Import The Libraries

In [1]:
```
import os
import numpy as np
import tensorflow as tf

from PIL import Image

from time import strftime

from numpy.random import seed
```

Stabilize

In [2]:
```
seed(888)
tf.random.set_seed(404)
```

Constants

In [3]:
```
# Relative
X_TRAIN_PATH = 'X_Train.csv'
Y_TRAIN_PATH = 'Y_Train.csv'

X_TEST_PATH = 'X_Test.csv'
Y_TEST_PATH = 'Y_Test.csv'

LOGGING_PATH = 'TensorBoard/'

NR_CLASSES = 10
VALIDATION_SIZE = 10000

CHANNELS = 1
IMAGE_WIDTH = 28
IMAGE_HEIGHT = 28

TOTAL_INPUTS = IMAGE_WIDTH * IMAGE_HEIGHT * CHANNELS
```

Get The Data

In [4]:
```
%%time

y_train_all = np.loadtxt(Y_TRAIN_PATH, delimiter=',', dtype=int)
```

```
Wall time: 398 ms
```

In [5]:
```
y_train_all.shape
```

Out[5]:
```
(60000,)
```

In [6]:
```
%%time

y_test = np.loadtxt(Y_TEST_PATH, delimiter=',', dtype=int)
```

```
Wall time: 88 ms
```

In [7]:
```
%%time

x_train_all = np.loadtxt(X_TRAIN_PATH, delimiter=',', dtype=int)
```

```
Wall time: 32.1 s
```

In [8]:
```
%%time

x_test = np.loadtxt(X_TEST_PATH, delimiter=',', dtype=int)
```

```
Wall time: 5.43 s
```

Explore

In [9]:
```
# Data
# 28 x 28 x 1 = 784
x_train_all.shape
```

Out[9]:
```
(60000, 784)
```

In [10]:
```
# 00 - W
# FF - B
x_train_all[0]
```

Out[10]:
```
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,   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,   3,  18,  18,  18,
       126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
       253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253,
       253, 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 219, 253,
       253, 253, 253, 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        80, 156, 107, 253, 253, 205,  11,   0,  43, 154,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,  14,   1, 154, 253,  90,   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, 139, 253, 190,   2,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190, 253,  70,
```

```
          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,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,  45, 186, 253, 253, 150,  27,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,  16,  93, 252, 253, 187,
          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, 249,
        253, 249,  64,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  46, 130,
        183, 253, 253, 207,   2,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39, 148,
        229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114,
        221, 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  23,  66,
        213, 253, 253, 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 171,
        219, 253, 253, 253, 253, 195,  80,   9,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  55, 172,
        226, 253, 253, 253, 253, 244, 133,  11,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        136, 253, 253, 253, 212, 135, 132,  16,   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])
```

In [11]:
```
# Labels
y_train_all.shape
```

Out[11]: (60000,)

In [12]:
```
# Data
x_test.shape
```

Out[12]: (10000, 784)

In [13]:
```
# Labels
y_train_all[:5]
```

Out[13]: array([5, 0, 4, 1, 9])

Data Pre-Processing

In [14]:
```
# Rescaling Data
# I2F
# W : 0
# B : 1
x_train_all, x_test = x_train_all / 255.0, x_test / 255.0
```

Encoding

In [15]:
```
# AEI - Array Element Indexing
values = y_train_all[:5]
np.eye(10)[values]
```

Out[15]:
```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

In [16]:
```
np.eye(10)
```

Out[16]:
```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

In [17]:
```
np.eye(10)[2]
```

Out[17]:
```
array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.])
```

In [18]:
```
values
```

Out[18]:
```
array([5, 0, 4, 1, 9])
```

In [19]:
```
values[4]
```

Out[19]:
```
9
```

In [20]:
```
y_train_all = np.eye(NR_CLASSES)[y_train_all]
```

In [21]:
```
y_train_all.shape
```

Out[21]:
```
(60000, 10)
```

In [22]:
```
y_test = np.eye(NR_CLASSES)[y_test]
```

In [23]:
```
y_test.shape
```

Out[23]: (10000, 10)

Create Validation Dataset From Training Data

> Split The Training Dataset Into A Smaller Training Dataset And A Validation Dataset
> For The Features And The Labels

In [24]:
```
# Extract 0 -> 10,000
x_val = x_train_all[:VALIDATION_SIZE]
y_val = y_train_all[:VALIDATION_SIZE]
```

