.flatten(), nbr_bins, [0, 256])
  cdf = imhist.cumsum() # cumulative distribution function
  cdf = imhist.max()*cdf/cdf.max()  #normalize
  cdf_mask = np.ma.masked_equal(cdf, 0)
  cdf_mask = (cdf_mask - cdf_mask.min())*255/(cdf_mask.max()-cdf_mask.min())
  cdf = np.ma.filled(cdf_mask,0).astype('uint8')
  return cdf[im.astype('uint8')]

# apply the function on your dark image to increase the contrast
# we could observe that the contrast of the black background has increased
im50 = histeq(im40)
cv2_imshow(im50)
```

```
<IPython.core.display.Javascript object>
```

Figure 23. Histogram equalization of the image

Plotting the histogram of the image.

In [ ]:
```
# Extra: try to visualize the histogram of the image after histogram equalization
# Before histogram equalization
plt.hist(im40.ravel(),bins = 256, range = [0, 256])
plt.show()
```

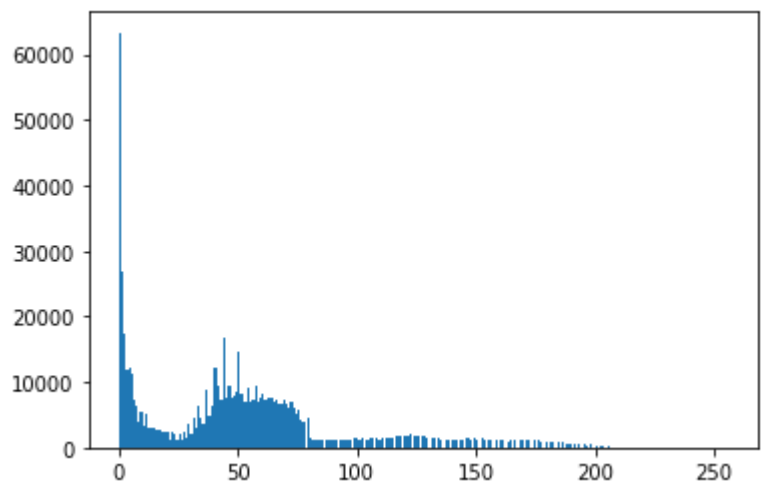

Figure 24. Histogram graph before histogram equalization of the image

```
In [ ]: # After histogram equalization
plt.hist(im50.ravel(),bins = 256, range = [0, 256])
plt.show()
```

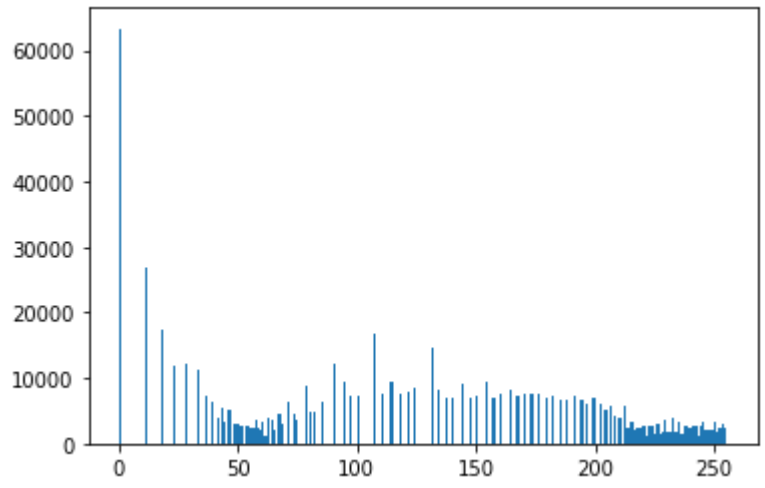

Figure 25. Histogram graph after histogram equalization of the image

Step5: Fourier Transform of Gray Images

FFT (https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_transforms/py_fourier_transform/py_fourier_transform.html)

> A fourier transform is used to find the frequency domian of an image. You can consider an image as a signal which is sampled in two directions. So taking a fourier transform in both X and Y directions gives you the frequency representation of image. For the sinusoidal signal, if the amplitude varies so fast in short time, you can say it is a high frequency signal. If it varies slowly, it is a low frequency signal. Edges and noises are high frequency contents in an image because they change drastically in images.

```
In [ ]: # Blur the grayscale image by a Guassian filter with kernel size of 10
imBlur = cv.blur(gray_image,(5,5))
# Transform the image to frequency domain
f = np.fft.fft2(imBlur)
# Bring the zero-frequency component to the center
fshift = np.fft.fftshift(f)
magnitude_spectrum = 30*np.log(np.abs(fshift))

plt.subplot(121),plt.imshow(imBlur, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

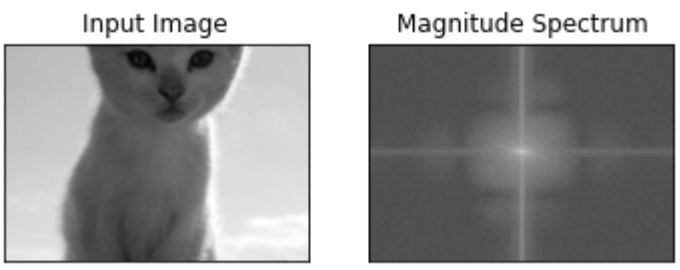

Figure 26. Fourier Transformation of gray images of the original image

TODO # 6: Generate a fourier transform of your grayscale image

In [ ]:
```
# todo: Implement your code here


# Blur the grayscale image by a Guassian filter with kernel size of 10
imBlur2 = cv.blur(myGrayImg,(5,5))
# Transform the image to frequency domain
f2 = np.fft.fft2(imBlur2)
# Bring the zero-frequency component to the center
fshift2 = np.fft.fftshift(f2)
magnitude_spectrum = 30*np.log(np.abs(fshift2))

plt.subplot(121),plt.imshow(imBlur2, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

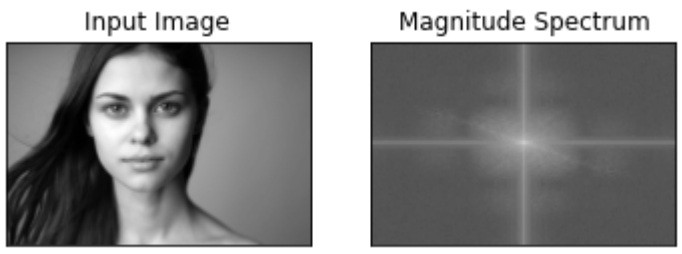

Figure 27. Fourier Transformation of gray images of the image

Step6: Finding Edges by Highpass Filtering in FFT

This section demonstrates conducting a high pass filter to remove the low frequency component, resulting in a sharpened image which contains the edges

In [ ]:
```
rows, cols = imBlur.shape
crow,ccol = round(rows/2) , round(cols/2)
# remove low frequencies with a rectangle size of 10
fshift[crow-10:crow+10, ccol-10:ccol+10] = 0
f_ishift = np.fft.ifftshift(fshift)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)

plt.figure(figsize=([20, 20]))
plt.subplot(131),plt.imshow(imBlur, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_back, cmap = 'gray')
plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img_back)
plt.title('Result in JET'), plt.xticks([]), plt.yticks([])
plt.show()
```

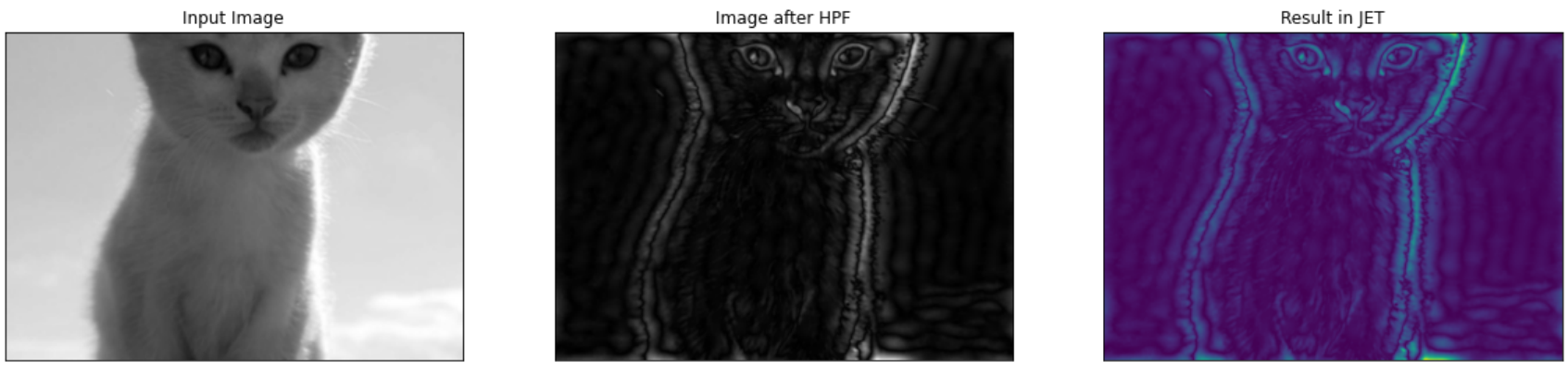

Figure 28. Finding edges by Highpass filtering in FFT of the original image

Finding Edges by Highpass Filtering in FFT of your own image

In [ ]:
```
rows, cols = imBlur2.shape
crow,ccol = round(rows/2) , round(cols/2)
# remove low frequencies with a rectangle size of 10
fshift2[crow-10:crow+10, ccol-10:ccol+10] = 0
f_ishift2 = np.fft.ifftshift(fshift2)
img_back2 = np.fft.ifft2(f_ishift2)
img_back2 = np.abs(img_back2)

