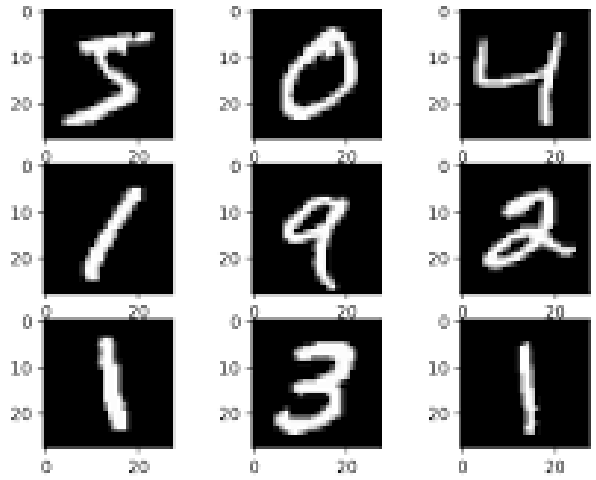
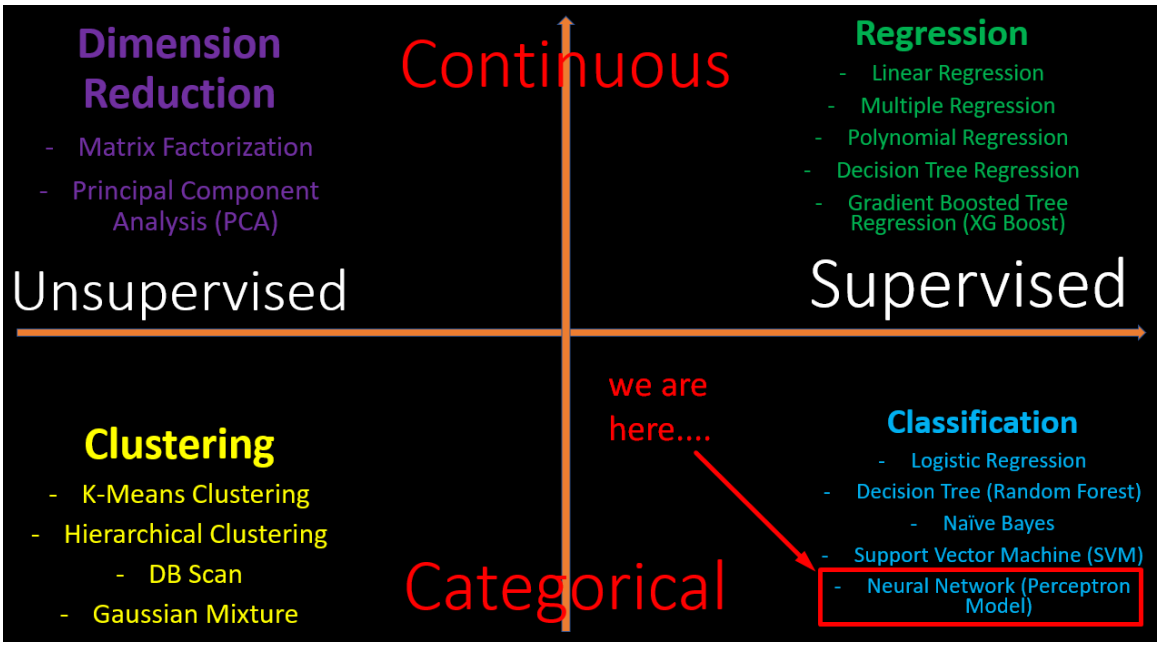
CLASSIFYING HANDWRITTEN DIGITS WITH CONVOLUTIONAL NEURAL NETWORK (CNN)

USING TENSORFLOW

DR. ALVIN ANG



CONTENTS



- Above is a table categorizing the different Machine Learning algorithms.
- Objective of Neural Network is to predict a CATEGORY.

(actually, it can also be used to predicut Regression….but most literature use it for classifying images like cats vs dogs….so we mainly use it for Classification….)

