

## Feedforward Neural Network Implementation with Keras and TensorFlow: MNIST

### Dataset

This Jupyter notebook demonstrates the implementation of a feedforward neural network using Keras and TensorFlow. The goal of this assignment is to walk through the process of building, training, and evaluating a neural network for image classification. We will use the MNIST or CIFAR-10 dataset as per your choice.

Dataset: <https://www.tensorflow.org/datasets/catalog/mnist>

### Assignment Steps:

- Import the necessary packages:** We will start by importing the required Python libraries, including Keras, TensorFlow, and other essential packages.
- Load the training and testing data:** In this step, we will load the dataset (either MNIST or CIFAR-10) for training and testing the neural network.
- Define the network architecture using Keras:** Here, we will define the architecture of our feedforward neural network, specifying the number of layers, neurons, and activation functions.
- Train the model using SGD (Stochastic Gradient Descent):** We will use the Stochastic Gradient Descent optimizer to train the neural network on the training data.
- Evaluate the network:** After training, we will evaluate the performance of the model on the testing data to assess its accuracy and other relevant metrics.
- Plot the training loss and accuracy:** We will create plots to visualize the training loss and accuracy over epochs, providing insights into the model's learning progress.

Feel free to follow along with the code and explanations provided in this notebook to gain a better understanding of building and training neural networks for image classification.

### Build the Image classification model by dividing the model into following 4 stages:

- Loading and preprocessing the image data
- Defining the model's architecture
- Training the model
- Estimating the model's performance

Note: Both assignment can use this code accordingly

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.datasets import mnist
```

Loading MNIST data from keras .dataset

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

```
#shape of numpy arrays
print(f"Shape of X_train {x_train.shape}")
print(f"Shape of y_train {y_train.shape}")
print(f"Shape of x_test {x_test.shape}")
print(f"Shape of y_test {y_test.shape}")
# x_train( 60k images of 28*28 dimentions) Pixel values range from 0 to 255.
# y_train: digit labels (integers in range 0-9)

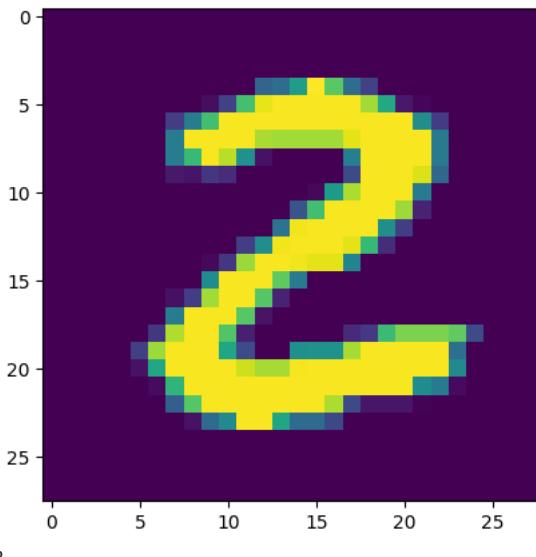
Shape of X_train (60000, 28, 28)
Shape of y_train (60000,)
Shape of x_test (10000, 28, 28)
Shape of y_test (10000,)
```

Image Dimension -> 28 X 28

Grayscale image -> 1 channel

```
#displaying the image
plt.imshow(x_train[25])
plt.show()
```

```
#printing corresponding label
print(y_train[25])
```



2

### Image Labels

```
#unique values in Y_train
print(np.unique(y_train))
print(np.unique(y_test))
```

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

All the images have the same dimension in this dataset, if not then convert it into same dimension to provide it to neural network

```
#scaling the values
x_train = x_train/255
x_test = x_test/255
# 0/255 = 255
#255/255 = 1 maximum value
```

```
print(x_train.shape, x_test.shape)
(60000, 28, 28) (10000, 28, 28)
```

```
# print(x_train[10])
# plt.imshow(x_train[10])
```

## ▼ Bulding Neural Network

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(50,activation='relu',name='L1'),
    keras.layers.Dense(50,activation='relu',name='L2'),
    keras.layers.Dense(10,activation='softmax',name='L3')
])
```

```
C:\Users\Parth\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\reshaping\flatten.py:37: UserWarning
super().__init__(**kwargs)
```

```
model.compile(optimizer="sgd",loss=tf.keras.losses.SparseCategoricalCrossentropy(),
metrics=['accuracy'])
```