In [25]:
```
# Extract 10,000 -> 50,000
x_train = x_train_all[VALIDATION_SIZE:]
y_train = y_train_all[VALIDATION_SIZE:]
```

In [26]:
```
x_train.shape
```

Out[26]: (50000, 784)

In [27]:
```
x_val.shape
```

Out[27]: (10000, 784)

Setup Tensorflow Graph

> - RGB : Rank = 3
> - B/W : Rank = 1

In [28]:
```
tf.compat.v1.disable_eager_execution()

# Features
X = tf.compat.v1.placeholder(tf.float32, shape=[None, TOTAL_INPUTS], name='X')

# Labels
Y = tf.compat.v1.placeholder(tf.float32, shape=[None, NR_CLASSES], name='labels')
```

Neural Network Architecture

> Hyper Parameters

In [29]:
```
nr_epochs = 16

# Scientific Notation
learning_rate = 1e-3

# Neurons
```

```
n_hidden1 = 512
n_hidden2 = 64
```

In [30]:

```
# Initial Setup

# Initialization --> Calculations
# Initializing W & B

# MatMul -> Matrix Multiplication
# Last Layer -> SoftMax --> Probabilities Associated With Each Of The Outputs

def setup_layer(input, weight_dim, bias_dim, name):
    with tf.name_scope(name):
        initial_w = tf.compat.v1.truncated_normal(shape=weight_dim, stddev=0.1, seed=42
        w = tf.Variable(initial_value=initial_w, name='W')

        initial_b = tf.constant(value=0.0, shape=bias_dim)
        b = tf.Variable(initial_value=initial_b, name='B')

        layer_in = tf.matmul(input, w) + b

        if name == 'out':
            layer_out = tf.nn.softmax(layer_in)
        else:
            layer_out = tf.nn.relu(layer_in)

        tf.summary.histogram('Weights', w)
        tf.summary.histogram('Biases', b)

        return layer_out
```

In [31]:

```
layer_1 = setup_layer(X, weight_dim=[TOTAL_INPUTS, n_hidden1],
                      bias_dim=[n_hidden1], name='layer_1')

layer_drop = tf.compat.v1.nn.dropout(layer_1, rate=1 - 0.8, name='dropout_layer')

layer_2 = setup_layer(layer_drop, weight_dim=[n_hidden1, n_hidden2],
                      bias_dim=[n_hidden2], name='layer_2')

output = setup_layer(layer_2, weight_dim=[n_hidden2, NR_CLASSES],
                     bias_dim=[NR_CLASSES], name='out')

model_name = f'{n_hidden1}-DO-{n_hidden2} LR{learning_rate} E{nr_epochs}'
```

TensorBoard Setup

In [32]:

```
# Folder for TensorBoard

folder_name = f'{model_name}'
directory = os.path.join(LOGGING_PATH, folder_name)

try:
    os.makedirs(directory)
except OSError as exception:
    print(exception.strerror)
```

```
else:
    print('Successfully Created Directories!')
```

```
Successfully Created Directories!
```

Loss, Optimisation & Metrics

> Defining Loss Function

In [33]:
```
# Labels -> Actual Labels That We Supply
# Logits -> Outputs From The Output Layer
# We Have Individual Batches, So We Need To Take Average Of Loss
with tf.name_scope('Loss_Calc'):
    loss = tf.reduce_mean(tf.compat.v1.nn.softmax_cross_entropy_with_logits_v2(labels=Y
```

> Defining Optimizer

In [34]:
```
# O shape and mold the model into its most accurate possible form by futzing with the w
# Adaptive Moment Estimation
# Goal: Minimizing our loss function.
with tf.name_scope('Optimizer'):
    optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate)
    train_step = optimizer.minimize(loss)
```

