```
!pip install jupyter-dash
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: jupyter-dash in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: dash in /usr/local/lib/python3.7/dist-packages (from j
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: flask in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: ipykernel in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: retrying in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: ipython in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: ansi2html in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: flask-compress in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/python3.7/di
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from pl
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: brotli in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/d
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: tornado>=4.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pexpect in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
```

```
import plotly.express as px
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html
```

✓ 0s completed at 2:14 AM

webpage

Google Drive

```
# Connectiong to the Google Drive for the permanent storage
from google.colab import drive
drive.mount('/content/drive')
```

```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
# Checking if the correct version of tensorflow in installed or not
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

```
     TensorFlow version: 2.8.2
```

```
from numpy.random import seed
seed(786)
# Tensor flow for the fast calculation or computation
import tensorflow
tensorflow.random.set_seed(786)
```

```
# Other imports like os for directory management and numpy for mathematical operation on ar
import os

# Numpy for performing mathematical operations on arrays
import numpy as np

# Importing pandas for data visualization in the forms of tables or data frames
import pandas as pd

# Time for dealing with real time and the pillow for dealing with the image for predictions
from time import strftime

# Pillow for the image used for prediction
from PIL import Image
```

```
# Loading all the files from the directory into variables
```

```
X_TRAINING_PATH =  '/content/drive/MyDrive/MNIST/digit_xtrain.csv'
X_TESTING_PATH =   '/content/drive/MyDrive/MNIST/digit_xtest.csv'
Y_TRAINING_PATH =  '/content/drive/MyDrive/MNIST/digit_ytrain.csv'
Y_TESTING_PATH =   '/content/drive/MyDrive/MNIST/digit_ytest.csv'
```

```
%%time
# Geting the wall time required for reading the y-training csv file
y_training_all = np.loadtxt(Y_TRAINING_PATH, delimiter=',', dtype=int)
```

```
CPU times: user 271 ms, sys: 12 ms, total: 283 ms
Wall time: 324 ms
```

```
y_training_all.shape
```

```
(60000,)
```

```
%%time
y_testing = np.loadtxt(Y_TESTING_PATH, delimiter=',', dtype=int)
```

```
CPU times: user 39.6 ms, sys: 1.98 ms, total: 41.6 ms
Wall time: 44 ms
```

```
%%time

x_training_all = np.loadtxt(X_TRAINING_PATH, delimiter=',', dtype=int)
```

```
CPU times: user 35.5 s, sys: 4.48 s, total: 40 s
Wall time: 48.4 s
```

```
%%time

x_testing = np.loadtxt(X_TESTING_PATH, delimiter=',', dtype=int)
```

```
CPU times: user 4.22 s, sys: 218 ms, total: 4.43 s
Wall time: 4.44 s
```

Visualizing the Dataset