plt.figure(figsize=([20, 20]))
plt.subplot(131),plt.imshow(imBlur2, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_back2, cmap = 'gray')
plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img_back2)
plt.title('Result in JET'), plt.xticks([]), plt.yticks([])
plt.show()
```

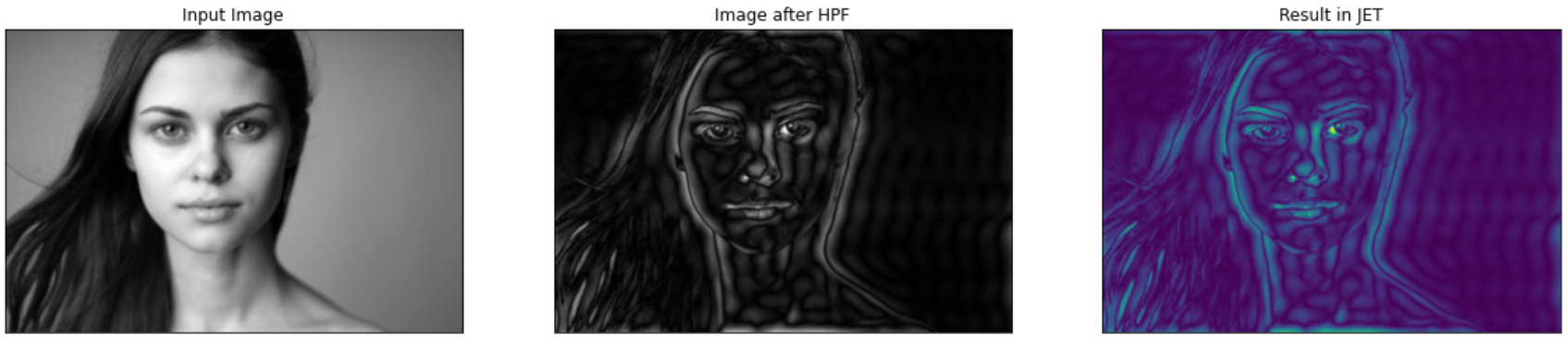

Figure 29. Finding edges by Highpass filtering in FFT of the image

Q4: At the end, comment on the results.

A4. Results:

Based on the techniques and methods for image processing, the new image loaded was processed for gray scale image, histogram equalization, contour image, etc. The grayscale image is first obtained of the original image which is then used for the further image processing techniques.

While comparing the histogram visual of both the colored image and the grayscaled image it is observed that the frequency in the histogram of the grayscale image is decreased as compared to the histogram of the colored image. The grayscale image obtained is then contoured for which a threshold is set in order to find the contour for the image. For the grayscale transformation and histogram equalization, we observe that the grayscale image has an inverse operation been executed on the image, which is completely like an inverted image where the dark pixels have becomes bright and vice-versa. But after adding a constant value to the inversed image, the pixels become bright and a new transformed image is obtained, and finally the brightness level which was increased is dropped to a minimum level resulting in a grayscaled transformed image.

Histogram equalization on the other hand describes the equalization on a dark image where the contrast of the dark image is increased and the pixels with a black background have a increase in the contrast. When the histograms of the equalization are compared it is observed that the frequency of the higher distribution is decreased as compared to the original picture and vice versa. This is due to the contrast effect applied on the darker image generated. Lastly, fourier transformation of gray images is performed which helps to find the frequency domain of the image. Here, the low frequency component is removed which has resulted in a sharpened image.

Part II: MNIST Classification

Q1: Load the supplied EAI6010_DL_MNIST_classification_Week4.ipynb script to Google Colab (from, e.g., your Google Drive or your GitHub).

A1. Loaded the supplied file to Google Colab.

Q2: Training on a conventional CPU may be slower than on GPU/TPU. To request that in Colab go to Runtime −> Change Runtime Type and select GPU from the dropdown list before running the script:

A2. Changed Runtime Type as per instructed.

CNN Model for Handwritten Digit Classification

In [ ]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

import tensorflow as tf
```

Read the dataset

In [ ]:
```
mnist = tf.keras.datasets.mnist
[X_train, Y_train],[X_test, Y_test] = mnist.load_data()
X_train = X_train[..., None]
X_test = X_test[..., None]

# one hot for multi-class classification
# MNIST = 10 classes [0-9]
Y_train_oh = np.zeros((Y_train.size, 10))
Y_train_oh[np.arange(Y_train.size), Y_train] = 1
Y_test_oh = np.zeros((Y_test.size, 10))
Y_test_oh[np.arange(Y_test.size), Y_test] = 1

print('X_train:', X_train.shape)
print('Y_train:', Y_train.shape)
print('Y_train_oh:', Y_train_oh.shape)
print('X_test:', X_test.shape)
print('Y_test:', Y_test.shape)
print('Y_test_oh:', Y_test_oh.shape)
```

```
X_train: (60000, 28, 28, 1)
Y_train: (60000,)
Y_train_oh: (60000, 10)
X_test: (10000, 28, 28, 1)
Y_test: (10000,)
Y_test_oh: (10000, 10)
```

Display several randomly selected handwritten digits to provide a raw impression how the digits look like.

In [ ]:
```
plt.rcParams['figure.figsize'] = (20, 10)
fig, AX = plt.subplots(3, 6, sharex=True, sharey=True)

np.random.seed(1234)
for ax in AX.ravel():
    rindex = np.random.randint(Y_train.size)
    img_show = X_train[rindex][...,0]
    ax.imshow(img_show, cmap='gray')
    # title label + one-hot
    title = f'{Y_train[rindex]} :: '
    title += ''.join([str(int(e)) for e in Y_train_oh[rindex]])
    ax.set_title(title)
plt.grid(False)
```

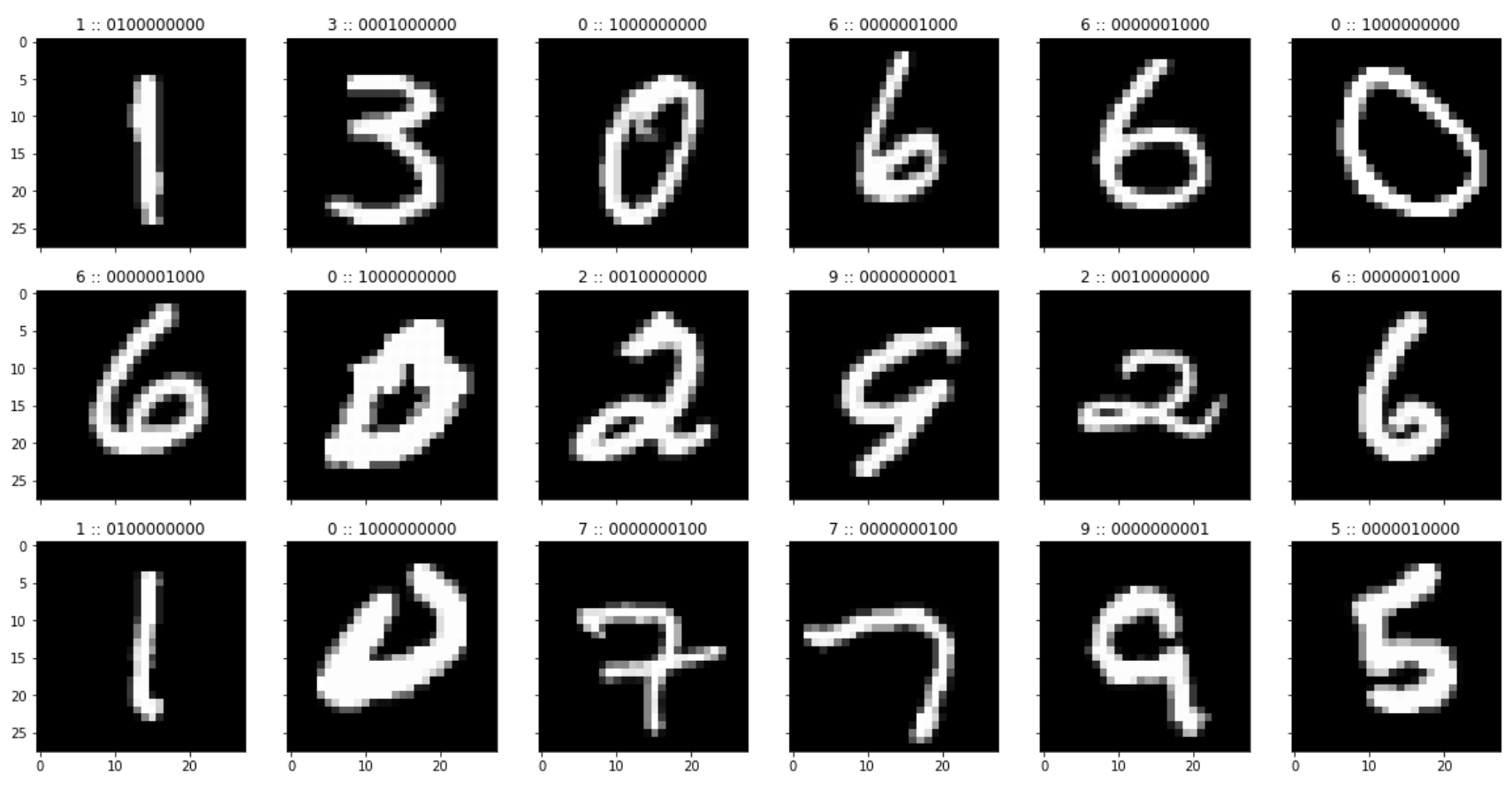

Figure 30. Randomly selected handwritten digits

Transform dataset

To a format suitable for the training and predictions later on. A separate *X_test* dataset is allocated.