## ▼ train neural network

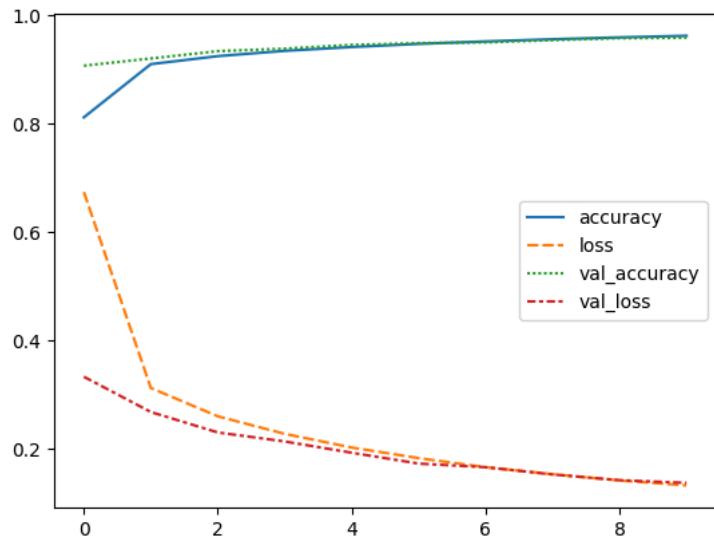
```
# Training the model
history = model.fit(x_train, y_train,
batch_size=30,
```

```
epochs=10,
validation_data=(x_test, y_test),
shuffle=True)
```

```
Epoch 1/10
2000/2000 10s 2ms/step - accuracy: 0.6683 - loss: 1.1263 - val_accuracy: 0.9072 - val_loss: 0.3335
Epoch 2/10
2000/2000 4s 2ms/step - accuracy: 0.9075 - loss: 0.3288 - val_accuracy: 0.9205 - val_loss: 0.2685
Epoch 3/10
2000/2000 4s 2ms/step - accuracy: 0.9235 - loss: 0.2674 - val_accuracy: 0.9340 - val_loss: 0.2308
Epoch 4/10
2000/2000 4s 2ms/step - accuracy: 0.9340 - loss: 0.2345 - val_accuracy: 0.9386 - val_loss: 0.2141
Epoch 5/10
2000/2000 4s 2ms/step - accuracy: 0.9388 - loss: 0.2105 - val_accuracy: 0.9456 - val_loss: 0.1933
Epoch 6/10
2000/2000 4s 2ms/step - accuracy: 0.9467 - loss: 0.1856 - val_accuracy: 0.9487 - val_loss: 0.1733
Epoch 7/10
2000/2000 4s 2ms/step - accuracy: 0.9519 - loss: 0.1682 - val_accuracy: 0.9503 - val_loss: 0.1666
Epoch 8/10
2000/2000 4s 2ms/step - accuracy: 0.9551 - loss: 0.1538 - val_accuracy: 0.9543 - val_loss: 0.1533
Epoch 9/10
2000/2000 4s 2ms/step - accuracy: 0.9588 - loss: 0.1463 - val_accuracy: 0.9580 - val_loss: 0.1427
Epoch 10/10
2000/2000 4s 2ms/step - accuracy: 0.9627 - loss: 0.1326 - val_accuracy: 0.9587 - val_loss: 0.1377
```

```
import seaborn as sns
sns.lineplot(model.history.history)
```

