NAME: KSHITIJ VINOD SALI
CLASS: BE-A ROLL NO:
SUBJECT: LP-IV: DEEP LEARNING

LAB ASSIGNMENT - 02

Title: Feedforward Neural Network for Image Classification.

Course Outcome: CO2 - Build and train a deep neural network models for use in various applications.

Date of Completion:
Assessment Grade/ Marks:
Assessor's Sign with Date:

Problem Statement: Implementing Feedforward neural network with keras and TensorFlow.

a) Import the necessary packages.
b) Load the training and testing data (MNIST/ CIFAR10)
c) Define the network architecture using Keras
d) Train the model using SGD.
e) Evaluate the network.
f) Plot the training loss and accuracy.

Blooms Taxanomy Catagory: Applying, Analyzing.

Requirements: Python, TensDrflow, Keras API,

python libraries (Pandas, Numpy, Matplotlib), MNIST dataset.

Theory:

A Feedforward Neural Network is a fundamental type of artificial neural network where connections between nodes don't form a cycle. For image classification, the network is designed to take 28×28 pixels image as input and output a prediction of which digit (0-9) image representation.

The conceptual architecture of network is depicted in figure. It consists of 3-main parts: input mechanism, hidden layer & output layer.

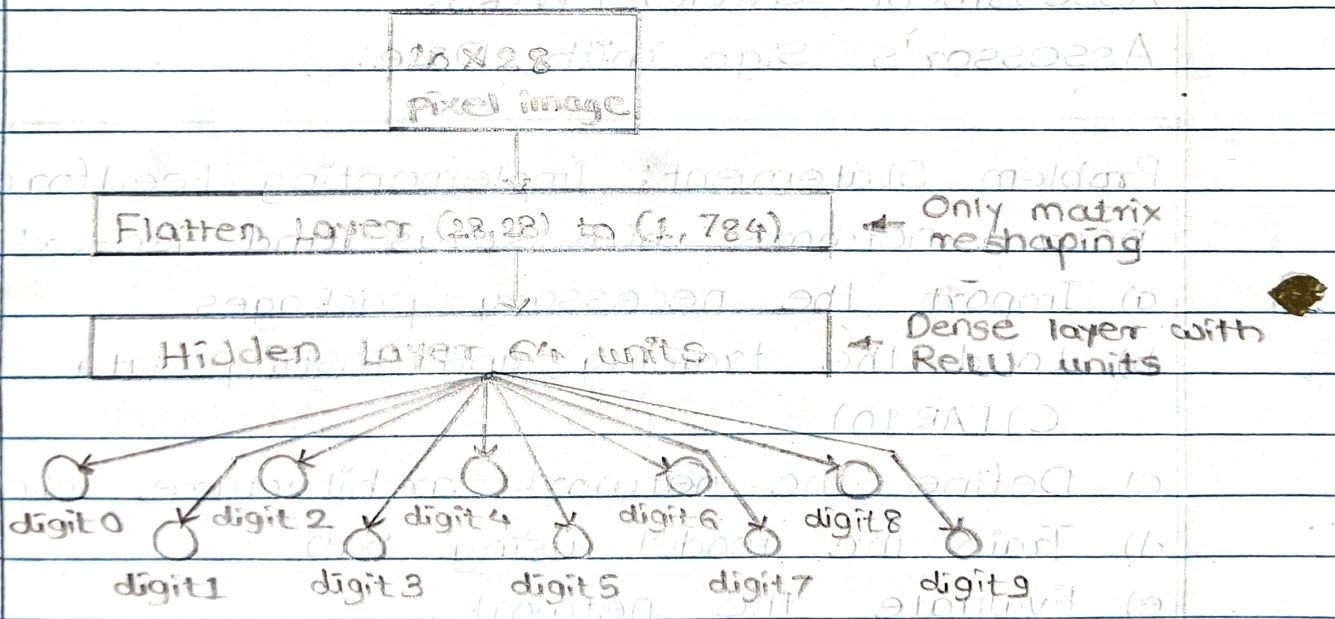

Fig: Conceptual Diagram of Neural Network

i) Flatten layer: The input to network is a 2D image of size 28×28 pixels. The first step is to transform 2D matrix into 1D vector.

The Flatten layer accomplish by unrolling the matrix into single vector of size 784, without altering pixels.

ii) Dense Layer: Each neuron in Dense layer receives input from all neurons in previous layer. It computes a weighted sum of inputs, adds a bias & then passes result through activation function. In this model, two dense layers are used:

a) Hidden Layer - Uses ReLU activation function, $f(x) = \max(0, x)$, which outputs the input, if +ve & zero otherwise.

b) Output Layer - Uses softmax function, convert raw scores into a probability distribution.

Algorithm & Steps:

The implementation follows a structured, six step process:

i) Import Necessary Packages: The first step is to import all required libraries.

Code -

```
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pylot as plt.
```

ii) Load and Prepare the Data - The MNIST dataset is loaded using the Keras API.

The pixel values of the images, which range from 0 to 255, are normalized to a range of 0 to 1 by dividing by 255. This normalization helps the training process converge more efficiently.

Code -

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
y. x_test = x_test / 255.0
```

iii) Define the Network Architecture: The models is define using the keras. Sequential API, which allows for the simple stacking of layers. The architecture consists of Flatten layer to process input shape, Dense hidden layer with 128 neurons & ReLU activation, & a Dense output layer with 10 neurons & softmax activation.

iv) Compile and Train the Model: Before training, model is compiled. This step configures learning process by specifying an optimizer, a loss function & metrics.

a) Optimizer - Stochastic Gradient Descent (sgd) is used to update network's weights.

b) Loss Function - sparse_categorical_crossentropy is chosen. This specific loss function is used because lables are provided as

integers rather than one-hot encoded vectors. This choise streamlines data preperation process by avoiding explicit one-hot encoding step, demonstrating an efficient use of keras API.

a) Metrics - Accuracy is monitored during training.

Code -

```
model.compile (optimizer = 'sgd',
               loss = 'sparse_categorical_crossentropy',
               metrics = ['accuracy'])
history = model.fit (x_train, y_train, validation_
data = (x_test, y_test), epochs = 10)
```

v) Evaluate the Network: After training, the model's final performance is evaluated on test dataset, which it has not seen during training. This provides an unbiased estimate of model's generalization ability

Code -

```
test_loss, test_acc = model.evaluate (x_test,
y_test)
print ("Loss = %.3f" % test_loss)
print ("Accuracy = %.3f" % test_acc)
# Expected Output: Loss = 0.162, Accuracy = 0.952
  (or similar)
```

vi) Plot Training Loss and Accuracy: The history object returned by model.fit() contains loss & accuracy values for each epoch. These are plotted to visualize the

learning process. The ideal plot shows both training & validation accuracy increasing & converging, while both loss values decrease & converge.

Inference: With above code we can see that, throughout the epochs, our model accuracy increases & loss decreases that is good since our model gains confidence with our predictions.