```
x_training_all.shape
```

```
(60000, 784)
```

```
x_training_all[0]
```

```
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,
```

```
y_training_all.shape
```

```
(60000,)
```

```
x_testing.shape
```

```
(10000, 784)
```

```
# First 10 labels from training dataset
y_training_all[:10]
```

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4])
```

```
# Our Features are between 0 and 255 which is a large range. So, we have to rescale our tra
# After rescaling our data is between 0 and 1
x_training_all = x_training_all / 255.0
x_testing      =  x_testing / 255.0
```

```
x_training_all[0]
```

```
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.        , 0.01176471, 0.07058824, 0.07058824,
       0.07058824, 0.49411765, 0.53333333, 0.68627451, 0.10196078,
       0.65098039, 1.        , 0.96862745, 0.49803922, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.11764706, 0.14117647, 0.36862745, 0.60392157,
       0.66666667, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.99215686, 0.88235294, 0.6745098 , 0.99215686, 0.94901961,
       0.76470588, 0.25098039, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.19215686, 0.93333333,
       0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.98431373, 0.36470588,
       0.32156863, 0.32156863, 0.21960784, 0.15294118, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.07058824, 0.85882353, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.77647059, 0.71372549,
       0.96862745, 0.94509804, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.31372549, 0.61176471, 0.41960784, 0.99215686, 0.99215686,
       0.80392157, 0.04313725, 0.        , 0.16862745, 0.60392157,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.05490196,
```

```
# Our labels are 10 in total from 0 to 9
NR_CLASSES = 10


y_training_all = np.eye(NR_CLASSES)[y_training_all]


y_training_all.shape
```

```
(60000, 10)
```

```
y_testing = np.eye(NR_CLASSES)[y_testing]
y_testing.shape
```

```
(10000, 10)
```

```
LOGGING_PATH = '/content/sample_data/MNISTtensorboard_mnist_digit_logs/'

VALIDATION_SIZE = 10000
IMAGE_WIDTH = 28
```

```
IMAGE_HEIGHT = 28
CHANNELS = 1
# Total number of features
TOTAL_INPUTS = IMAGE_WIDTH*IMAGE_HEIGHT*CHANNELS


# Now we have to divide our training dataset into smaller training and validation dataset
# Training dataset contains 50000 and avalidation contains 10000


# From start to the validation size
x_val = x_training_all[:VALIDATION_SIZE]
y_val = y_training_all[:VALIDATION_SIZE]


x_val.shape
```

```
(10000, 784)
```

```
# From validation size till the end
x_training = x_training_all[VALIDATION_SIZE:]
y_training = y_training_all[VALIDATION_SIZE:]


x_training.shape
```

```
(50000, 784)
```

```
# Creating tensors
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

# tf.placeholder is used for creating tensors 2 parameters (datatype, shape of tensor)
X = tf.placeholder(tf.float32, shape=[None, TOTAL_INPUTS], name='X')
Y = tf.placeholder(tf.float32, shape=[None, NR_CLASSES], name='labels')


# Hyperparameters
# nr_epochs = 25
# learning_rate = 1e-3
nr_epochs = 50
learning_rate = 1e-3

n_hidden1 = 512
n_hidden2 = 64


1e-3
```

```
0.001
```

```
def setup_layer(input, weight_dim, bias_dim, name):

    with tf.name_scope(name):
      """
      Iniatializing the weights
      tf.truncated_normal generates random values execpt for extreme values
      Shape = no of inputs and neurons in the layer
      Standard Deviation tells how far appart weights should from each others
      """
      initial_w = tf.truncated_normal(shape=weight_dim, stddev=0.1, seed=42)
      # Calculations for weights
      w = tf.Variable(initial_value=initial_w, name='W')

      """
      Initializing the biases
      All biases start from same value that is 0
      Shape is no of neurons in layer
      """
      initial_b = tf.constant(value=0.0, shape=bias_dim)
      # Calculation of the biases
      b = tf.Variable(initial_value=initial_b, name='B')

      """
      Input layer of next hidden layer
      MatrixMultiplication
      """
      layer_in = tf.matmul(input, w) + b

      # Checking for the the last hidden layer to apply activation accordingly
      if name=='out':
        # Apply softmax for the last layer
        layer_out = tf.nn.softmax(layer_in)
      else:
        # Apply relu for the remaining layers
        layer_out = tf.nn.relu(layer_in)

      tf.summary.histogram('weights', w)
      tf.summary.histogram('biases', b)

      return layer_out


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

layer_drop = tf.nn.dropout(layer_1, keep_prob=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, NB_CLASSES],
```

```
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}'


# Folder for Tensorboard

folder_name = f'{model_name} at {strftime("%H:%M")}'
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!
```

Defining Loss Function

```
with tf.name_scope('loss_calc'):
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=Y, logits=outpu
```

Defining Optimizer

```
with tf.name_scope('optimizer'):
    optimizer = tf.train.AdamOptimizer(learning_rate)
    train_step = optimizer.minimize(loss)
```

Accuracy Metric

```
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))


with tf.name_scope('performance'):
    tf.summary.scalar('accuracy', accuracy)
    tf.summary.scalar('cost', loss)
```

Check Input Images in Tensorboard

```
with tf.name_scope('show_image'):
    x_image = tf.reshape(X, [-1, 28, 28, 1])
    tf.summary.image('image_input', x_image, max_outputs=4)
```

Run Session

```
sess = tf.Session()
```

Setup Filewriter and Merge Summaries

```
merged_summary = tf.summary.merge_all()

train_writer = tf.summary.FileWriter(directory + '/train')
train_writer.add_graph(sess.graph)

validation_writer = tf.summary.FileWriter(directory + '/validation')
```

Initialise all the variables

```
init = tf.global_variables_initializer()
sess.run(init)
```

Batching the Data

```
size_of_batch = 1000
# size_of_batch = 500
# size_of_batch = 2000
```

```
num_examples = y_training.shape[0]
nr_iterations = int(num_examples/size_of_batch)

index_in_epoch = 0
```

this function is to go to the next batch

```
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

```
import array as arr
accuracy_array = []
```

```
%%time
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_training, labels=y_t

