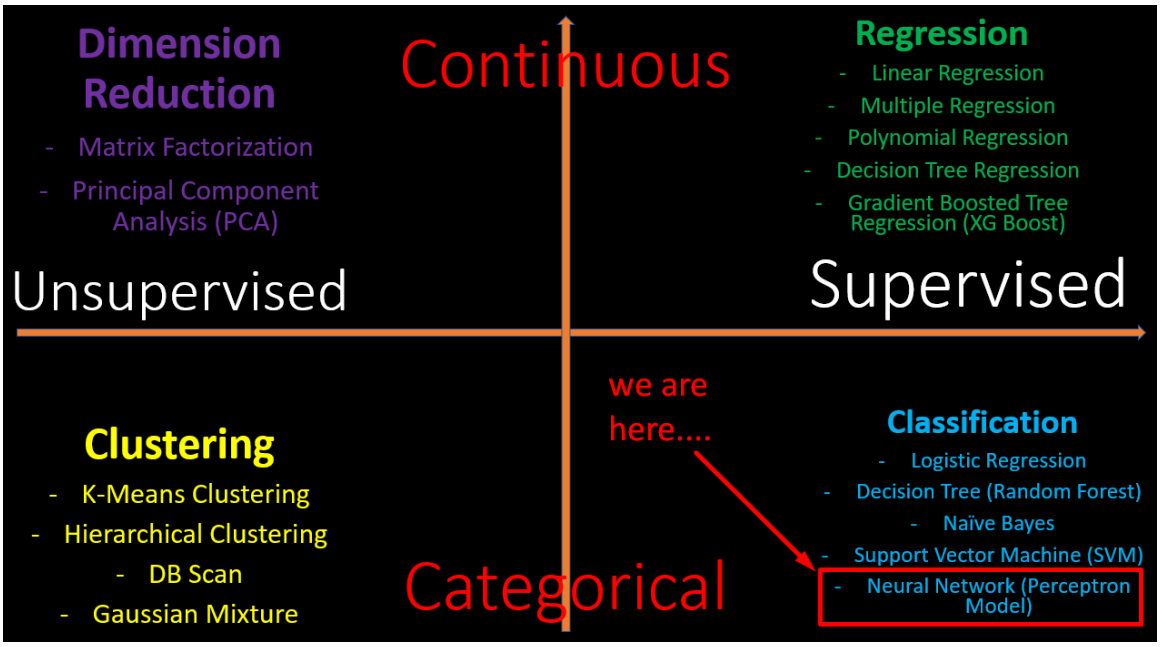
# CLASSIFICATION WITH ARTIFICIAL NEURAL NETWORK (ANN)

## 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-3-ANN-Classification.ipynb

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

Step 1: Prepare the Data

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

```
[ ] import tensorflow as tf
    from tensorflow import keras
    #tf.random.set_seed(1234).. don't think this is needed here...
```

### A. IMPORT THE DATA + TRAIN TEST SPLIT

1a) Import the Data + Train Test Split

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

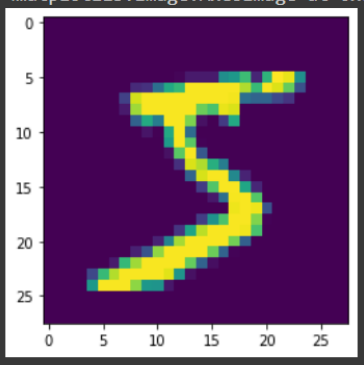
```
[ ] #import the dataset
    mnist = keras.datasets.mnist
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

## B. VISUALIZE THE DATA

1b) Visualize the Data

```
plt.imshow(x_train[0])

#this is the very first image stored in x_train
```

```
<matplotlib.image.AxesImage at 0x7f29672030d0>
```




```
y_train[0]

#this is the corresponding LABELLED value (5) given to the first image
#in x_train[0]
#Meaning, this is pre-labelled, NOT trained nor predicted output.
```