In [ ]:
```
# Change dtype to float32
X_train = X_train.astype(np.float32)
Y_train = Y_train.astype(np.float32)
X_test = X_test.astype(np.float32)
Y_test = Y_test.astype(np.float32)

# Change axes and normalization
X_train, Y_train = X_train/255, Y_train.reshape(-1, 1)
X_test, Y_test = X_test/255, Y_test.reshape(-1, 1)

# reshape Xs
x_train = X_train.reshape(-1, 28*28)
x_test = X_test.reshape(-1, 28*28)

print('X_train:', X_train.shape)
print('x_train:', x_train.shape)
print('Y_train:', Y_train.shape)
print('Y_train_oh:', Y_train_oh.shape)
print('X_test:', X_test.shape)
print('x_test:', x_test.shape)
print('Y_test:', Y_test.shape)
print('Y_test_oh:', Y_test_oh.shape)
```

```
X_train: (60000, 28, 28, 1)
x_train: (60000, 784)
Y_train: (60000, 1)
Y_train_oh: (60000, 10)
X_test: (10000, 28, 28, 1)
x_test: (10000, 784)
Y_test: (10000, 1)
Y_test_oh: (10000, 10)
```

Classification using *Convolutional Neural Network (CNN)*

Training

Create Keras' Sequential CNN model, which is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

First, define the *structure* or *architecture* of the CNN model:

In [ ]:
```
# Get dimensions
N1, N2, C = X_train[0].shape

# Model
modelCNN = tf.keras.Sequential([
    tf.keras.layers.Input(X_train[0].shape),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(N1*N2, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

modelCNN.summary() # Display the structure
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_7 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_6 (Dropout)         (None, 12, 12, 64)        0

 flatten_3 (Flatten)         (None, 9216)              0

 dense_6 (Dense)             (None, 784)               7226128

 dropout_7 (Dropout)         (None, 784)               0

 dense_7 (Dense)             (None, 10)                7850

=================================================================
Total params: 7,252,794
Trainable params: 7,252,794
Non-trainable params: 0
_________________________________________________________________
```

Table 1. CNN model summary

Now, build the actual model, that is allocate computer memory, connections, hyperparameters, etc. per definitions above.

In [ ]:
```
modelCNN.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Begin the CNN training for a given number of epochs.

```
Tip: Training on a conventional CPU is significantly slower than on GPU/TPU. To request that go to "Run
time" -> "Change Runtime Type" and select "GPU" from the dropdown list prior to running the script.
```

Number of Epochs = 10

In [ ]:
```
NUMBER_OF_EPOCHS = 10
modelCNN.fit(X_train, Y_train_oh, epochs=NUMBER_OF_EPOCHS)
```

```
Epoch 1/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1213 - accuracy: 0.9625
Epoch 2/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0516 - accuracy: 0.9842
Epoch 3/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0368 - accuracy: 0.9884
Epoch 4/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0292 - accuracy: 0.9912
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0226 - accuracy: 0.9930
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0191 - accuracy: 0.9940
Epoch 7/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0155 - accuracy: 0.9949
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0159 - accuracy: 0.9948
Epoch 9/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0120 - accuracy: 0.9960
Epoch 10/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0122 - accuracy: 0.9959
```

Out[25]:
```
<keras.callbacks.History at 0x7fc3d34fe910>
```

Table 2. CNN training model with number of epochs = 10

Testing

Now, we have the trained model and can do predictions.

In [ ]:
```
fig, AX = plt.subplots(5, 10, sharex=True)
AX = [b for a in AX for b in a]

prediction_on_test = modelCNN.predict(X_test)

np.random.seed(1234)
for ax in AX:
    index = np.random.randint(Y_test.size)
    # Predict
    A_ = prediction_on_test[index]
    Y_ = np.argmax(A_)
    # Prepare image to show
    img_show = np.ones((N1, N2, 3))
    img_show *= X_test[index]
    # Green square: classified correctly
    # Red square: classified wrongly
    if Y_ == Y_test[index]:
        img_show[-4:, -4:] = (0, 1, 0)
    else:
        img_show[-4:, -4:] = (1, 0, 0)
    ax.imshow(img_show)
    ax.set_title(r'$\hat{Y_i}$ = ' + str(Y_) + r' ; $A_i$ = {:.02f}'.format(float(A_[Y_])), fontsize=10)
```

```
313/313 [==============================] - 1s 2ms/step
```

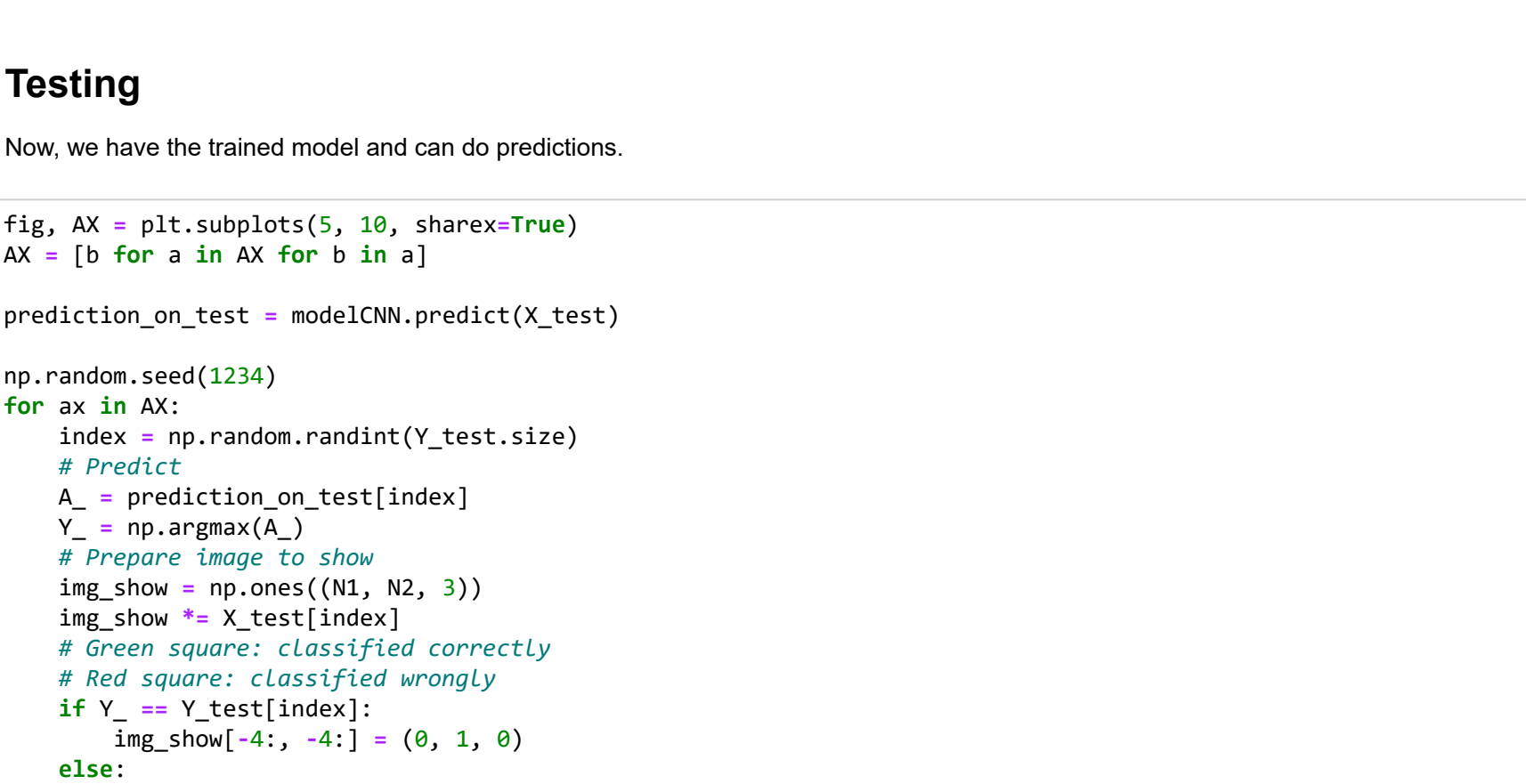

Figure 31. Prediction & Classification of handwritten digits where number of epochs is 10

Calculate a validation metric as:

In [ ]:
```
print('Validation')
#prediction_on_test = modelCNN.predict(X_test)
print('Test:', np.sum((Y_test_oh - prediction_on_test)**2, axis=1).mean())

prediction_on_train = modelCNN.predict(X_train)
print('Train:', np.sum((Y_train_oh - prediction_on_train)**2, axis=1).mean())
```

```
Validation
Test: 0.013838054481971167
1875/1875 [==============================] - 3s 2ms/step
Train: 0.0005020378157852767
```

Result with Number of Epochs = 10

1. Only 1 digit is wrongly classified as per the testing and prediction visualization figure
2. The validation metric for test dataset is 0.014
3. The loss at the last epoch is 0.01 and the accuracy is 0.996

Q3: After making predictions and calculating validation metrics change the number of training epochs, get new validation results, and visualize and reflect on them with the original number of epochs. Report whether the results improved or not in your opinion.

A3. Number of training epochs is changed in the below code, and new validation and visualization results are obtained which are compared with the original results to check for improvement in the model.

Model Building

In [ ]:
```
# Get dimensions
N1, N2, C = X_train[0].shape

# Model
modelCNN2 = tf.keras.Sequential([
    tf.keras.layers.Input(X_train[0].shape),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(N1*N2, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])


modelCNN2.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Number of Epochs = 20

In [ ]:
```
NUMBER_OF_EPOCHS = 20
modelCNN2.fit(X_train, Y_train_oh, epochs=NUMBER_OF_EPOCHS)
```

```
Epoch 1/20
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1237 - accuracy: 0.9615
Epoch 2/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0518 - accuracy: 0.9839
Epoch 3/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0380 - accuracy: 0.9881
Epoch 4/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0291 - accuracy: 0.9909
Epoch 5/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0236 - accuracy: 0.9924
Epoch 6/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0206 - accuracy: 0.9934
Epoch 7/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0186 - accuracy: 0.9943
Epoch 8/20
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0141 - accuracy: 0.9953
Epoch 9/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0132 - accuracy: 0.9961
Epoch 10/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0119 - accuracy: 0.9960
Epoch 11/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0104 - accuracy: 0.9965
Epoch 12/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0108 - accuracy: 0.9964
Epoch 13/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0107 - accuracy: 0.9967
Epoch 14/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0082 - accuracy: 0.9973
Epoch 15/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0097 - accuracy: 0.9971
Epoch 16/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0087 - accuracy: 0.9974
Epoch 17/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0070 - accuracy: 0.9980
Epoch 18/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0092 - accuracy: 0.9974
Epoch 19/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0078 - accuracy: 0.9977
Epoch 20/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0081 - accuracy: 0.9976
```