II. STEP 1: PREPARE THE DATA

https://www.alvinang.sg/s/Dr_Alvins_IBF_Day_4_CNN_on_MNIST_Digits_Dataset.ipynb

https://keras.io/api/datasets/mnist/

```
Step 1: Load Data

[77] import numpy as np
     import pandas as pd
     %matplotlib inline
     import matplotlib as mpl
     import matplotlib.pyplot as plt
```

A. IMPORT THE DATA + TRAIN TEST SPLIT

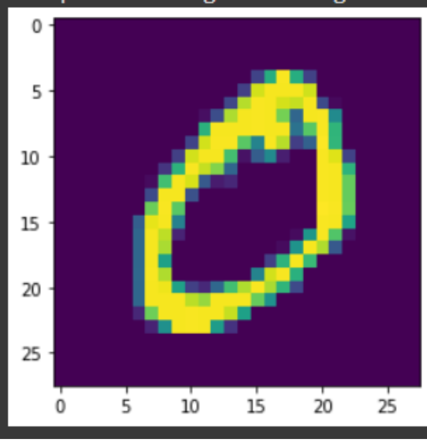
```
1a) Import the Data + Train Test Split

[78] import tensorflow as tf
     from tensorflow import keras

     mnist = keras.datasets.mnist
     (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

B. VISUALIZE THE DATA

1b) Visualize the Data

```
plt.imshow(x_train[1])
```

```
<matplotlib.image.AxesImage at 0x7f382188ad50>
```



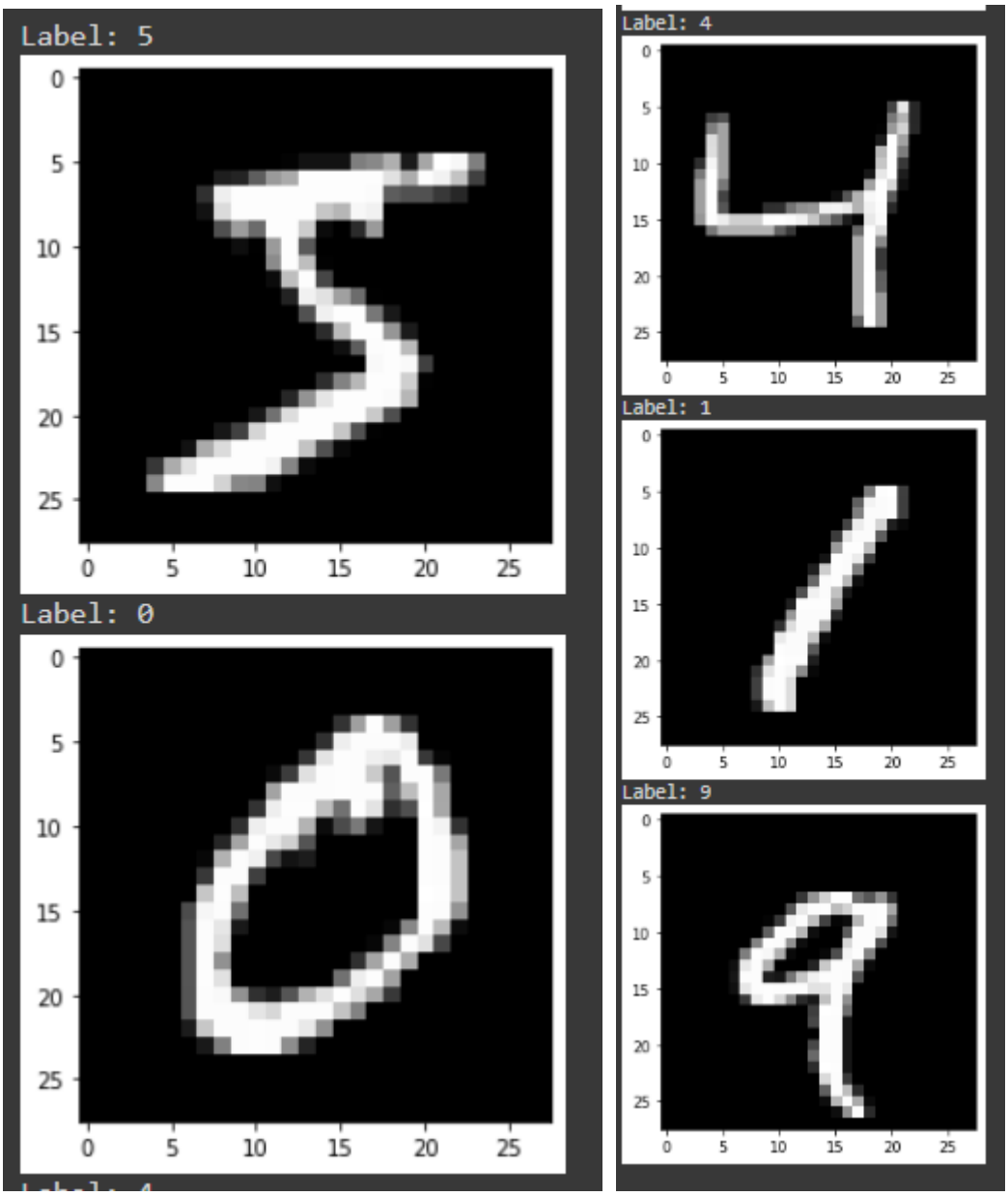