```
5
```

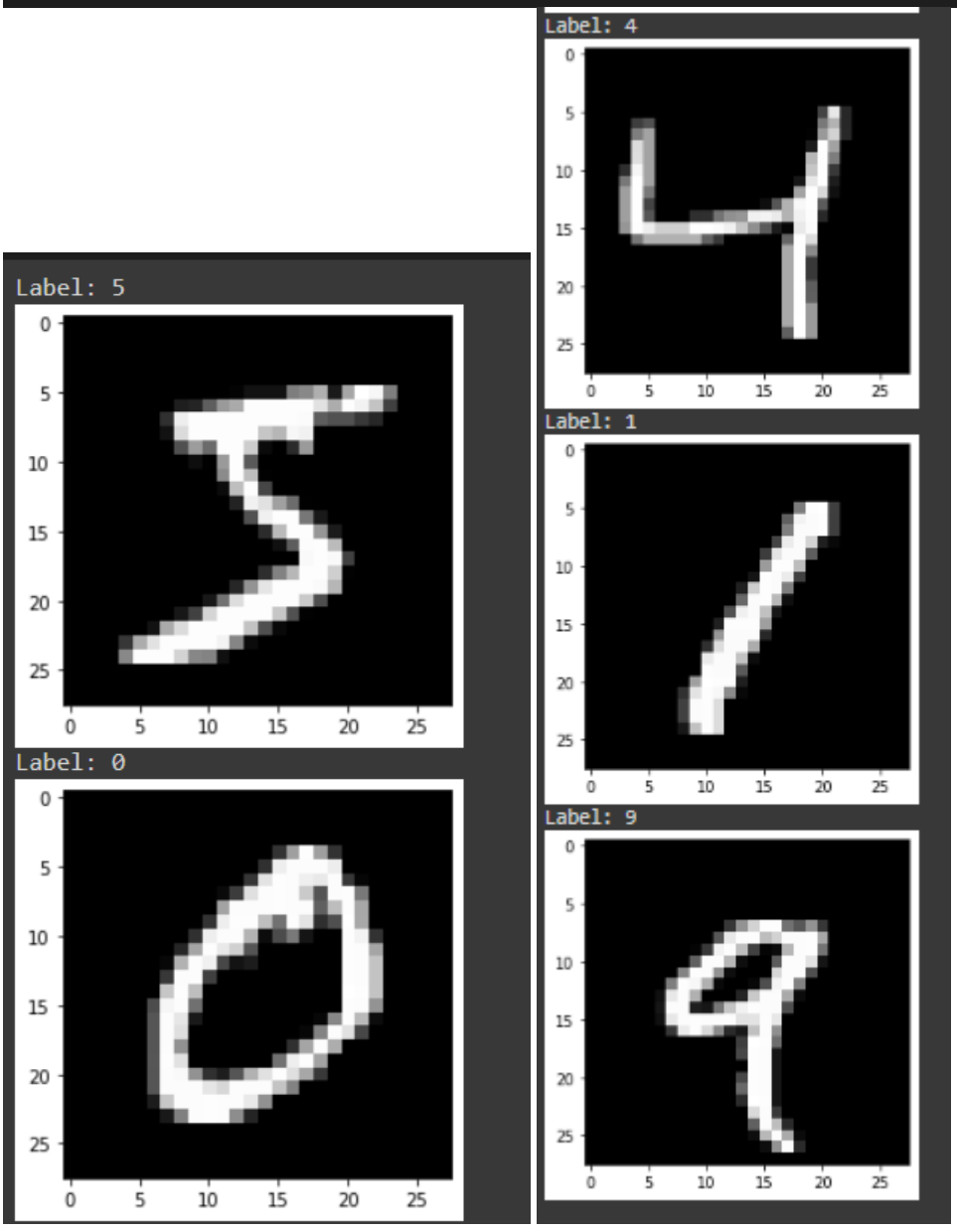
```
display(y_train)

# we preview the labels given to every single image stored
# within the x_train
```

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

```
print(len(np.unique(y_train)))
#to see how many classes there are... 10 classes (0 to 9 digits)
```

```
10
```

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

#we preview the first 5 images and their corresponding labels in the
#train dataset...
```



## C. NORMALIZE THE DATA

1c) Normalize the Data

```
[ ] #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
    #1 is pure white (or 255 is pure white)
```

## D. ONE HOT ENCODING

1d) One Hot Encoding

```
#one hot encoding is
#not needed for sparse_categorical_crossentropy
#but needed for categorical_crossentropy

from tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
0          [1 0 0 0 0 0 0 0 0 0]
1          [0 1 0 0 0 0 0 0 0 0]
2          [0 0 1 0 0 0 0 0 0 0]
3          [0 0 0 1 0 0 0 0 0 0]
4          [0 0 0 0 1 0 0 0 0 0]
5          [0 0 0 0 0 1 0 0 0 0]
6          [0 0 0 0 0 0 1 0 0 0]
7          [0 0 0 0 0 0 0 1 0 0]
8          [0 0 0 0 0 0 0 0 1 0]
9          [0 0 0 0 0 0 0 0 0 1]
```