Out[29]:
```
<keras.callbacks.History at 0x7fc31b22d310>
```

Table 3. CNN training model with number of epochs = 20

Testing and classification

In [ ]:
```
fig, AX = plt.subplots(5, 10, sharex=True)
AX = [b for a in AX for b in a]

prediction_on_test = modelCNN2.predict(X_test)

np.random.seed(1234)
for ax in AX:
    index = np.random.randint(Y_test.size)
    # Predict
    A_ = prediction_on_test[index]
    Y_ = np.argmax(A_)
    # Prepare image to show
    img_show = np.ones((N1, N2, 3))
    img_show *= X_test[index]
    # Green square: classified correctly
    # Red square: classified wrongly
    if Y_ == Y_test[index]:
        img_show[-4:, -4:] = (0, 1, 0)
    else:
        img_show[-4:, -4:] = (1, 0, 0)
    ax.imshow(img_show)
    ax.set_title(r'$\hat{Y_i}$ = ' + str(Y_) + r' ; $A_i$ = {:.02f}'.format(float(A_[Y_])), fontsize=10)
```

```
313/313 [==============================] - 1s 2ms/step
```

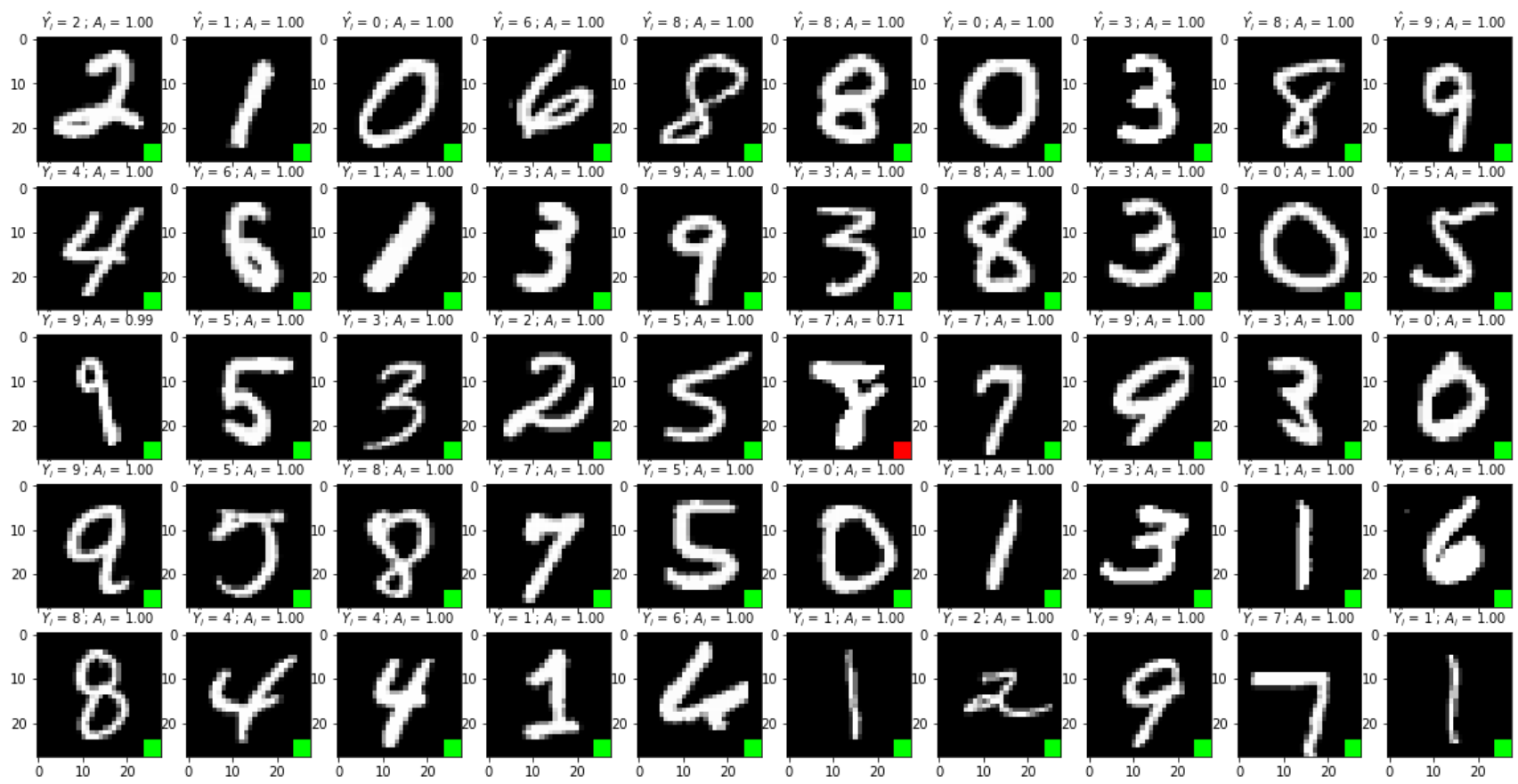

Figure 32. Prediction & Classification of handwritten digits where number of epochs is 20

Validation metric

In [ ]:
```
print('Validation')
#prediction_on_test = modelCNN.predict(X_test)
print('Test:', np.sum((Y_test_oh - prediction_on_test)**2, axis=1).mean())

prediction_on_train = modelCNN2.predict(X_train)
print('Train:', np.sum((Y_train_oh - prediction_on_train)**2, axis=1).mean())
```

```
Validation
Test: 0.011353574646967866
1875/1875 [==============================] - 3s 2ms/step
Train: 0.00019035698497492002
```

Result with Number of Epochs = 20

1. One digit is wrongly classified as per the testing and prediction visualization figure
2. The validation metric for test dataset is 0.011
3. The loss at the last epoch is 0.01 and the accuracy is 0.998

Model Building

In [ ]:
```
# Get dimensions
N1, N2, C = X_train[0].shape

# Model
modelCNN3 = tf.keras.Sequential([
    tf.keras.layers.Input(X_train[0].shape),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(N1*N2, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])


modelCNN3.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Number of Epochs = 5

In [ ]:
```
NUMBER_OF_EPOCHS = 5
modelCNN3.fit(X_train, Y_train_oh, epochs=NUMBER_OF_EPOCHS)
```

```
Epoch 1/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1278 - accuracy: 0.9605
Epoch 2/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0524 - accuracy: 0.9841
Epoch 3/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0371 - accuracy: 0.9886
Epoch 4/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0288 - accuracy: 0.9909
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0238 - accuracy: 0.9926
```

Out[33]:
```
<keras.callbacks.History at 0x7fc3193be6d0>
```

Table 4. CNN training model with number of epochs = 5

In [ ]:
```
fig, AX = plt.subplots(5, 10, sharex=True)
AX = [b for a in AX for b in a]

prediction_on_test = modelCNN3.predict(X_test)

np.random.seed(1234)
for ax in AX:
    index = np.random.randint(Y_test.size)
    # Predict
    A_ = prediction_on_test[index]
    Y_ = np.argmax(A_)
    # Prepare image to show
    img_show = np.ones((N1, N2, 3))
    img_show *= X_test[index]
    # Green square: classified correctly
    # Red square: classified wrongly
    if Y_ == Y_test[index]:
        img_show[-4:, -4:] = (0, 1, 0)
    else:
        img_show[-4:, -4:] = (1, 0, 0)
    ax.imshow(img_show)
    ax.set_title(r'$\hat{Y_i}$ = ' + str(Y_) + r' ; $A_i$ = {:.02f}'.format(float(A_[Y_])), fontsize=10)
```

```
313/313 [==============================] - 1s 2ms/step
```

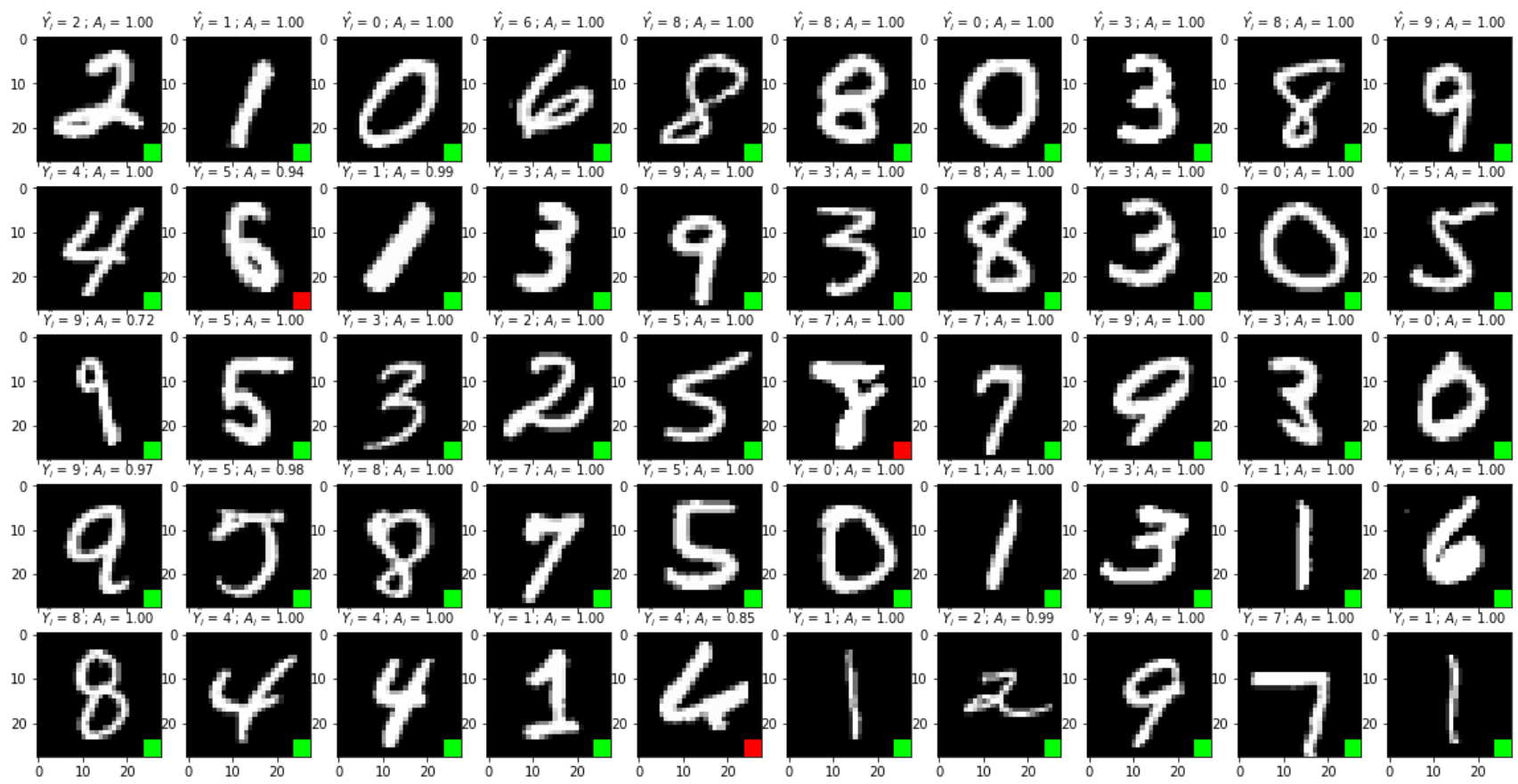

Figure 33. Prediction & Classification of handwritten digits where number of epochs is 5

In [ ]:
```
print('Validation')
#prediction_on_test = modelCNN.predict(X_test)
print('Test:', np.sum((Y_test_oh - prediction_on_test)**2, axis=1).mean())