        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_dictio

    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)

    accuracy_array.append(batch_accuracy)
```

```
print('Done training!')
```

```
Epoch 0         | Training Accuracy = 0.8450000286102295
Epoch 1         | Training Accuracy = 0.859000027179718
Epoch 2         | Training Accuracy = 0.8659999966621399
Epoch 3         | Training Accuracy = 0.8730000257492065
Epoch 4         | Training Accuracy = 0.9739999771118164
Epoch 5         | Training Accuracy = 0.9769999980926514
Epoch 6         | Training Accuracy = 0.9810000061988831
Epoch 7         | Training Accuracy = 0.9819999933242798
Epoch 8         | Training Accuracy = 0.9819999933242798
Epoch 9         | Training Accuracy = 0.9860000014305115
Epoch 10        | Training Accuracy = 0.9850000143051147
Epoch 11        | Training Accuracy = 0.9829999804496765
Epoch 12        | Training Accuracy = 0.9829999804496765
Epoch 13        | Training Accuracy = 0.9890000224113464
Epoch 14        | Training Accuracy = 0.9900000095367432
Epoch 15        | Training Accuracy = 0.9909999966621399
Epoch 16        | Training Accuracy = 0.9879999756813049
Epoch 17        | Training Accuracy = 0.9909999966621399
Epoch 18        | Training Accuracy = 0.9890000224113464
Epoch 19        | Training Accuracy = 0.9879999756813049
Epoch 20        | Training Accuracy = 0.9900000095367432
Epoch 21        | Training Accuracy = 0.9879999756813049
Epoch 22        | Training Accuracy = 0.9909999966621399
Epoch 23        | Training Accuracy = 0.9900000095367432
Epoch 24        | Training Accuracy = 0.9919999837875366
Epoch 25        | Training Accuracy = 0.9919999837875366
Epoch 26        | Training Accuracy = 0.9909999966621399
Epoch 27        | Training Accuracy = 0.9890000224113464
Epoch 28        | Training Accuracy = 0.9909999966621399
Epoch 29        | Training Accuracy = 0.9919999837875366
Epoch 30        | Training Accuracy = 0.9919999837875366
Epoch 31        | Training Accuracy = 0.9919999837875366
Epoch 32        | Training Accuracy = 0.9919999837875366
Epoch 33        | Training Accuracy = 0.9919999837875366
Epoch 34        | Training Accuracy = 0.9919999837875366
Epoch 35        | Training Accuracy = 0.9900000095367432
Epoch 36        | Training Accuracy = 0.9909999966621399
Epoch 37        | Training Accuracy = 0.9900000095367432
Epoch 38        | Training Accuracy = 0.9909999966621399
Epoch 39        | Training Accuracy = 0.9909999966621399
Epoch 40        | Training Accuracy = 0.9919999837875366
Epoch 41        | Training Accuracy = 0.9909999966621399
Epoch 42        | Training Accuracy = 0.9919999837875366
Epoch 43        | Training Accuracy = 0.9919999837875366
Epoch 44        | Training Accuracy = 0.9919999837875366
Epoch 45        | Training Accuracy = 0.9919999837875366
Epoch 46        | Training Accuracy = 0.9919999837875366
Epoch 47        | Training Accuracy = 0.9919999837875366
Epoch 48        | Training Accuracy = 0.9919999837875366
Epoch 49        | Training Accuracy = 0.9919999837875366
Done training!
CPU times: user 3min 20s, sys: 4.59 s, total: 3min 24s
```

```
Wall time: 1min 53s
```

```
# accuracy of the whole training model
accuracy_of_model = np.mean(accuracy_array)
print("Accuracy of model = {}%".format(accuracy_of_model*100))
```

```
Accuracy of model = 97.88000583648682%
```

Make a Prediction

```
im = Image.open('/content/drive/MyDrive/MNIST/2.png')
im
```

```
img = im.resize((28,28))
```

```
bw = img.convert('L')
```

```
img_array = np.invert(bw)
```

```
img_array.shape
```

```
(28, 28)
```

```
test_img = img_array.ravel()
```

```
test_img.shape
```

```
(784,)
```

```
img_array
```

```
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,  38, 111, 160, 170,
        149,  78,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   3, 134, 199, 136,  82,  63,
         95, 180, 185,  39,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   2, 157, 174,  19,   0,   0,   0,
          0,   0, 101, 208,  20,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0, 111, 184,   6,   0,   0,   0,   0,
          0,   0,   0, 143, 142,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,  10, 206,  40,   0,   0,   0,   0,   0,
          0,   0,   0,  24, 206,  20,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,  67, 186,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 186,  61,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0, 153, 109,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 157,  96,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   3, 199,  53,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 154,  98,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   4, 154,  24,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   1, 189,  56,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  32, 204,  12,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0, 142, 139,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  73, 209,  22,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  38, 216,  55,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  14, 199,  97,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  30,
 189, 134,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  63, 211,
```

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


print(f'Prediction for test image is {prediction}')
```

```
Prediction for test image is [2]
```