## E. MINI EXAMPLE FOR ONE HOT ENCODING

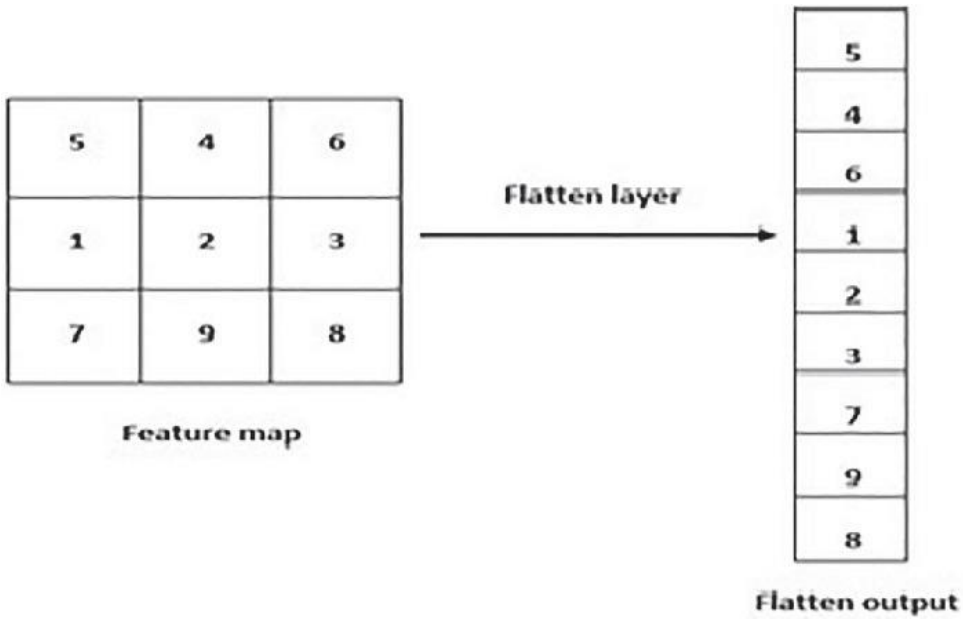
1e) Mini example for One Hot Encoding

```
[ ] #small example for to categorical

    import numpy as np

    y = np.array([0,1,2,3,1])
    print(y)
    print (to_categorical(y))
```

```
[0 1 2 3 1]
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]]
```

## Step 2: Build the Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

model = Sequential()

#Flattening means to
#convert a 28 by 28 matrix (the image) into a 784 vector
model.add(Flatten(input_shape=(28,28)))

#Dense only takes in vectors
model.add(Dense(64, activation = 'relu'))
model.add(Dense(32, activation = 'relu'))

#10 classes = 10 nodes/perceptrons
#we use softmax for 3 or more classes
model.add(Dense(y_train.shape[1], activation = 'softmax'))
```



### A. SUMMARY OF THE MODEL

## 2a) Summary of the Model

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_3 (Dense)             (None, 64)                50240

 dense_4 (Dense)             (None, 32)                2080

 dense_5 (Dense)             (None, 10)                330

=================================================================
Total params: 52,650
Trainable params: 52,650
Non-trainable params: 0
_________________________________________________________________
```