```
[80] x_train[1]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  51, 159, 253, 159,  50,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  48, 238, 252, 252, 252, 237,   0,   0,   0,   0,   0,   0,
          0,   0],
```

```
[81] x_train[1].shape

     #we see that the handwritten digit 0
     #in x_train[1] is represented by 28 rows x 28 columns of data
     #each represents 0 (black) to 255 (white)

     (28, 28)

[82] y_train[1]

     0
```

```
[83] display(y_train)

     array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

[84] print(len(np.unique(y_train)))
     #to see how many classes there are... 10 classes

     10
```

```
for i in range(5):
    print(f'Label: {y_train[i]}')
    plt.imshow(x_train[i], cmap='gray')
    plt.show()
```



C. NORMALIZE THE DATA

1c) Normalize the Data

```
[86] #normalize the dataset
     x_train, x_test = x_train/255, x_test/255
     #pictures are from grayscale 0 to 255, so we divide by 255 to
     #normalize from 0 to 1 probability

     #0 is pure black
     #255 (or 1) is pure white
```

```
[87] print(f'Before: {x_train.shape}')

     #this means there are 60,000 images inside x_train
     #and they are 28x28 pixels
```

```
Before: (60000, 28, 28)
```

D. RESHAPE THE DATA, OR RATHER, ADD IN ADDITIONAL PARAMETER → CHANNELS (RGB)

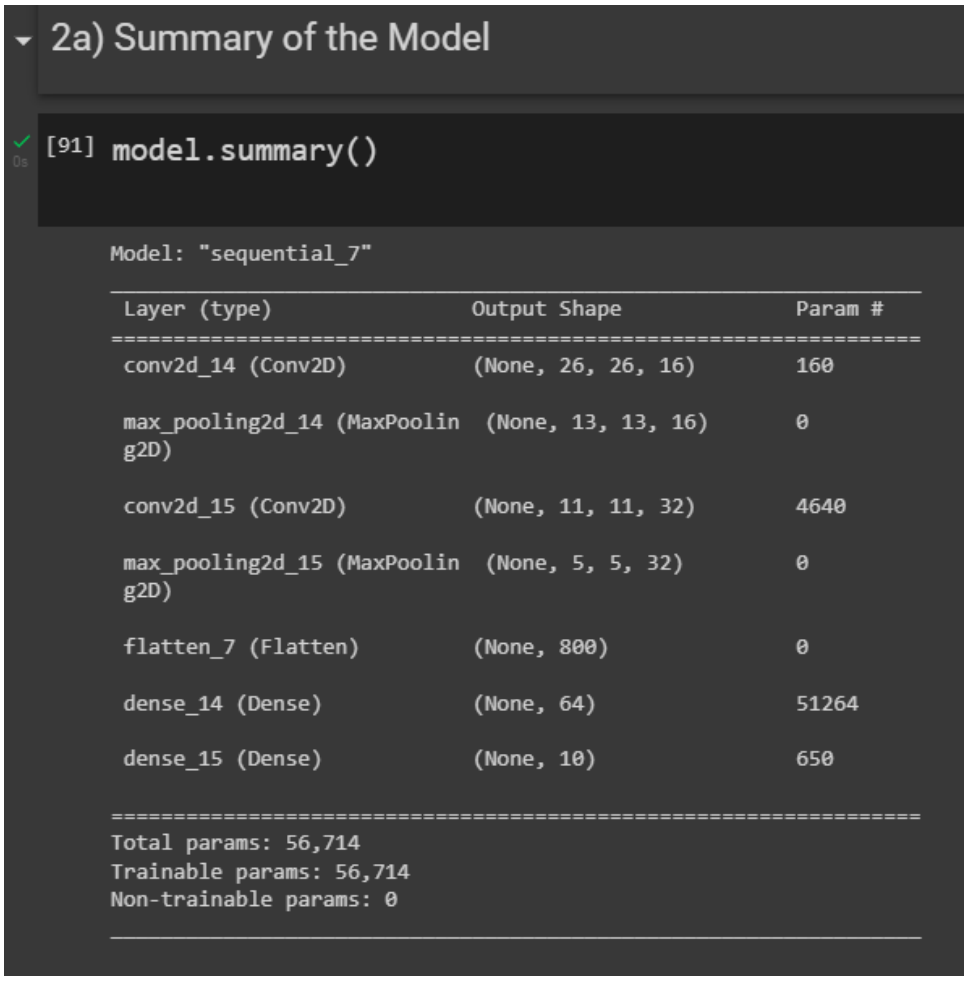
1d) Reshape the Data, or rather, add in additional Parameter --> Channels (RGB)