Testing and Evaluation

```
test_accuracy = sess.run(fetches=accuracy, feed_dict={X:x_testing, Y:y_testing})
print(f'Accuracy on test set is {test_accuracy:0.2%}')
```

```
Accuracy on test set is 97.66%
```

```
def predict_test_image(Img_url):
  im = Image.open(Img_url)
  img = im.resize((28, 28))
  bw = img.convert('L')
  img_array = np.invert(bw)
  test_img = img_array.ravel()
  prediction = sess.run(fetches=tf.argmax(output, axis=1), feed_dict={X:[test_img]})
  print(f'Prediction for test image is {prediction}')


def predict_test_image1(Img_url):
  im = Image.open(Img_url)
  img = im.resize((28, 28))
  bw = img.convert('L')
  img_array = np.invert(bw)
  test_img = img_array.ravel()
  prediction = sess.run(fetches=tf.argmax(output, axis=1), feed_dict={X:[test_img]})
  print(f'Prediction for test image is {prediction}')
```

Prediction 1

```
img = Image.open('/content/drive/MyDrive/MNIST/1.png')
display(img)
predict test image('/content/drive/MyDrive/MNIST/1.png');
```

```
Prediction for test image is [1]
```

Prediction 2

```
img = Image.open('/content/drive/MyDrive/MNIST/2.png')
display(img)
predict_test_image('/content/drive/MyDrive/MNIST/2.png');
```

```
Prediction for test image is [2]
```

Prediction 3

```
img = Image.open('/content/drive/MyDrive/MNIST/3.png')
display(img)
predict_test_image('/content/drive/MyDrive/MNIST/3.png');
```

```
Prediction for test image is [3]
```

Prediction 4

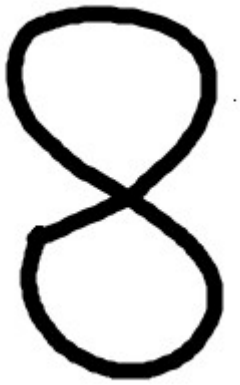
```
img = Image.open('/content/drive/MyDrive/MNIST/4.png')
display(img)
predict_test_image('/content/drive/MyDrive/MNIST/4.png');
```

```
Prediction for test image is [4]
```

```
img = Image.open('/content/drive/MyDrive/MNIST/ii_4.png')
display(img)
predict_test_image('/content/drive/MyDrive/MNIST/ii_4.png');
```

```
Prediction for test image is [4]
```

```
img = Image.open('/content/drive/MyDrive/MNIST/8.png')
display(img)
predict_test_image('/content/drive/MyDrive/MNIST/8.png');
```



```
Prediction for test image is [8]
```

```
img = Image.open('/content/drive/MyDrive/MNIST/7777.png')
display(img)
predict_test_image('/content/drive/MyDrive/MNIST/7777.png');
```

```
Prediction for test image is [7]
```

```
img = Image.open('/content/drive/MyDrive/MNIST/22222.png')
display(img)
predict_test_image('/content/drive/MyDrive/MNIST/22222.png');
```

```
Prediction for test image is [2]
```

```
img = Image.open('/content/drive/MyDrive/MNIST/555555.png')
display(img)
predict_test_image('/content/drive/MyDrive/MNIST/555555.png');
```

```
Prediction for test image is [8]
```

Webpage

```
temp = test_accuracy*100
model_accuracy = round(temp,2)
df = px.data.tips()# Build App
app = JupyterDash(__name__)
app.layout = html.Div([
    html.H1("ML SEMESTER PROJECT"),
    html.H1("DIGIT RECOGNITION"),
    html.H2("Model Accuracy: {}%".format(model_accuracy)),
    html.H2("Img:"),
    #html.Img(src = "https://drive.google.com/uc?export=view&id=1UcMUe3ifP9ijRuQtHWavWsvVnF
    html.Img(src = "https://drive.google.com/uc?export=view&id=1_L1CCgQ-x1c5FCeExEPmXoXmBpr

    html.H2("Prediction: {}".format(prediction)),
])

# app.run_server(mode='external')
app.run_server(host="127.0.0.1", port="8000")
```

```
Dash app running on:
http://127.0.0.1:8000/
```

Reset for the Next Run

```
train_writer.close()
validation_writer.close()
sess.close()
tf.reset_default_graph()
```