prediction_on_train = modelCNN3.predict(X_train)
print('Train:', np.sum((Y_train_oh - prediction_on_train)**2, axis=1).mean())
```

```
Validation
Test: 0.014502411429347325
1875/1875 [==============================] - 4s 2ms/step
Train: 0.0037749076765147952
```

Result with Number of Epochs = 5

1. Three digits are wrongly classified as per the testing and prediction visualization figure
2. The validation metric for test dataset is 0.015
3. The loss at the last epoch is 0.02 and the accuracy is 0.993

Q3. Report whether the results improved or not in your opinion.

Based on the number of epochs changed from 10 to number of epochs 20, and then changed to 5 epochs, it is observed that the results obtained after changing the number of epochs has a minute difference in the test results.

When the model with the original number of epochs was executed, i.e., number of epochs = 10, there was **only one digit which was wrongly classified** with a test validation metric of **0.014**, whereas for number of epochs = 20, despite only **one digit being wrongly classified**, the test validation metric was **0.011**. And for number of epochs = 5, there were **three digits which were wrongly classified** and the test validation metric was **0.015**.

Thus, in my opinion the results have sightly improved by increasing the number of epochs based on the validation metric and the visualization/prediction results, because for 5 epochs i.e., decreasing the number of epochs, there are 3 digits which are wrongly classified and also the loss at the last epoch is more and accuracy is less compared to others, whereas for **20 epochs** the validation metric is less and the **accuracy at the last epoch is slightly more** along with **less loss at last epoch** as compared to the original model.

Hence, the results have slightly improved by changing the number of epochs.

Exploring parameters of the model

Q4: Explore alternatives to the default model parameters, such as loss, optimizer, and metrics. Experiment with the alternatives, visualize and report results, and interpret them, including the motivation behind parameter selection. Report whether the results improved or not in your opinion.

A4. Alternatives to the default parameters are explored in the below mentioned code, and new validation and visualization results are obtained which are compared with the original results to check for improvement in the model.

Model Building

In [ ]:
```
# Get dimensions
N1, N2, C = X_train[0].shape

# Model
modelCNN4 = tf.keras.Sequential([
    tf.keras.layers.Input(X_train[0].shape),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(N1*N2, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

modelCNN4.summary() # Display the structure
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 26, 26, 32)        320

 conv2d_15 (Conv2D)          (None, 24, 24, 64)        18496

 max_pooling2d_7 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_14 (Dropout)        (None, 12, 12, 64)        0

 flatten_7 (Flatten)         (None, 9216)              0

 dense_14 (Dense)            (None, 784)               7226128

 dropout_15 (Dropout)        (None, 784)               0

 dense_15 (Dense)            (None, 10)                7850

=================================================================
Total params: 7,252,794
Trainable params: 7,252,794
Non-trainable params: 0
_________________________________________________________________
```

Table 5. CNN model summary

In [ ]:
```
modelCNN4.compile(
    #loss='categorical_crossentropy',       # it is used for classification where target classes are more than t
    loss='mean_absolute_error',             # it computes the mean of absolute difference between labels and pre
    #optimizer='Adam',                      # implements the Adam algorithm
    optimizer='Adagrad',                    # implements the Adagrad algorithm
    metrics=['accuracy']
)
```

Motivation behind parameter selection

Considering the parameters of the CNN model: **loss, optimizer, and metrics.**

There are various types of loss values depending on the prediction type, for example, the default value i.e., categorical_crossentropy is used for classification where target classes are more than two and computes the crossentropy loss between the labels and predictions, whereas the mean_absolute_error computes the mean of absolute difference between labels and predictions. [5]

Similarly, there are various values for optimizers available in the Keras library. Adagrad, RMSprop, and Adam are the king of all the optimizers as it is very fast, robust, and flexible. [6]

Thus, the reason behind choosing these parameter values is that, mean absolute error will help in understanding the error difference between the labels and the prediction value whereas Adagrad is one of the top optimizers among all the others and is fast & flexible.

In [ ]:
```
NUMBER_OF_EPOCHS = 10
modelCNN4.fit(X_train, Y_train_oh, epochs=NUMBER_OF_EPOCHS)
```

```
Epoch 1/10
1875/1875 [==============================] - 55s 5ms/step - loss: 0.0780 - accuracy: 0.7504
Epoch 2/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0542 - accuracy: 0.7970
Epoch 3/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0449 - accuracy: 0.8218
Epoch 4/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0400 - accuracy: 0.8361
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0366 - accuracy: 0.8474
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0341 - accuracy: 0.8558
Epoch 7/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0324 - accuracy: 0.8610
Epoch 8/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0309 - accuracy: 0.8659
Epoch 9/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0296 - accuracy: 0.8718
Epoch 10/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0284 - accuracy: 0.8758
```

Out[51]:
```
<keras.callbacks.History at 0x7fc31b4bbad0>
```

Table 6. CNN training model with number of epochs = 10

In [ ]:
```
fig, AX = plt.subplots(5, 10, sharex=True)
AX = [b for a in AX for b in a]

prediction_on_test = modelCNN4.predict(X_test)

np.random.seed(1234)
for ax in AX:
    index = np.random.randint(Y_test.size)
    # Predict
    A_ = prediction_on_test[index]
    Y_ = np.argmax(A_)
    # Prepare image to show
    img_show = np.ones((N1, N2, 3))
    img_show *= X_test[index]
    # Green square: classified correctly
    # Red square: classified wrongly
    if Y_ == Y_test[index]:
        img_show[-4:, -4:] = (0, 1, 0)
    else:
        img_show[-4:, -4:] = (1, 0, 0)
    ax.imshow(img_show)
    ax.set_title(r'$\hat{Y_i}$ = ' + str(Y_) + r' ; $A_i$ = {:.02f}'.format(float(A_[Y_])), fontsize=10)
```

```
313/313 [==============================] - 1s 2ms/step
```

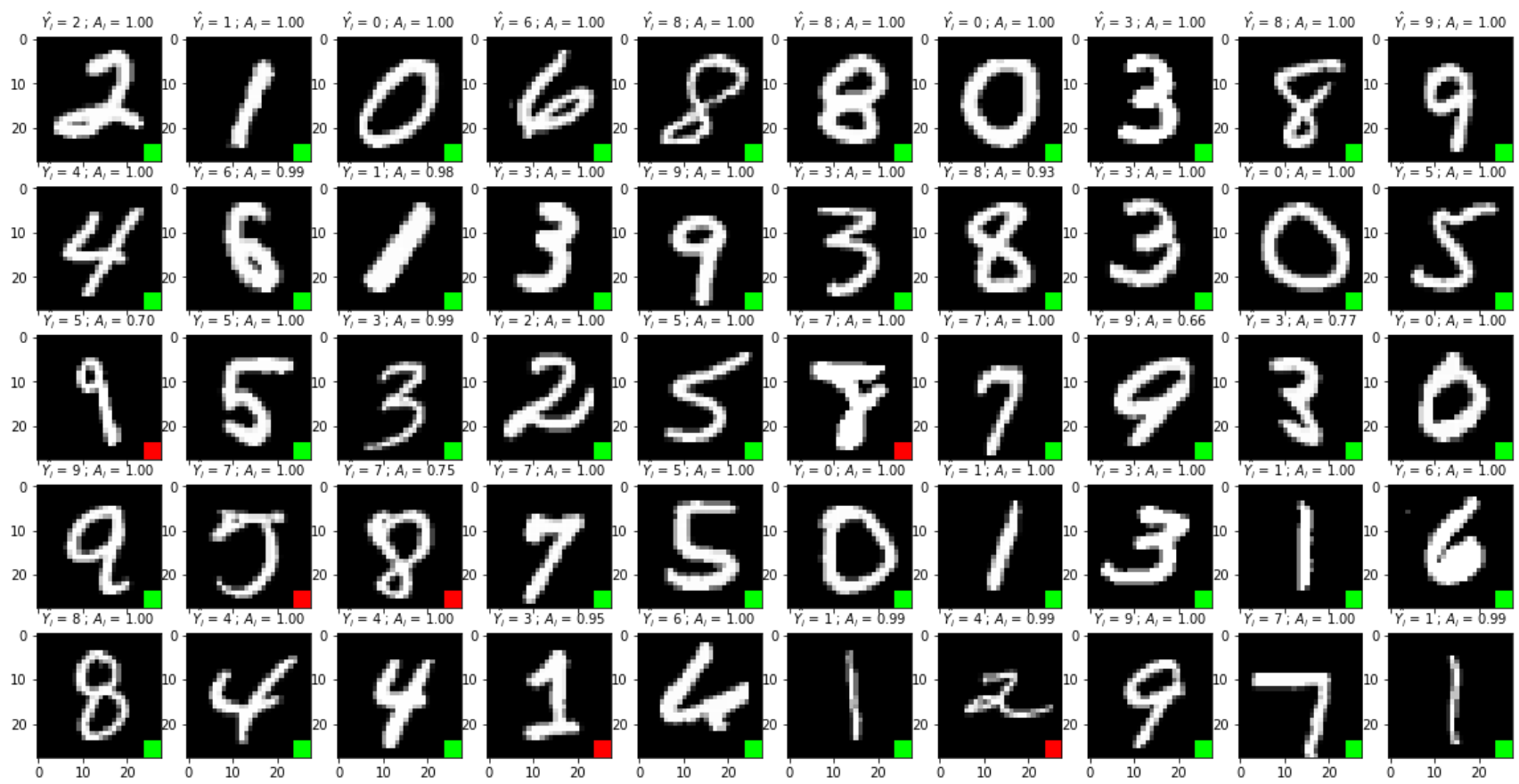

Figure 34. Prediction & Classification of handwritten digits where parameters are loss=mean_absolute_error and optimizer=Adagrad

In [ ]:
```
print('Validation')
#prediction_on_test = modelCNN.predict(X_test)
print('Test:', np.sum((Y_test_oh - prediction_on_test)**2, axis=1).mean())