```
[88] #Add a channels dimension
     x_train = x_train[..., tf.newaxis]
     x_test = x_test[..., tf.newaxis]
     print(f'After: {x_train.shape}')

     #this means that we convert the x_train to include another parameter called Channel

     After: (60000, 28, 28, 1)

[89] #alternatively, to RESHAPE the data... you can try
     #x_train = x_train.reshape((60000, 28, 28, 3))
     #x_test = x_test.reshape((10000, 28, 28, 3))
```

Step 2: Build the Model

```
[90] from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

     model = Sequential([
         Conv2D(filters = 16,
                kernel_size = (3, 3),
                activation = 'relu',
                input_shape = (28,28,1)),
         MaxPooling2D((2,2)),

         Conv2D(32, (3, 3), activation = 'relu'),
         MaxPooling2D((2, 2)),

         Flatten(),
         Dense(64, activation = 'relu'),
         Dense(10, activation = 'softmax')

     ])
```

A. SUMMARY OF THE MODEL



2a) Summary of the Model

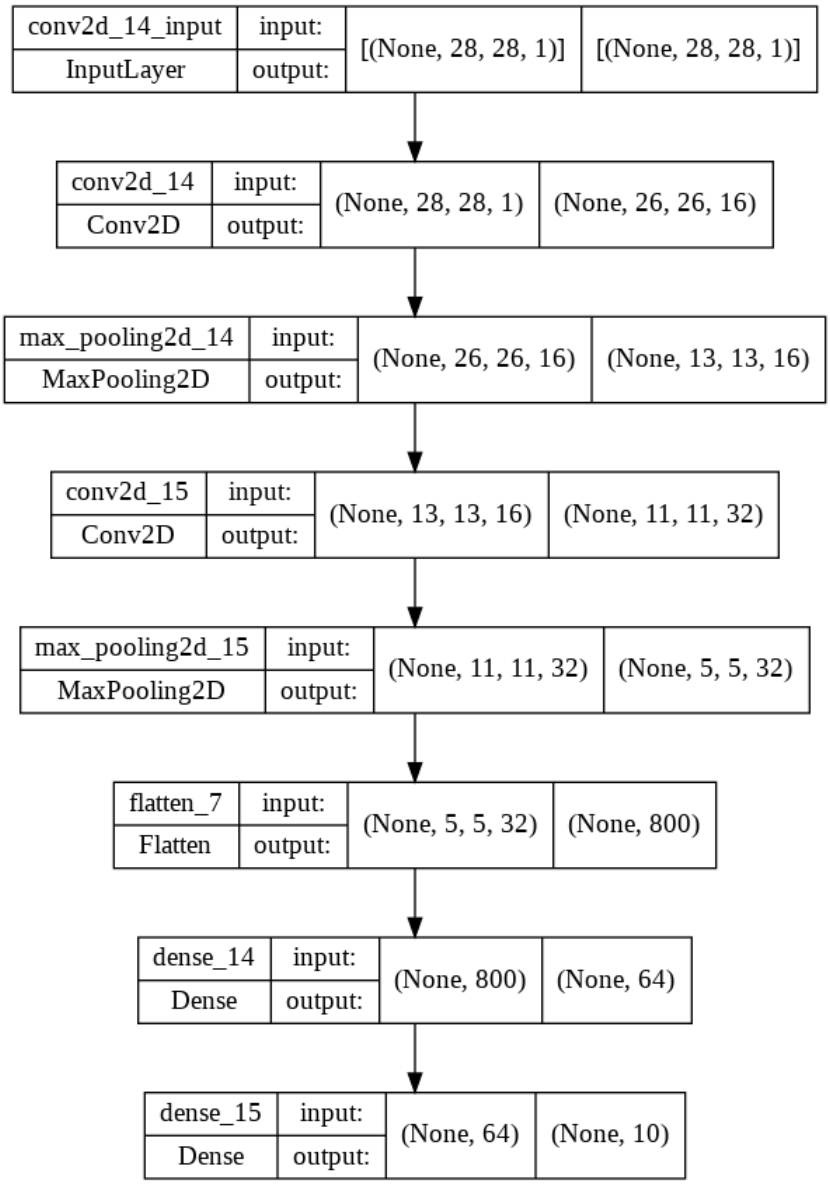
```
