In [1]:
```
'''
Problem Statement -- Implementing Feedforward neural networks with Keras and T
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 [7]:
```
# a. importing packages
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import random
```

In [8]:
```
# b. LOAD THE TRAINING AND TESTING DATA (MNIST)
mnist = tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()

x_train = x_train/255
x_test = x_test/255
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

In [9]:
```
# 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()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:3
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen 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)
```

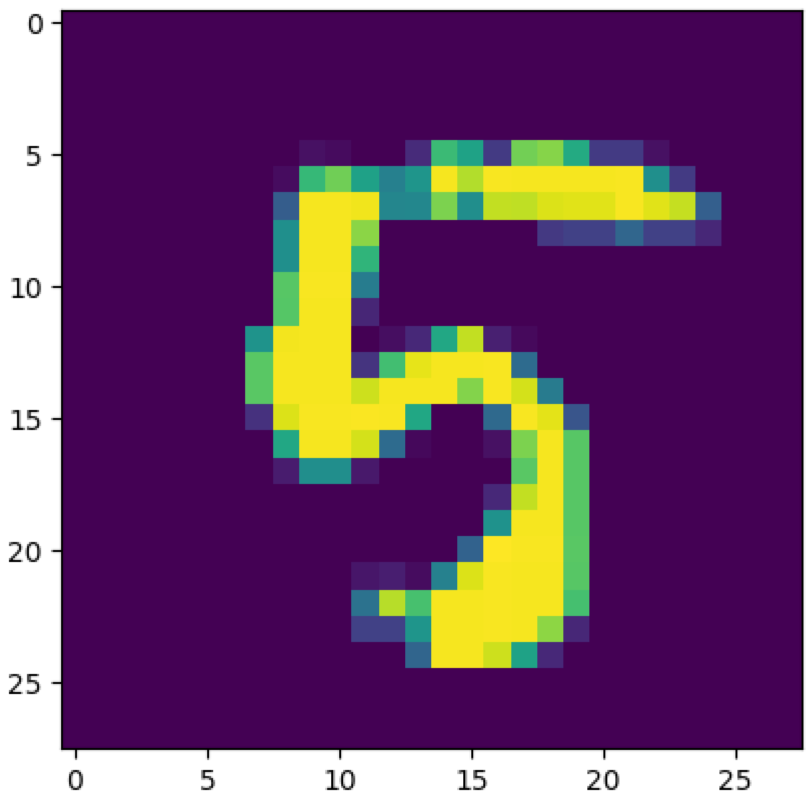
```
Non-trainable params: 0 (0.00 B)
```

```
In [10]: # 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=3)
```

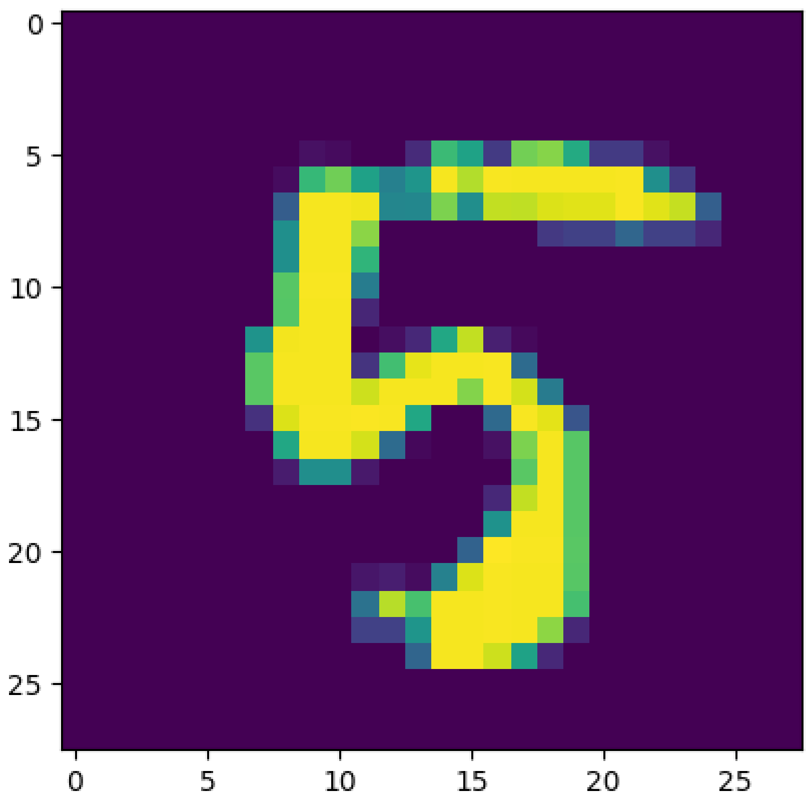
```
Epoch 1/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7399 - loss: 1.0029 -
val_accuracy: 0.9069 - val_loss: 0.3518
Epoch 2/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9053 - loss: 0.3476 -
val_accuracy: 0.9197 - val_loss: 0.2899
Epoch 3/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9189 - loss: 0.2963 -
val_accuracy: 0.9298 - val_loss: 0.2586
```

```
In [11]: # e. EVALUATE THE NETWORK
         test_loss,test_acc = model.evaluate(x_test,y_test)
         print("Loss=%.3f" %test_loss)
         print("Accuracy=%.3f" %test_acc)

         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()

         print("Predicted Value:",predicted_value[n])
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9193 - loss: 0.2955
Loss=0.259
Accuracy=0.930
```

