AD18511 – DEEP LEARNING LABORATORY

DATE:

EX.NO: 5 MULTI LAYER PERCEPTRON USING IMAGE CLASSIFICATION.

AIM:

To write a program to build a multi layer perceptron model using tensor flow.

DESCRIPTION:

- Multi layer perceptron (MLP) is a supplement of feed forward neural network.
- It consists of three types of layers—the input layer, output layer and hidden layer .
- The input layer receives the input signal to be processed. The required task such as prediction and classification is performed by the output layer.
- An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP.
- Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm.
- The computations taking place at every neuron in the output and hidden layer are as follows,

$$ox=Gb2+W2hx$$

$$hx=\Phi x=sb1+W1x$$

- with bias vectors b(1), b(2); weight matrices W(1), W(2) and activation functions G and s. The set of parameters to learn is the set $\theta = \{W(1), b(1), W(2), b(2)\}$.

PROGRAM:

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense,Flatten
from keras.utils import to_categorical
import matplotlib.pyplot as plt

#load and preprocess the mnist dataset
(x_train,y_train),(x_test,y_test)=mnist.load_data()
x_train=x_train.reshape((-1,28*28))/255.0
x_test=x_test.reshape((-1,28*28))/255.0
y_train=to_categorical(y_train,num_classes=10)
y_test=to_categorical(y_test,num_classes=10)

#build the MLP model
model=Sequential()
model.add(Dense(128,activation='relu',input_shape=(28*28,)))
model.add(Dense(64,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])

#train the model
history=model.fit(x_train,y_train,epochs=50,batch_size=64,validation_split=0.2)
```

OUTPUT:

```
Epoch 1/50
750/750 [==============================] - 1s 1ms/step - loss: 0.3140 - accuracy: 0.9103 - val_loss: 0.1590 -
val_accuracy: 0.9539
Epoch 2/50
750/750 [==============================] - 1s 888us/step - loss: 0.1314 - accuracy: 0.9614 - val_loss: 0.1173 -
val_accuracy: 0.9647
Epoch 3/50
750/750 [==============================] - 1s 904us/step - loss: 0.0881 - accuracy: 0.9736 - val_loss: 0.1034 -
val_accuracy: 0.9696
Epoch 4/50
750/750 [==============================] - 1s 900us/step - loss: 0.0669 - accuracy: 0.9794 - val_loss: 0.0947 -
val_accuracy: 0.9714
Epoch 5/50
750/750 [==============================] - 1s 890us/step - loss: 0.0534 - accuracy: 0.9829 - val_loss: 0.0909 -
val_accuracy: 0.9743
……
Epoch 48/50
750/750 [==============================] - 1s 900us/step - loss: 0.0049 - accuracy: 0.9987 - val_loss: 0.2094 -
val_accuracy: 0.9752
Epoch 49/50
750/750 [==============================] - 1s 914us/step - loss: 0.0053 - accuracy: 0.9983 - val_loss: 0.1832 -
val_accuracy: 0.9762
Epoch 50/50
750/750 [==============================] - 1s 906us/step - loss: 0.0020 - accuracy: 0.9994 - val_loss: 0.2035 -
val_accuracy: 0.9755
```