[91] model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 26, 26, 16)        160

 max_pooling2d_14 (MaxPoolin  (None, 13, 13, 16)       0
 g2D)

 conv2d_15 (Conv2D)          (None, 11, 11, 32)        4640

 max_pooling2d_15 (MaxPoolin  (None, 5, 5, 32)         0
 g2D)

 flatten_7 (Flatten)         (None, 800)               0

 dense_14 (Dense)            (None, 64)                51264

 dense_15 (Dense)            (None, 10)                650

=================================================================
Total params: 56,714
Trainable params: 56,714
Non-trainable params: 0
_________________________________________________________________
```

B. VISUALIZE THE MODEL

2b) Visualize the Model

```
[92] import pydot
     keras.utils.plot_model(model, 'model.png', show_shapes=True)
```



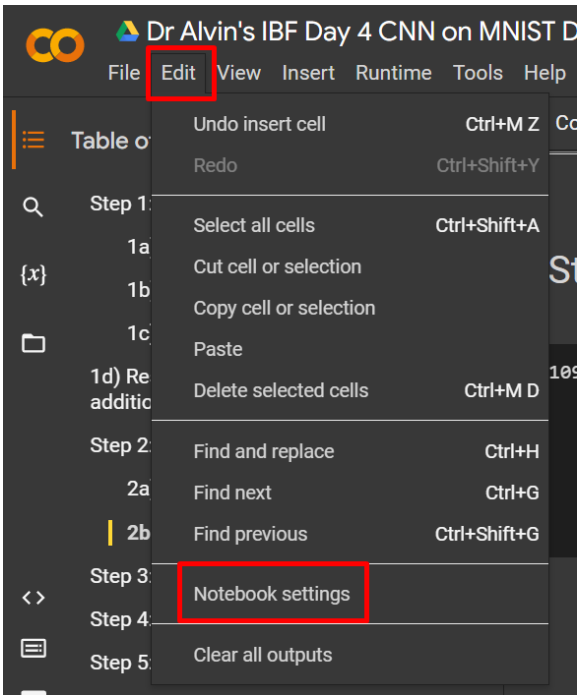
Step 3: Compile the Model

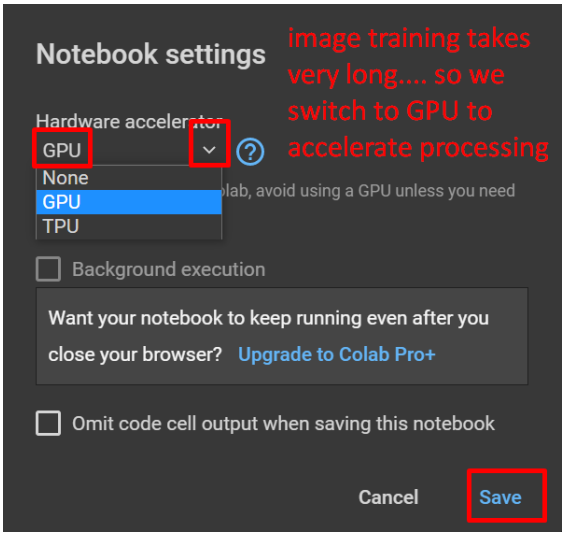
```
[93] model.compile(
        optimizer='adam',
        loss = 'sparse_categorical_crossentropy',
        metrics = ['accuracy']
    )