> Accuracy Metric

In [35]:
```
# Compare Output Probability Prediction With Actual Label
with tf.name_scope('Accuracy_Calc'):
    correct_pred = tf.equal(tf.argmax(output, axis=1), tf.argmax(Y, axis=1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

In [36]:
```
with tf.name_scope('Performance'):
    tf.compat.v1.summary.scalar('Accuracy', accuracy)
    tf.compat.v1.summary.scalar('Cost', loss)
```

Check Input Images in Tensorboard

In [37]:
```
with tf.name_scope('Show_Image'):
    x_image = tf.reshape(X, [-1, 28, 28, 1])
    tf.compat.v1.summary.image('Image_Input', x_image, max_outputs=4)
```

Run Session

In [38]:
```
# Initialize Session Obj
sess = tf.compat.v1.Session()
```

> Setup Filewriter & Merge Summaries

In [39]:
```
merged_summary = tf.compat.v1.summary.merge_all()

train_writer = tf.compat.v1.summary.FileWriter(directory + '/Train')
train_writer.add_graph(sess.graph)

validation_writer = tf.compat.v1.summary.FileWriter(directory + '/Validation')
```

> Initialize All The Variables

In [40]:
```
init = tf.compat.v1.global_variables_initializer()
sess.run(init)
```

Batching The Data

In [41]:
```
size_of_batch = 1000
```

In [42]:
```
num_examples = y_train.shape[0]
nr_iterations = int(num_examples / size_of_batch)

index_in_epoch = 0
```

In [43]:
```
def next_batch(batch_size, data, labels):
    global num_examples
    global index_in_epoch

    start = index_in_epoch
    index_in_epoch += batch_size

    if index_in_epoch > num_examples:
        start = 0
        index_in_epoch = batch_size

    end = index_in_epoch

    return data[start:end], labels[start:end]
```

Training Loop

In [44]:
```
for epoch in range(nr_epochs):

    # ================== Training Dataset ================

    for i in range(nr_iterations):
        batch_x, batch_y = next_batch(batch_size=size_of_batch, data=x_train, labels=y_

        feed_dictionary = {X: batch_x, Y: batch_y}
```

```
        sess.run(train_step, feed_dict=feed_dictionary)

    s, batch_accuracy = sess.run(fetches=[merged_summary, accuracy], feed_dict=feed_dic

    train_writer.add_summary(s, epoch)

    print(f'Epoch {epoch} \t| Training Accuracy = {batch_accuracy}')

    # ================== Validation ======================

    summary = sess.run(fetches=merged_summary, feed_dict={X: x_val, Y: y_val})
    validation_writer.add_summary(summary, epoch)

print('Successfully Trained The Model, Ready For Use!:)')
```

```
Epoch 0         | Training Accuracy = 0.8460000157356262
Epoch 1         | Training Accuracy = 0.8640000224113464
Epoch 2         | Training Accuracy = 0.8659999966621399
Epoch 3         | Training Accuracy = 0.8650000095367432
Epoch 4         | Training Accuracy = 0.8759999871253967
Epoch 5         | Training Accuracy = 0.875
Epoch 6         | Training Accuracy = 0.8809999823570251
Epoch 7         | Training Accuracy = 0.9710000157356262
Epoch 8         | Training Accuracy = 0.9789999723434448
Epoch 9         | Training Accuracy = 0.984000027179718
Epoch 10        | Training Accuracy = 0.9829999804496765
Epoch 11        | Training Accuracy = 0.9869999885559082
Epoch 12        | Training Accuracy = 0.9860000014305115
Epoch 13        | Training Accuracy = 0.9869999885559082
Epoch 14        | Training Accuracy = 0.9869999885559082
Epoch 15        | Training Accuracy = 0.9829999804496765
Successfully Trained The Model, Ready For Use!:)
```

Make A Prediction & Calculate Output For The Input Image Given By The User

In [45]:
```
img = Image.open('User_Input/8/IMG (17).jpg')
```

In [46]:
```
# Display Input Taken From User
img
```

Out[46]:

In [47]:
```
img_array = np.array(img)
```

In [48]:
```
img_array.shape
```

Out[48]: (28, 28)

In [49]:
```
test_img = img_array.ravel()
```

In [50]:
```
test_img.shape
```

Out[50]: (784,)

In [51]:
```
prediction = sess.run(fetches=tf.argmax(output, axis=1), feed_dict={X:[test_img]})
```

In [52]:
```
print(f'Prediction For Input Image From The User = {prediction} :)')
```

```
Prediction For Input Image From The User = [8] :)
```

Testing And Evaluation

In [53]:
```
test_accuracy = sess.run(fetches=accuracy, feed_dict={X:x_test, Y:y_test})
print(f'Accuracy On User Input Set Is {test_accuracy:0.2%}')
```

```
Accuracy On User Input Set Is 97.04%
```

Reset For The Next Run

In [54]:
```
# train_writer.close()
# validation_writer.close()
# sess.close()
# tf.compat.v1.reset_default_graph()
```