```
# visualize the training progress
plt.figure(figsize=(12,4))
```

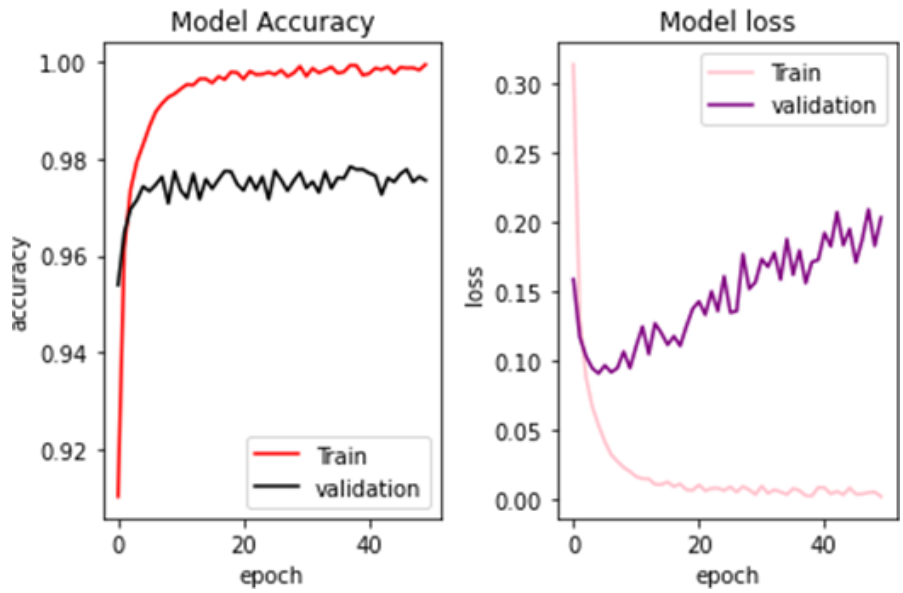
OUTPUT:

```
<Figure size 864x288 with 0 Axes>
<Figure size 864x288 with 0 Axes>
```

```
#plot training and validation accuracy values
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'],label='Train',color='r')
plt.plot(history.history['val_accuracy'],label='validation',color='black')
plt.title('Model Accuracy')
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.legend()

#plot training and validation loss values
plt.subplot(1,2,2)
plt.plot(history.history['loss'],label="Train",color='pink')
plt.plot(history.history['val_loss'],label='validation',color='purple')
plt.title('Model loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.tight_layout()
plt.show()
```

OUTPUT:




```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
import random

#generate random indices for example predictions
example_indices=random.sample(range(len(x_test)),5)

#get the predicted labels for the example prediction
example_predictions=model.predict(x_test[example_indices])
example_predicted_labels=np.argmax(example_predictions,axis=1)
```

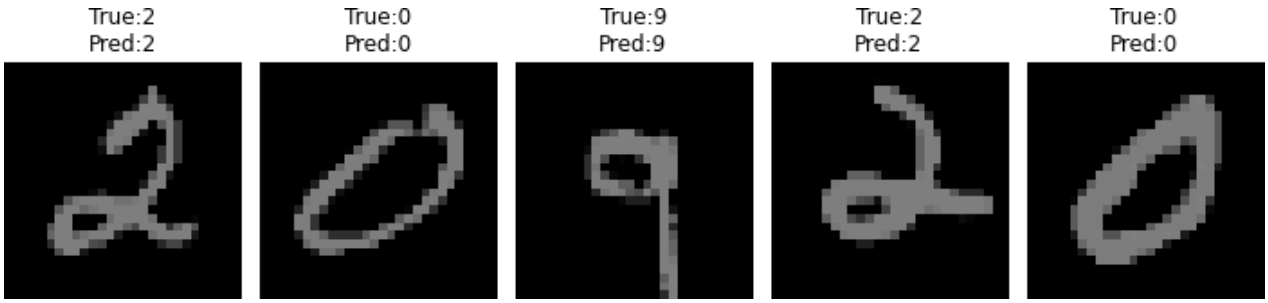
OUTPUT:

```
1/1 [==============================] - 0s 11ms/step
```

```
# get the true labels for the example predictions
example_true_labels=np.argmax(y_test[example_indices],axis=1)
```