    #sparse_categorical_crossentropy is slower to train
    #but easier to code
    #try converting to categorical_crossentropy for faster

    #sparse_categorical_crossentropy dun need "One-Hot Encoding" for
    #both inputs and outputs

    #adam is for cross entropy --> Classification
    #RMSprop is for Regression
```

Step 4: Train the Model

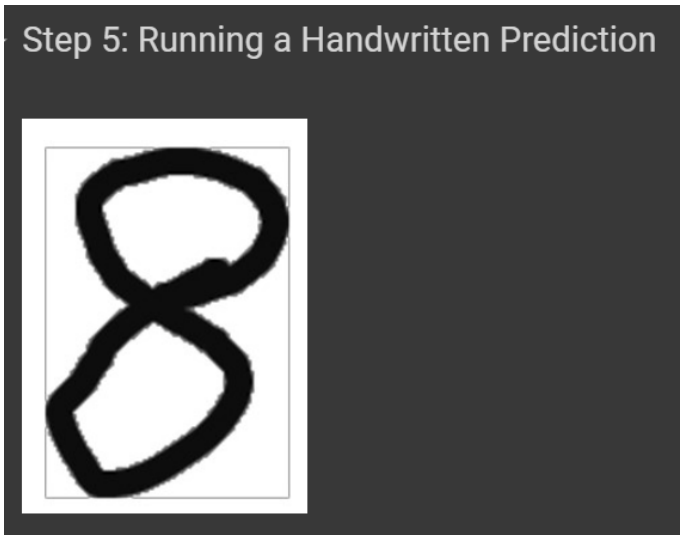
```
history = model.fit(
    x_train, y_train,
    epochs = 10,
    validation_data = (x_test, y_test)
)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0405 - accuracy: 0.9876 - val_loss: 0.0399 - val_accuracy: 0.9876
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0307 - accuracy: 0.9904 - val_loss: 0.0375 - val_accuracy: 0.9877
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0240 - accuracy: 0.9926 - val_loss: 0.0391 - val_accuracy: 0.9873
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0197 - accuracy: 0.9936 - val_loss: 0.0304 - val_accuracy: 0.9910
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0157 - accuracy: 0.9950 - val_loss: 0.0437 - val_accuracy: 0.9874
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0119 - accuracy: 0.9962 - val_loss: 0.0484 - val_accuracy: 0.9876
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0115 - accuracy: 0.9961 - val_loss: 0.0336 - val_accuracy: 0.9903
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0086 - accuracy: 0.9971 - val_loss: 0.0430 - val_accuracy: 0.9895
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0083 - accuracy: 0.9971 - val_loss: 0.0354 - val_accuracy: 0.9907
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0068 - accuracy: 0.9977 - val_loss: 0.0321 - val_accuracy: 0.9918
```





VI. STEP 5: RUNNING A HANDWRITTEN PREDICTION



A. IMPORTING IMAGE RESHAPING LIBRARIES

5a) Importing Image Reshaping Libraries