prediction_on_train = modelCNN4.predict(X_train)
print('Train:', np.sum((Y_train_oh - prediction_on_train)**2, axis=1).mean())
```

```
Validation
Test: 0.14367662599288955
1875/1875 [==============================] - 3s 2ms/step
Train: 0.15298729057696556
```

Q4. Report whether the results improved or not in your opinion.

A4. The results after changing the parameters have not improved and the accuracy of the model has dropped drastically. The prediction visualization obtained shows that there are 6 digits which are wrongly classified by the model, and thus the results have not improved by changing the parameter values from its default value.

Exploring parameters of the model - 2

In [ ]:
```
# Get dimensions
N1, N2, C = X_train[0].shape

# Model
modelCNN5 = tf.keras.Sequential([
    tf.keras.layers.Input(X_train[0].shape),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(N1*N2, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

modelCNN5.summary() # Display the structure
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 26, 26, 32)        320

 conv2d_17 (Conv2D)          (None, 24, 24, 64)        18496

 max_pooling2d_8 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_16 (Dropout)        (None, 12, 12, 64)        0

 flatten_8 (Flatten)         (None, 9216)              0

 dense_16 (Dense)            (None, 784)               7226128

 dropout_17 (Dropout)        (None, 784)               0

 dense_17 (Dense)            (None, 10)                7850

=================================================================
Total params: 7,252,794
Trainable params: 7,252,794
Non-trainable params: 0
_________________________________________________________________
```

Table 7. CNN model summary

In [ ]:
```
modelCNN5.compile(
    #loss='categorical_crossentropy',       # it is used for classification where target classes are more than t
    #loss='mean_absolute_error',            # it computes the mean of absolute difference between labels and pre
    #optimizer='Adam',                      # implements the Adam algorithm
    #optimizer='Adagrad',                   # implements the Adagrad algorithm
    loss='categorical_crossentropy',
    optimizer='RMSprop',                    # implements the RMSprop algorithm
    metrics=['accuracy']
)
```

Motivation behind parameter selection

Since the accuracy drastically dropped when the parameter values of loss and optimizers were changed above to mean_absolute_error and Adagrad respectively, in the second iteration, the aim of selecting the parameters was to try an improve the model, and hence the optimizer value is changed to **RMSprop**, the top gradient descent optimizer whereas the loss value is changed to default value, i.e., **categorical_crossentropy** considering the objective of the task.

In [ ]:
```
NUMBER_OF_EPOCHS = 10
modelCNN5.fit(X_train, Y_train_oh, epochs=NUMBER_OF_EPOCHS)
```

```
Epoch 1/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.1229 - accuracy: 0.9627
Epoch 2/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0620 - accuracy: 0.9827
Epoch 3/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0579 - accuracy: 0.9836
Epoch 4/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0555 - accuracy: 0.9847
Epoch 5/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0563 - accuracy: 0.9839
Epoch 6/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0594 - accuracy: 0.9840
Epoch 7/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0585 - accuracy: 0.9841
Epoch 8/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0637 - accuracy: 0.9832
Epoch 9/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0660 - accuracy: 0.9826
Epoch 10/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0710 - accuracy: 0.9827
```

Out[56]:
```
<keras.callbacks.History at 0x7fc3d32991d0>
```

Table 8. CNN training model with number of epochs = 10

In [ ]:
```
fig, AX = plt.subplots(5, 10, sharex=True)
AX = [b for a in AX for b in a]

prediction_on_test = modelCNN5.predict(X_test)

np.random.seed(1234)
for ax in AX:
    index = np.random.randint(Y_test.size)
    # Predict
    A_ = prediction_on_test[index]
    Y_ = np.argmax(A_)
    # Prepare image to show
    img_show = np.ones((N1, N2, 3))
    img_show *= X_test[index]
    # Green square: classified correctly
    # Red square: classified wrongly
    if Y_ == Y_test[index]:
        img_show[-4:, -4:] = (0, 1, 0)
    else:
        img_show[-4:, -4:] = (1, 0, 0)
    ax.imshow(img_show)
    ax.set_title(r'$\hat{Y_i}$ = ' + str(Y_) + r' ; $A_i$ = {:.02f}'.format(float(A_[Y_])), fontsize=10)
```

```
313/313 [==============================] - 1s 2ms/step
```

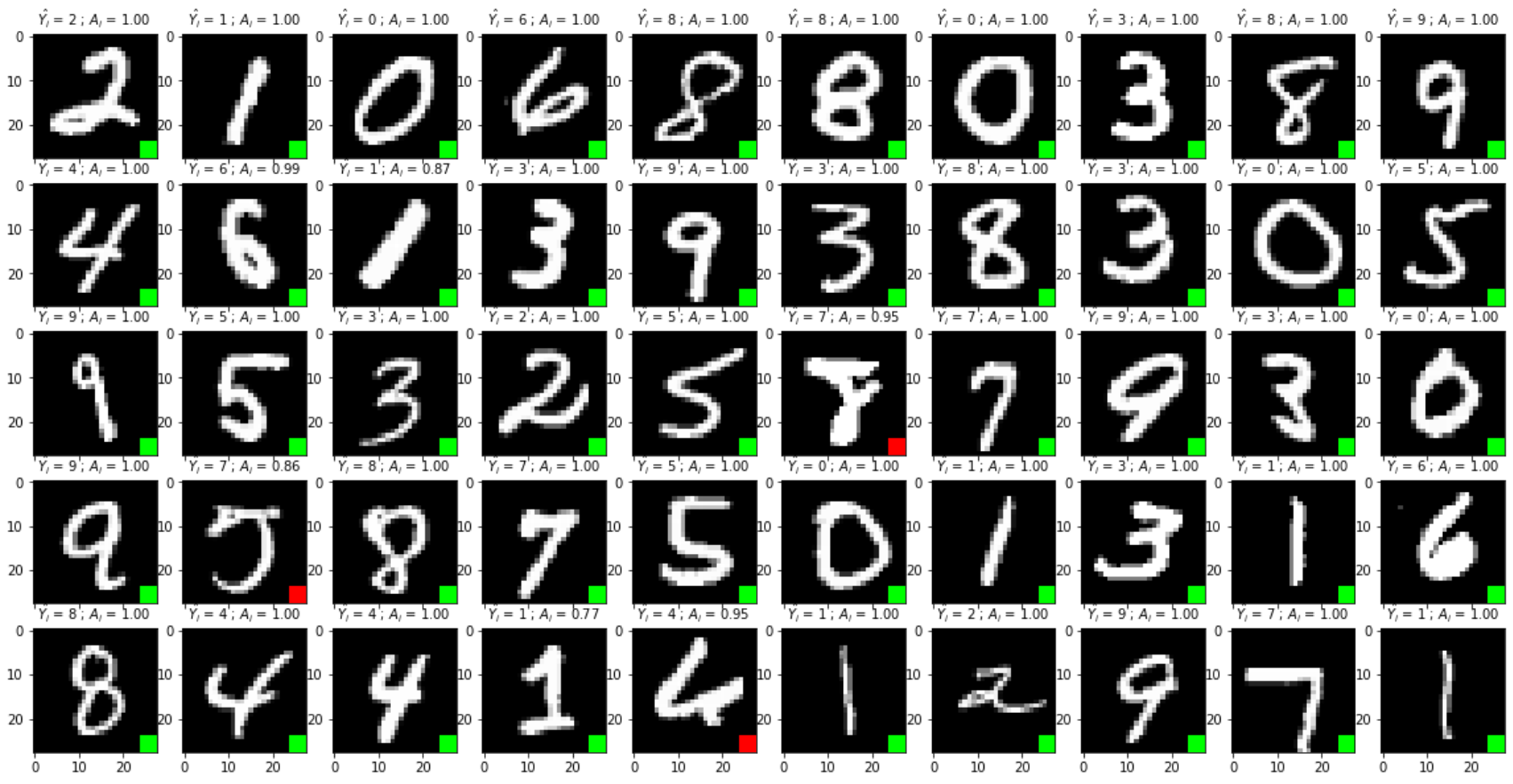

Figure 35. Prediction & Classification of handwritten digits where parameters are loss=categorical_crossentropy and optimizer=RMSprop

In [ ]:
```
print('Validation')
#prediction_on_test = modelCNN.predict(X_test)
print('Test:', np.sum((Y_test_oh - prediction_on_test)**2, axis=1).mean())

prediction_on_train = modelCNN5.predict(X_train)
print('Train:', np.sum((Y_train_oh - prediction_on_train)**2, axis=1).mean())
```