<Axes: >



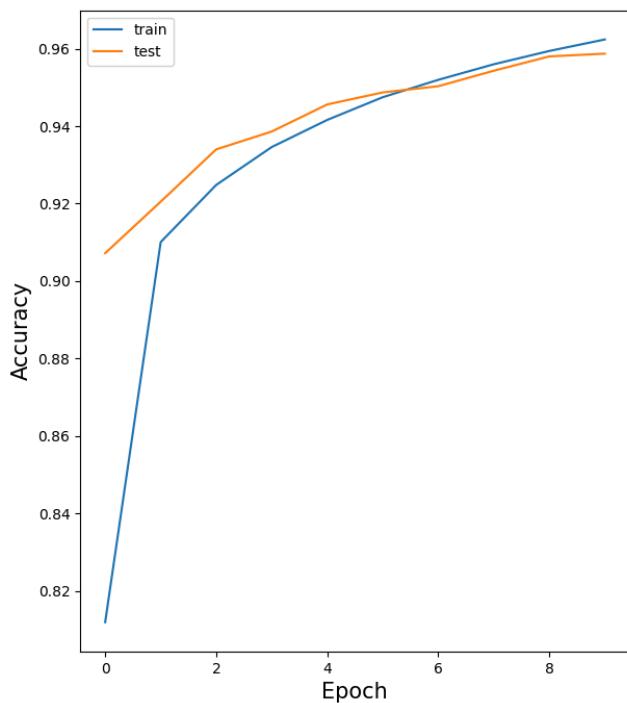
## ▼ Alternative Plot below for more details

```
# Plotting the Model Accuracy & Model Loss vs Epochs
plt.figure(figsize=[15,8])

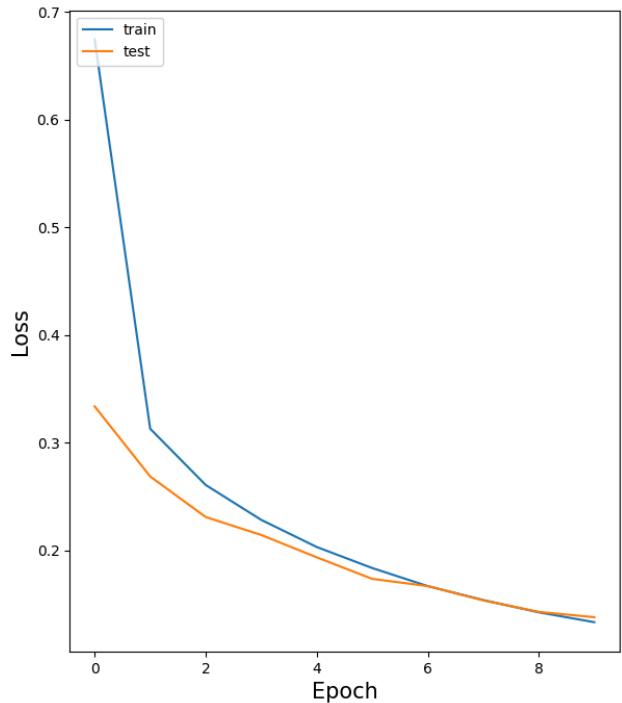
# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')

# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

## Model Accuracy



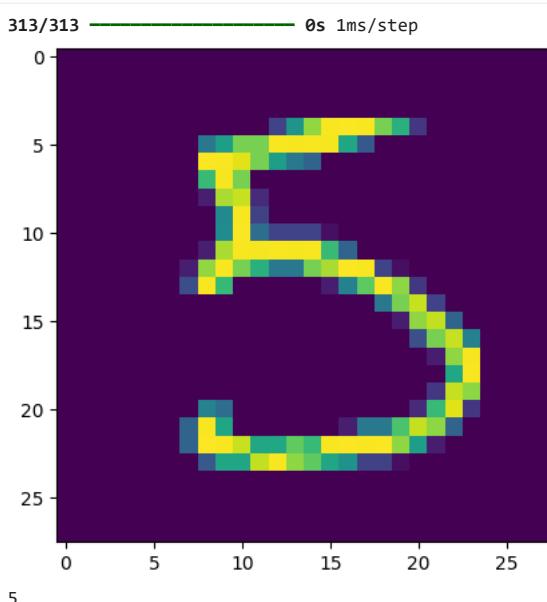
## Model Loss



## Evaluation

```
loss,accuracy = model.evaluate(x_test,y_test)  
313/313 ━━━━━━ 1s 2ms/step - accuracy: 0.9520 - loss: 0.1627
```

```
predicted_value=model.predict(x_test)  
plt.imshow(x_test[15])  
plt.show()  
print(np.argmax(predicted_value[15], axis=0))
```



Start coding or [generate](#) with AI.