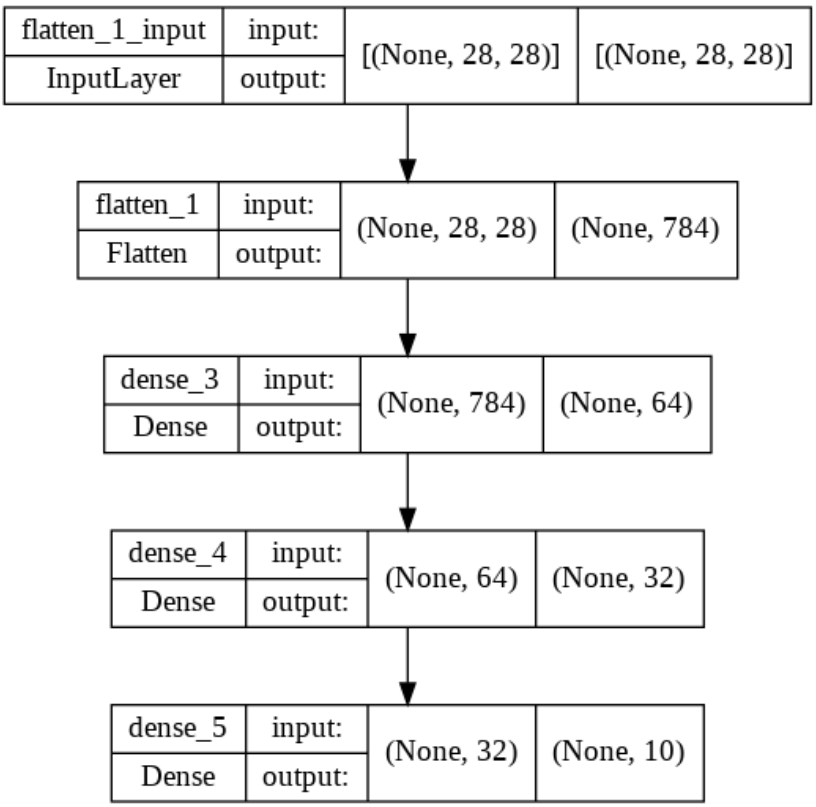
## B. VISUALIZE THE MODEL

2b) Visualize the Model

```
!pip install pydot
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pydot in /usr/local/lib/python3.7/dist-packages (1.3.0)
Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.7/dist-packages (from pydot) (3.0.9)
```

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

#u can see the model.png created in the folder
#at the left hand side
```



## Step 3: Compile the Model

adam is for classification

```
[ ] model.compile(
        optimizer = 'adam',
        loss = 'categorical_crossentropy', # output must be one-hotted
        metrics = ['accuracy'] #accuracy will be plotted out later for explanation
    )
```

- adam is the best OPTIMIZER
- https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
- how to choose which LOSS?
- https://medium.com/analytics-vidhya/activation-functions-and-loss-functions-for-neural-networks-how-to-pick-the-right-one-542e1dd523e0

| Problem Type | Last-layer Output Nodes | Hidden-layer activation | Last-layer activation | Loss function |
|---|---|---|---|---|
| Binary classification | 1 | RELU (first choice), Tanh (for RNNs) | Sigmoid | Binary Crossentropy |
| Multi-class, single-label classification | Number of classes | | Softmax | Categorical Crossentropy |
| Multi-class, multi-label classification | Number of classes | | Sigmoid (one for each class) | Binary Crossentropy |
| Regression to arbitrary values | 1 | | None | MSE |
| Regression to values between 0 and 1 | 1 | | Sigmoid | MSE/Binary Crossentropy |

| Loss function | Usage | Examples | |
|---|---|---|---|
| | | Using probabilities | Using logits |
| | | *from_logits=False* | *from_logits=True* |
| BinaryCrossentropy | Binary classification | y_true: 1<br>y_pred: 0.69 | y_true: 1<br>y_pred: 0.8 |
| CategoricalCrossentropy | Multiclass classification | y_true: 0 0 1<br>y_pred: 0.30 0.15 0.55 | y_true: 0 0 1<br>y_pred: 1.5 0.8 2.1 |
| Sparse CategoricalCrossentropy | Multiclass classification | y_true: 2<br>y_pred: 0.30 0.15 0.55 | y_true: 2<br>y_pred: 1.5 0.8 2.1 |

## V. STEP 4: TRAIN THE MODEL

Step 4: Train the Model

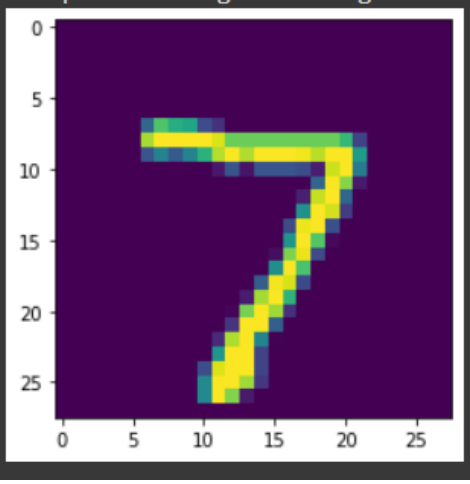
```
[ ] history = model.fit(
        x_train, y_train,
        epochs=10,
        validation_data=(x_test, y_test)
    )
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3078 - accuracy: 0.9092 - val_loss: 0.1702 - val_accuracy: 0.9494
Epoch 2/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1382 - accuracy: 0.9595 - val_loss: 0.1168 - val_accuracy: 0.9624
Epoch 3/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1015 - accuracy: 0.9689 - val_loss: 0.1006 - val_accuracy: 0.9689
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0815 - accuracy: 0.9749 - val_loss: 0.0996 - val_accuracy: 0.9690
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0672 - accuracy: 0.9789 - val_loss: 0.0899 - val_accuracy: 0.9719
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0566 - accuracy: 0.9822 - val_loss: 0.0901 - val_accuracy: 0.9733
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0478 - accuracy: 0.9850 - val_loss: 0.0900 - val_accuracy: 0.9733
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0413 - accuracy: 0.9867 - val_loss: 0.0900 - val_accuracy: 0.9747
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0370 - accuracy: 0.9880 - val_loss: 0.0926 - val_accuracy: 0.9744
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0326 - accuracy: 0.9887 - val_loss: 0.0991 - val_accuracy: 0.9737
```

## Step 5: Run a Prediction