```
[110] import numpy as np
      import cv2
      from skimage import img_as_ubyte
      from skimage.color import rgb2gray
      from keras.models import load_model
```

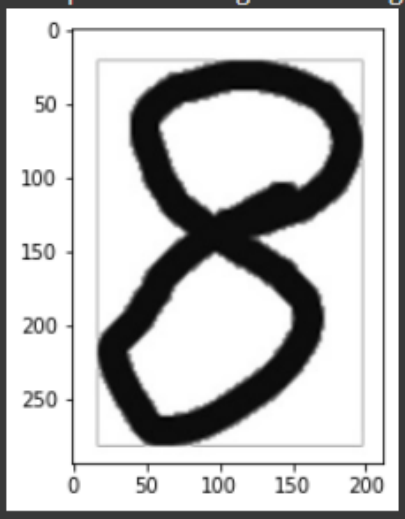
B. IMPORTING AND PREVIEWING THE HANDWRITTEN IMAGE

5b) Importing and Previewing the Handwritten Image

```
[111] eight = cv2.imread('/content/8.jpeg')
```

```
[112] plt.imshow(eight)
```

```
<matplotlib.image.AxesImage at 0x7f38b8983850>
```

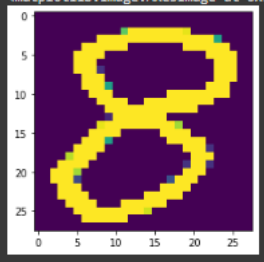


C. RE-COLORING / RE-SHAPING / RE-SIZING

5c) Re-Coloring / Re-Shaping / Re-Sizing

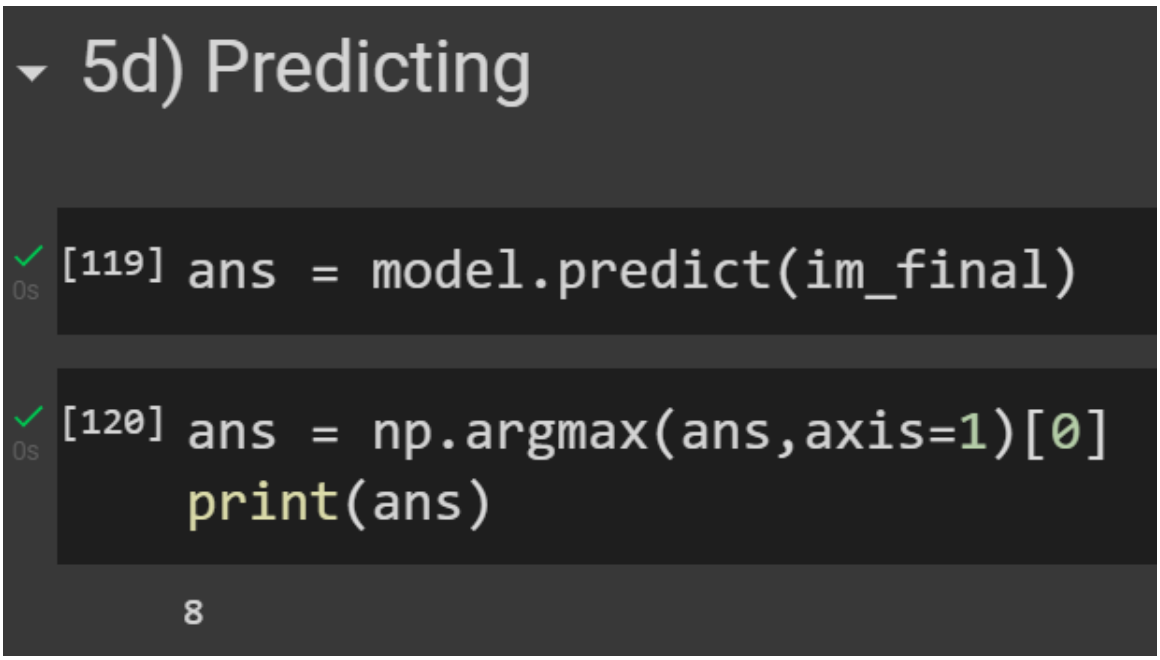
```
[113] eight_gray = rgb2gray(eight)
      eight_gray_u8 = img_as_ubyte(eight_gray)