```
Validation
Test: 0.029603002820570493
1875/1875 [==============================] - 3s 2ms/step
Train: 0.025628242678109946
```

Q4. Report whether the results improved or not in your opinion.

A4. The results after changing the parameters to categorical_crossentropy and RMSprop, the accuracy of the training model has increased as compared to the previous model built, but is less as compared to the original model and hence it can be concluded that the model has not improved when compared to the original model or for the default parameters of the model, but when compared to the model with optimizer value Adagrad, the results have improved.

Changing the Hyperparameters of the CNN model / Architecture of the CNN model

Q5: Bonus task: experiment with the hyperparameters of the model and/or the architecture of the CNN, make predictions, and visualize, report, and interpret the results of the experiments. Make sure to change only one parameter at a time. Report whether the results improved or not in your opinion.

A5. Changed the hyperparameters of the CNN model in the below code and reported the results.

Model Building

(Changing the activation function)

In [ ]:
```
# Get dimensions
N1, N2, C = X_train[0].shape

# Model
modelCNN6 = tf.keras.Sequential([
    tf.keras.layers.Input(X_train[0].shape),
    tf.keras.layers.Conv2D(32, 3, activation='sigmoid'),
    tf.keras.layers.Conv2D(64, 3, activation='sigmoid'),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(N1*N2, activation='sigmoid'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

modelCNN6.summary() # Display the structure
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 26, 26, 32)        320

 conv2d_19 (Conv2D)          (None, 24, 24, 64)        18496

 max_pooling2d_9 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_18 (Dropout)        (None, 12, 12, 64)        0

 flatten_9 (Flatten)         (None, 9216)              0

 dense_18 (Dense)            (None, 784)               7226128

 dropout_19 (Dropout)        (None, 784)               0

 dense_19 (Dense)            (None, 10)                7850

=================================================================
Total params: 7,252,794
Trainable params: 7,252,794
Non-trainable params: 0
_________________________________________________________________
```

Table 9. CNN model summary for activation function = sigmoid

In [ ]:
```
modelCNN6.compile(
    loss='categorical_crossentropy',
    optimizer='Adam',
    metrics=['accuracy']
)
```

In [ ]:
```
NUMBER_OF_EPOCHS = 10
modelCNN6.fit(X_train, Y_train_oh, epochs=NUMBER_OF_EPOCHS)
```

```
Epoch 1/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.9306 - accuracy: 0.6961
Epoch 2/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2357 - accuracy: 0.9288
Epoch 3/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1285 - accuracy: 0.9599
Epoch 4/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0920 - accuracy: 0.9714
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0706 - accuracy: 0.9778
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0582 - accuracy: 0.9815
Epoch 7/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0492 - accuracy: 0.9845
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0405 - accuracy: 0.9872
Epoch 9/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0363 - accuracy: 0.9887
Epoch 10/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0342 - accuracy: 0.9886
```

Out[61]:
```
<keras.callbacks.History at 0x7fc3194f5190>
```

Table 10. CNN training model with number of epochs = 10

In [ ]:
```
fig, AX = plt.subplots(5, 10, sharex=True)
AX = [b for a in AX for b in a]

prediction_on_test = modelCNN6.predict(X_test)

np.random.seed(1234)
for ax in AX:
    index = np.random.randint(Y_test.size)
    # Predict
    A_ = prediction_on_test[index]
    Y_ = np.argmax(A_)
    # Prepare image to show
    img_show = np.ones((N1, N2, 3))
    img_show *= X_test[index]
    # Green square: classified correctly
    # Red square: classified wrongly
    if Y_ == Y_test[index]:
        img_show[-4:, -4:] = (0, 1, 0)
    else:
        img_show[-4:, -4:] = (1, 0, 0)
    ax.imshow(img_show)
    ax.set_title(r'$\hat{Y_i}$ = ' + str(Y_) + r' ; $A_i$ = {:.02f}'.format(float(A_[Y_])), fontsize=10)
```

```
313/313 [==============================] - 1s 2ms/step
```

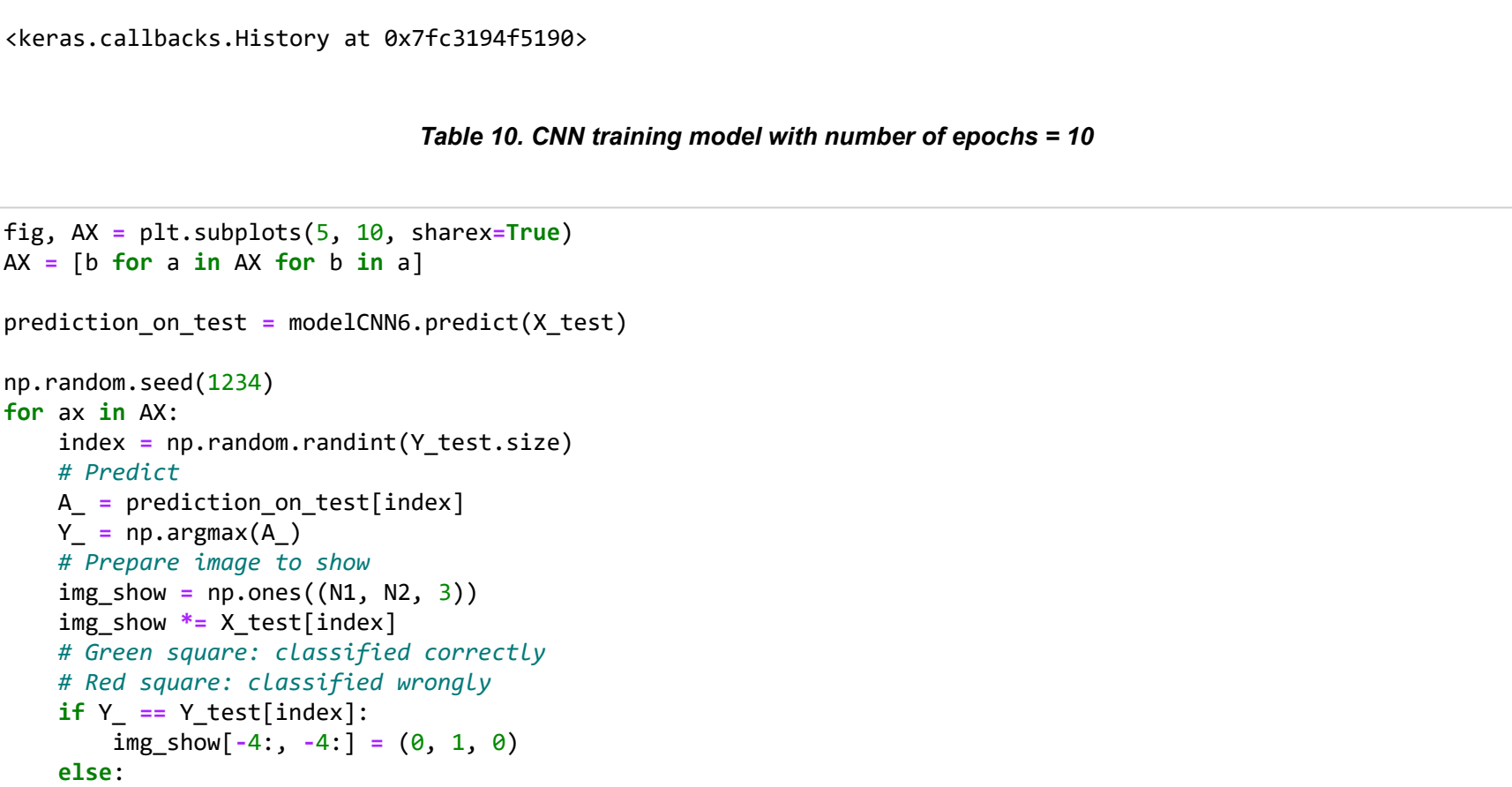

Figure 36. Prediction & Classification of handwritten digits where hyperparameters are changed (activation function=sigmoid)

In [ ]:
```
print('Validation')
#prediction_on_test = modelCNN.predict(X_test)
print('Test:', np.sum((Y_test_oh - prediction_on_test)**2, axis=1).mean())

prediction_on_train = modelCNN6.predict(X_train)
print('Train:', np.sum((Y_train_oh - prediction_on_train)**2, axis=1).mean())
```

```
Validation
Test: 0.017503157672531477
1875/1875 [==============================] - 4s 2ms/step
Train: 0.00427253611533537
```

Changing the hyperparameters - 2

(Changing the number of layers in the model)

In [ ]:
```
# Get dimensions
N1, N2, C = X_train[0].shape

# Model
modelCNN7 = tf.keras.Sequential([
    tf.keras.layers.Input(X_train[0].shape),
    tf.keras.layers.Conv2D(32, 4, activation='relu'),
    tf.keras.layers.Conv2D(64, 4, activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(N1*N2, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

modelCNN7.summary() # Display the structure
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 25, 25, 32)        544

 conv2d_21 (Conv2D)          (None, 22, 22, 64)        32832

 max_pooling2d_10 (MaxPoolin  (None, 11, 11, 64)       0
 g2D)

 dropout_20 (Dropout)        (None, 11, 11, 64)        0

 flatten_10 (Flatten)        (None, 7744)              0

 dense_20 (Dense)            (None, 784)               6072080

 dropout_21 (Dropout)        (None, 784)               0

 dense_21 (Dense)            (None, 10)                7850

=================================================================
Total params: 6,113,306
Trainable params: 6,113,306
Non-trainable params: 0
_________________________________________________________________
```

Table 11. CNN model summary for number of layers in a model = 4

In [ ]:
```
modelCNN7.compile(
    loss='categorical_crossentropy',
    optimizer='Adam',
    metrics=['accuracy']
)
```

In [ ]:
```
NUMBER_OF_EPOCHS = 10
modelCNN7.fit(X_train, Y_train_oh, epochs=NUMBER_OF_EPOCHS)
```