```
# plot the example predictions with images
    plt.figure(figsize=(10,6))
    for i,index in enumerate(example_indices):
        plt.subplot(2,5,i+1)
        plt.imshow(x_test[index].reshape(28,28),cmap='gray')
        plt.title(f"True:{example_true_labels[i]}\nPred:{example_predicted_labels[i]}")
        plt.axis('off')
    plt.tight_layout()
```

OUTPUT:



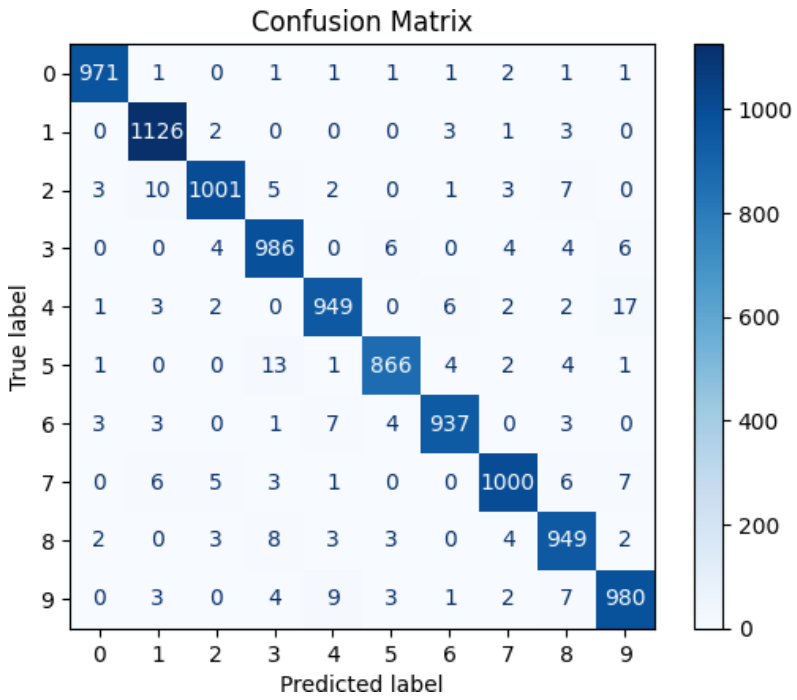

```
#compute confusion matrix
test_predictions=model.predict(x_test)
test_predicted_labels=np.argmax(test_predictions,axis=1)
conf_matrix=confusion_matrix(np.argmax(y_test,axis=1),test_predicted_labels)
```

OUTPUT:

```
313/313 [==============================] - 0s 454us/step
```

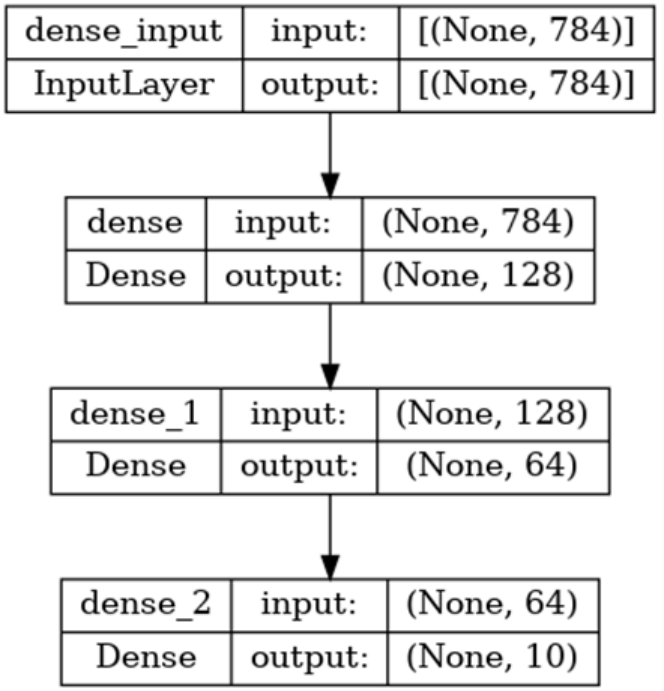
```
#display confusion matrix
plt.figure(figsize=(8,8))
ConfusionMatrixDisplay(conf_matrix,display_labels=range(10)).plot(cmap=plt.cm.Blues)
plt.title('confusion matrix')
plt.show()
```

OUTPUT:



```
from keras.utils.vis_utils import plot_model
from keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```

OUTPUT:



RESULT:

The Perceptron model is implemented using Tensorflow. The model is trained and tested and then the loss and accuracy are displayed**.**