[114] (thresh, im_binary) = cv2.threshold(eight_gray_u8, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

[115] img_resized = cv2.resize(im_binary,(28,28))

[116] im_gray_invert = 255 - img_resized

[117] plt.imshow(im_gray_invert)
```

```
<matplotlib.image.AxesImage at 0x7f38b89c0150>
```




```
[118] im_final = im_gray_invert.reshape(1,28,28,1)
```

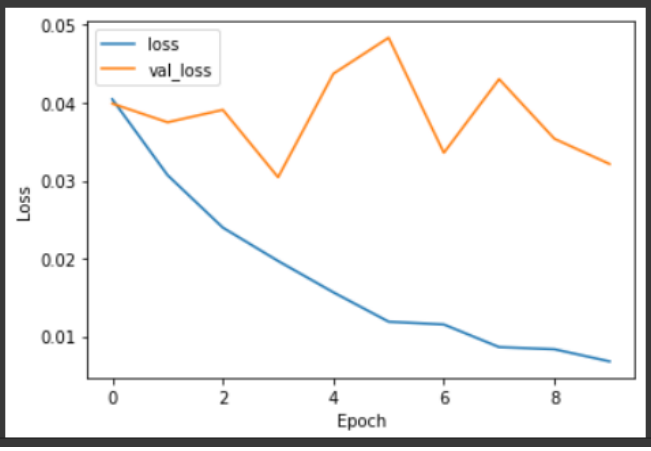
D. PREDICTING



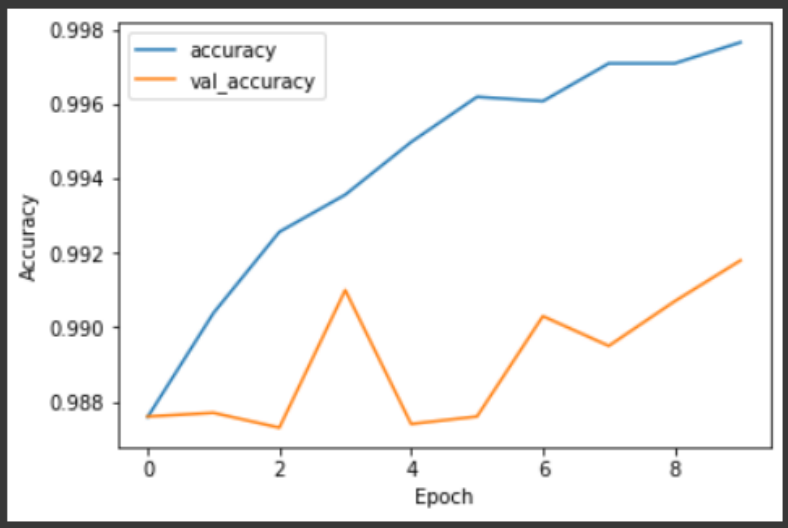
Step 6: Visualize the Loss or Error

```
[ ] loss = history.history['loss']
    val_loss = history.history['val_loss']
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    epoch = range(len(loss))
```

```
import matplotlib.pyplot as plt

plt.plot(epoch, loss, label = "loss")
plt.plot(epoch, val_loss, label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
plt.plot(epoch, acc, label = 'accuracy')
plt.plot(epoch, val_acc, label = 'val_accuracy')
plt.xlabel("Epoch")
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Step 7: Evaluate the Model

```
loss, acc = model.evaluate(x_test, y_test, verbose = 2)
#without verbose, your output gives less detail and more messy

print(f'Accuracy: {acc:5.2%}')
#string literal (putting another string into it)
#5.2%: 5 characters (including the point) with 2 decimals
```

```
313/313 - 1s - loss: 0.0321 - accuracy: 0.9918 - 555ms/epoch - 2ms/step
Accuracy: 99.18%
```

ABOUT DR. ALVIN ANG

Dr. Alvin Ang earned his Ph.D., Masters and Bachelor degrees from NTU, Singapore. He is a scientist, entrepreneur, as well as a personal/business advisor. More about him at www.AlvinAng.sg.