```
Epoch 1/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1211 - accuracy: 0.9632
Epoch 2/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0484 - accuracy: 0.9858
Epoch 3/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0380 - accuracy: 0.9883
Epoch 4/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0283 - accuracy: 0.9912
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0228 - accuracy: 0.9926
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0204 - accuracy: 0.9936
Epoch 7/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0171 - accuracy: 0.9948
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0144 - accuracy: 0.9950
Epoch 9/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0136 - accuracy: 0.9959
Epoch 10/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0130 - accuracy: 0.9959
```

Out[66]:
```
<keras.callbacks.History at 0x7fc3197a6510>
```

Table 12. CNN training model with number of epochs = 10

In [ ]:
```
fig, AX = plt.subplots(5, 10, sharex=True)
AX = [b for a in AX for b in a]

prediction_on_test = modelCNN7.predict(X_test)

np.random.seed(1234)
for ax in AX:
    index = np.random.randint(Y_test.size)
    # Predict
    A_ = prediction_on_test[index]
    Y_ = np.argmax(A_)
    # Prepare image to show
    img_show = np.ones((N1, N2, 3))
    img_show *= X_test[index]
    # Green square: classified correctly
    # Red square: classified wrongly
    if Y_ == Y_test[index]:
        img_show[-4:, -4:] = (0, 1, 0)
    else:
        img_show[-4:, -4:] = (1, 0, 0)
    ax.imshow(img_show)
    ax.set_title(r'$\hat{Y_i}$ = ' + str(Y_) + r' ; $A_i$ = {:.02f}'.format(float(A_[Y_])), fontsize=10)
```

```
313/313 [==============================] - 1s 2ms/step
```

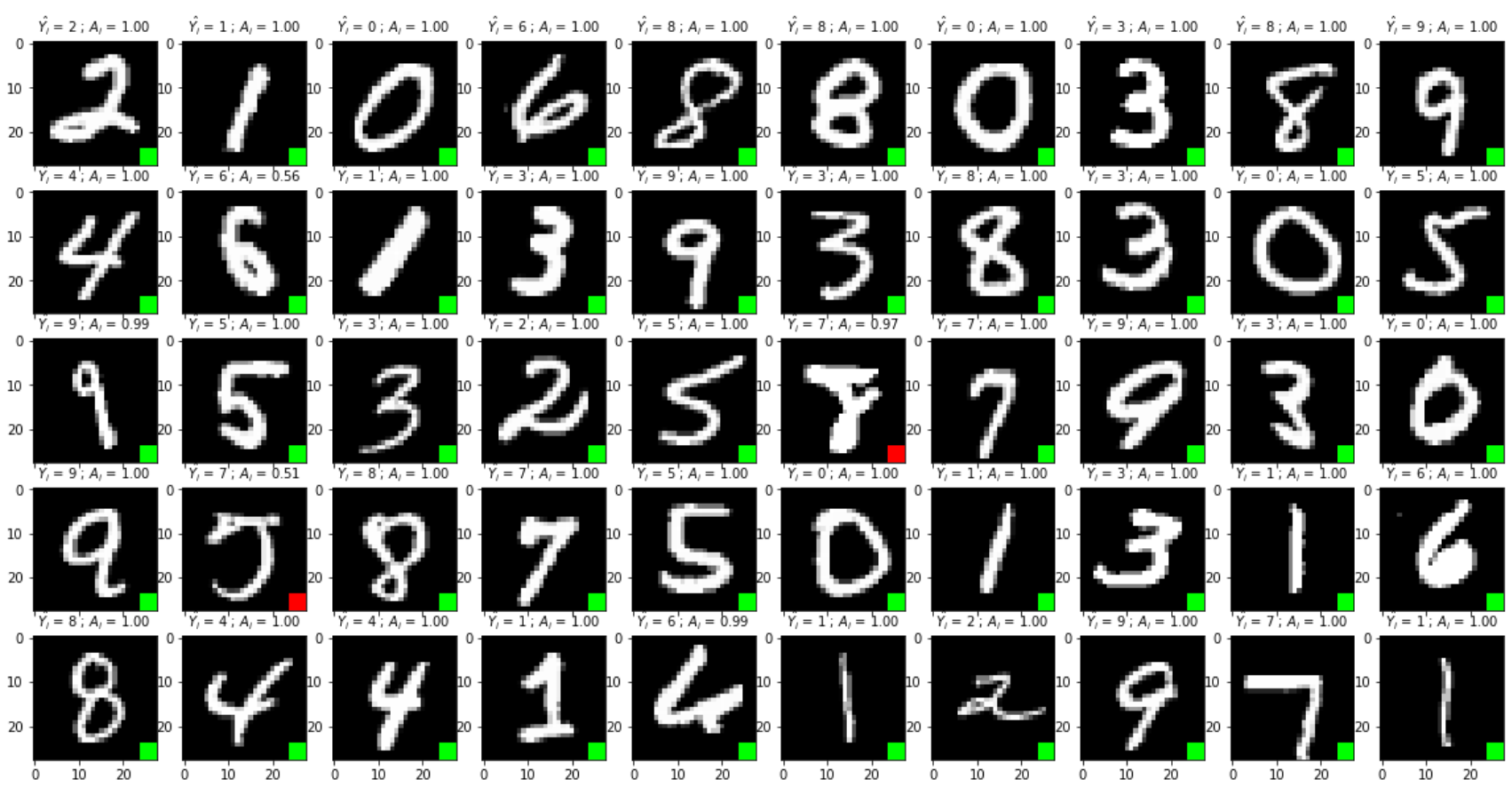

Figure 37. Prediction & Classification of handwritten digits where hyperparameters are changed (number of layers in a model = 4)

In [ ]:
```
print('Validation')
#prediction_on_test = modelCNN.predict(X_test)
print('Test:', np.sum((Y_test_oh - prediction_on_test)**2, axis=1).mean())

prediction_on_train = modelCNN7.predict(X_train)
print('Train:', np.sum((Y_train_oh - prediction_on_train)**2, axis=1).mean())
```

```
Validation
Test: 0.013523655228967105
1875/1875 [==============================] - 4s 2ms/step
Train: 0.0012902388413933974
```

Q5. Report whether the results improved or not in your opinion.

A5. It is observed that the results after changing the activation function from relu to sigmoid have slightly changed with respect to the accuracy and the model is not improved, whereas when the hyperparameter for the number of layers in the model is changed from 3 layers to 4 layers, the model has similar accuracy value but there are two digits which are wrongly classified.

The reason of choosing the number of layers in a model as 4 is because few layers of the neural network model will result in underfitting and more layers will result in overfitting of the model as that indicates the complexity of the model. [8]

Hence, in my opinion the results have not improved but is slightly affected by the changes in the hyperparameter values.

Conclusion

> Image Processing

Image Processing thus consists of the manipulation of the original images using the computer vision technology. With the help of image processing, various fields and domains are able to apply the technology for daily use of the organization.

Thus, in this assignment the goal was to manipulate the original images using the technology of image processing where the tasks that were implemented are histogram equalization, contouring of the images, grayscaling of images, fourier transformation, and highpass filtering. These techniques and the histogram graphs plotted for each image helped in understanding the difference between the frequencies of the original image and the transformed or manipulated images.

> MNIST Classification

One other application of Artificial Intelligence is handwritten digit recognition or classification which helps in the classification of the digits which are manually written. This use case is executed and build using the deep learning algorithms and neural networks, particularly the CNN algorithm. CNN algorithms are particularly used for the classification of the images that are deisgned using the keras tensorflow library.

MNIST classification is executed in this assignment where the data is loaded which has *60,000 training images and 10,000 testing images.* The task here was to train the model with the predefined number of epochs which is 10, and change the number of epochs to check if there is improvement in the model. The model was trained for number of epochs = 20 and number of epochs = 5, where it was observed that there was a slight decrease in the accuracy of the model, and the loss between the labels and predictions was a bit high as compared to the original model, thus the model did not improve.

The second task was to change the default parameters of the model and the hyperparameters of the CNN architecture in order to check for model improvement. Based on the description of each of the parameters of loss value and optimizer value, the parameters of the model were changed, but no improvement in the model was observed. Apart from this, after changing the hyperparameters of the model, i.e., the activation function and number of layers in a model, the model had no improvement and more number of the digits were wrongly classified as compared to the original model.

Hence, MNIST classification was built and executed using the Convolutional Neural Networks.

References

[1] (2022, November 23). What Is Image Processing? Overview, Applications, Benefits, and Who Should Learn It. Simplilearn.com. https://www.simplilearn.com/image-processing-article (https://www.simplilearn.com/image-processing-article)

[2] Digital Image Processing. (n.d.). https://www.tutorialspoint.com/dip/index.htm (https://www.tutorialspoint.com/dip/index.htm)

[3] Gupta, S. (2021, November 26). Newbie's Deep Learning Project to Recognize Handwritten Digit. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/11/newbies-deep-learning-project-to-recognize-handwritten-digit/ (https://www.analyticsvidhya.com/blog/2021/11/newbies-deep-learning-project-to-recognize-handwritten-digit/)

[4] Rajwal, S. (2021, July 12). Classification of Handwritten Digits Using CNN. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/07/classification-of-handwritten-digits-using-cnn/ (https://www.analyticsvidhya.com/blog/2021/07/classification-of-handwritten-digits-using-cnn/)

[5] Module: tf.keras.losses | TensorFlow v2.11.0. (n.d.). TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/losses (https://www.tensorflow.org/api_docs/python/tf/keras/losses)

[6] Maithani, M. (2021, January 27). Guide To Tensorflow Keras Optimizers. Analytics India Magazine. https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/ (https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/)

[7] Module: tf.keras.metrics | TensorFlow v2.11.0. (n.d.). TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/metrics (https://www.tensorflow.org/api_docs/python/tf/keras/metrics)

[8] Step 5: Tune Hyperparameters | Machine Learning |. (n.d.). Google Developers. https://developers.google.com/machine-learning/guides/text-classification/step-5 (https://developers.google.com/machine-learning/guides/text-classification/step-5)