```
plt.imshow(x_test[0])

#preview the image inside x_test[0]
```

```
<matplotlib.image.AxesImage at 0x7f29675282d0>
```




```
[ ] y_test[0]

    #it was originally lablled as 7 (one hot encoded below)

    array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)

[ ] a = model.predict(x_test)
    #fit the x_test to the model
```

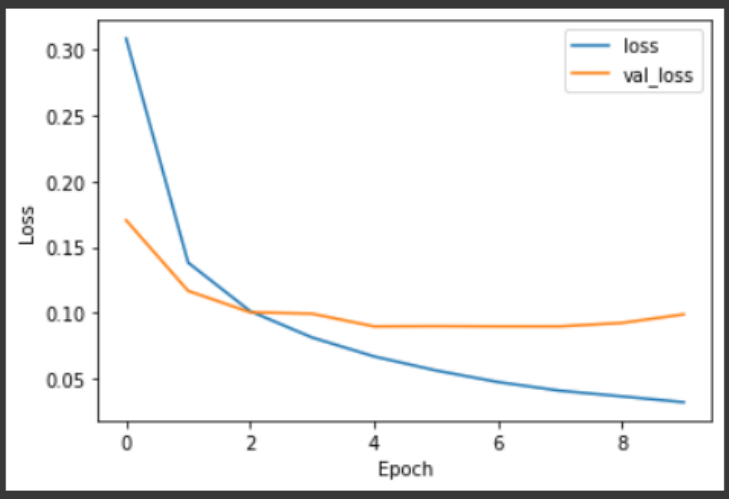
```
[ ] a[0]
    # the model works!
    # the predicted value is 99.999% change that its a '7'
    # as can be seen in the one hot encoded values below
    #(after the softmax layer)

    #however, note that ANN is not meant to be used for image processing
    #CNN is better used for image processing....
    #thus this is just a showcase (prelimnary step before we use CNN)
```

```
array([2.1622395e-11, 3.7964185e-08, 2.0395939e-08, 2.3157761e-06,
       3.1118405e-13, 5.4209579e-08, 1.9920401e-18, 9.9999762e-01,
       1.0898040e-08, 1.9527086e-08], dtype=float32)
```

## 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))
```

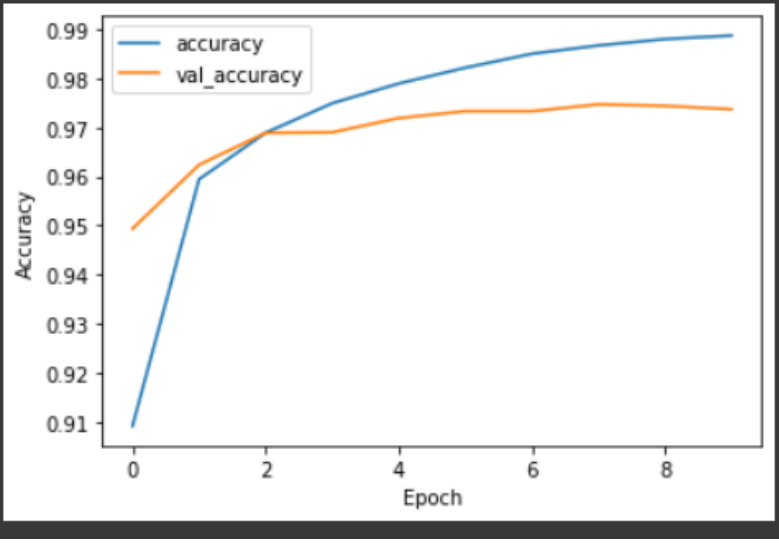
```
import matplotlib.pyplot as plt

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



## A. VISUALIZE THE ACCURACY

6a) Visualize the Accuracy

```
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)

    print(f'Testing Accuracy{acc:5.2f}')
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0991 - accuracy: 0.9737
Testing Accuracy 0.97
```

## 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.