Ashish Kumar B511002 BE-A

Assignment- 2. Implementing Feedforward neural networks 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

In [1]:
```
# a Importing Necessary packages
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import random
```

In [2]:
```
# b Load the training and testing data (MNIST)
mnist = tf.keras.datasets.mnist
```

In [3]:
```
# splitting it into training and testing data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [4]:
```
# Normalising or scalling data
x_train = x_train / 255
x_test = x_test / 255
```

In [5]:
```
# c Define the network architecture using keras

model = keras.Sequential([
keras.layers.Flatten(input_shape=(28,28)),
keras.layers.Dense(128, activation = 'relu'),
keras.layers.Dense(10, activation = 'softmax')
])

model.summary()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\layers\reshaping\flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 10)	1,290

Total params: 101,770 (397.54 KB)

Trainable params: 101,770 (397.54 KB)

Non-trainable params: 0 (0.00 B)

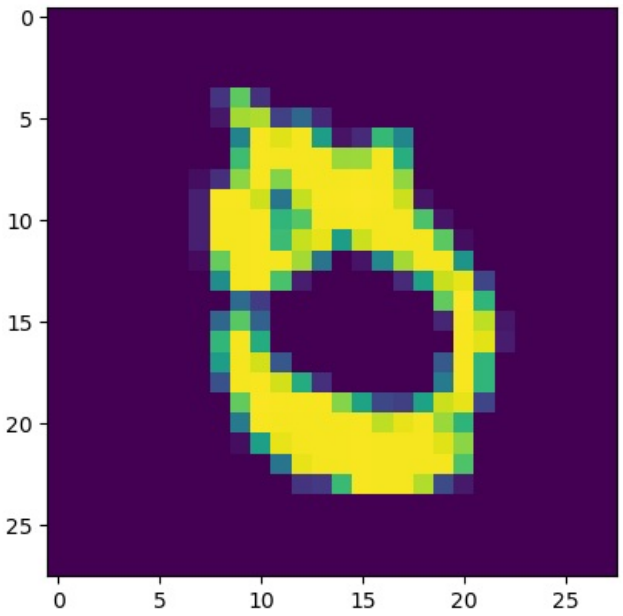
In [6]:
```
# d train the model using SGD

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)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 753us/step - accuracy: 0.7428 - loss: 0.9963 - val_accuracy: 0.9036 - val_loss
: 0.3535
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 679us/step - accuracy: 0.9029 - loss: 0.3501 - val_accuracy: 0.9198 - val_loss
: 0.2910
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 710us/step - accuracy: 0.9172 - loss: 0.2962 - val_accuracy: 0.9283 - val_loss
: 0.2562
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 691us/step - accuracy: 0.9268 - loss: 0.2593 - val_accuracy: 0.9362 - val_loss
: 0.2334
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 699us/step - accuracy: 0.9353 - loss: 0.2343 - val_accuracy: 0.9375 - val_loss
: 0.2168
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 689us/step - accuracy: 0.9397 - loss: 0.2158 - val_accuracy: 0.9406 - val_loss
: 0.2034
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 679us/step - accuracy: 0.9430 - loss: 0.2009 - val_accuracy: 0.9452 - val_loss
: 0.1904
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 659us/step - accuracy: 0.9474 - loss: 0.1870 - val_accuracy: 0.9489 - val_loss
: 0.1795
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 654us/step - accuracy: 0.9506 - loss: 0.1739 - val_accuracy: 0.9504 - val_loss
: 0.1719
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 646us/step - accuracy: 0.9530 - loss: 0.1671 - val_accuracy: 0.9536 - val_loss
: 0.1608
```

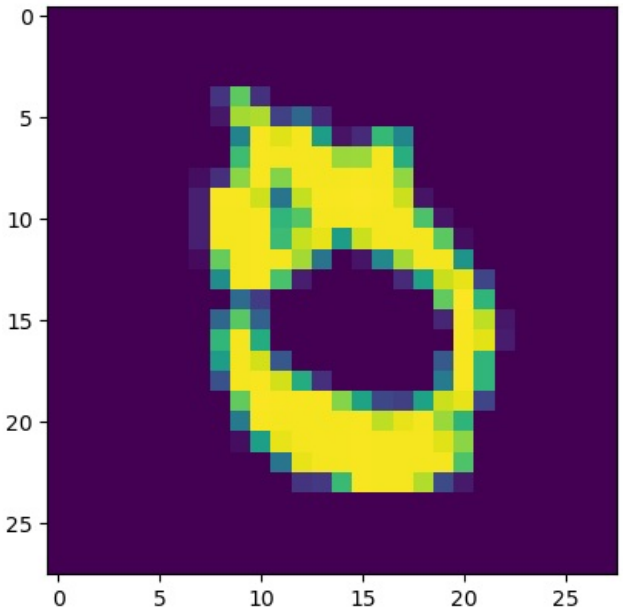
In [7]:
```
# e Evaluuate the network

test_loss, test_acc = model.evaluate(x_test, y_test)
print('loss=%.3f' %test_loss)
print('Accuracy=%.3f' %test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 478us/step - accuracy: 0.9444 - loss: 0.1886
loss=0.161
Accuracy=0.954
```

In [8]:
```
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
predicted_value=model.predict(x_test)
plt.imshow(x_test[n])
plt.show()
```




```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 564us/step
```



In [9]:
```
print('predicted value: ', predicted_value[n])
```

```
predicted value:  [6.16425514e-01 1.77973721e-04 2.15970371e-02 6.11990178e-03
 1.03721432e-05 2.89005190e-01 9.04849358e-03 3.96673642e-02
 1.42583195e-02 3.68982647e-03]
```

In [10]:
```
# f plot the training loss and accuracy

# plotting the training accuracy

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# plotting the training loss

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```