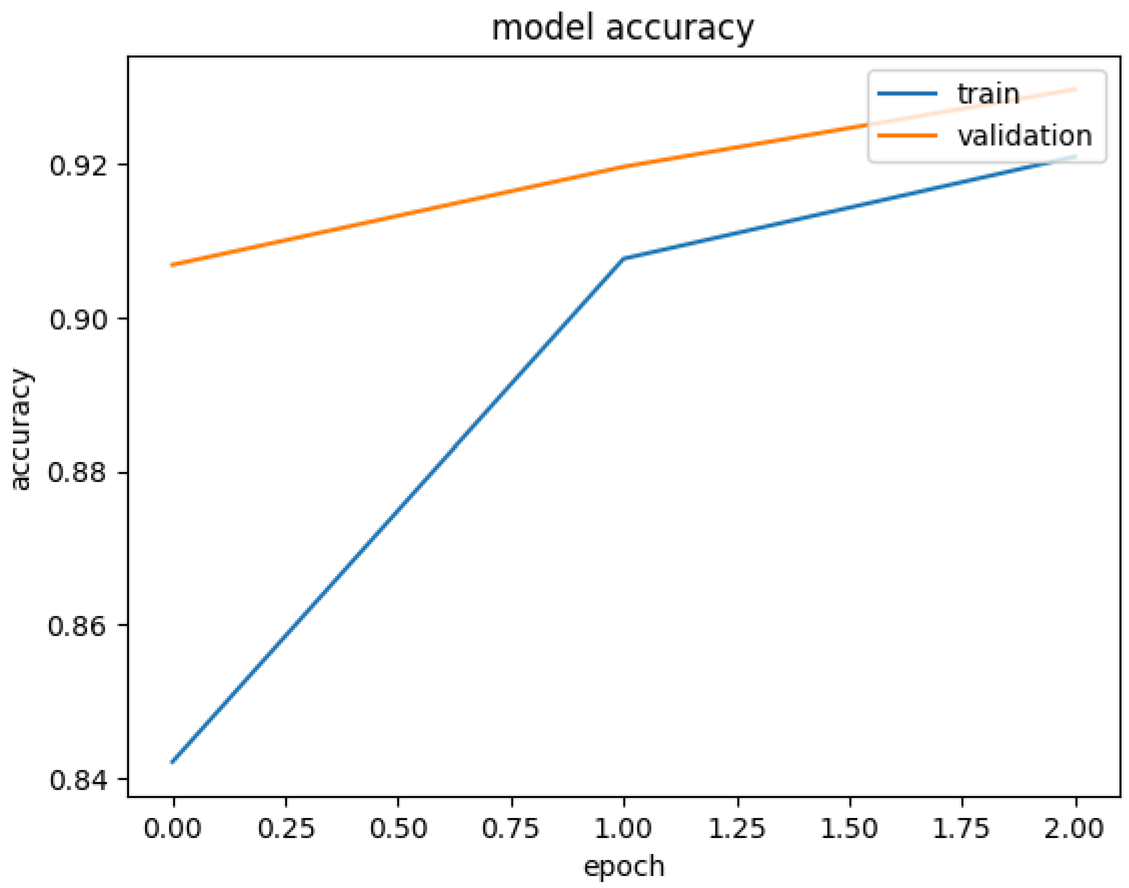


```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
```

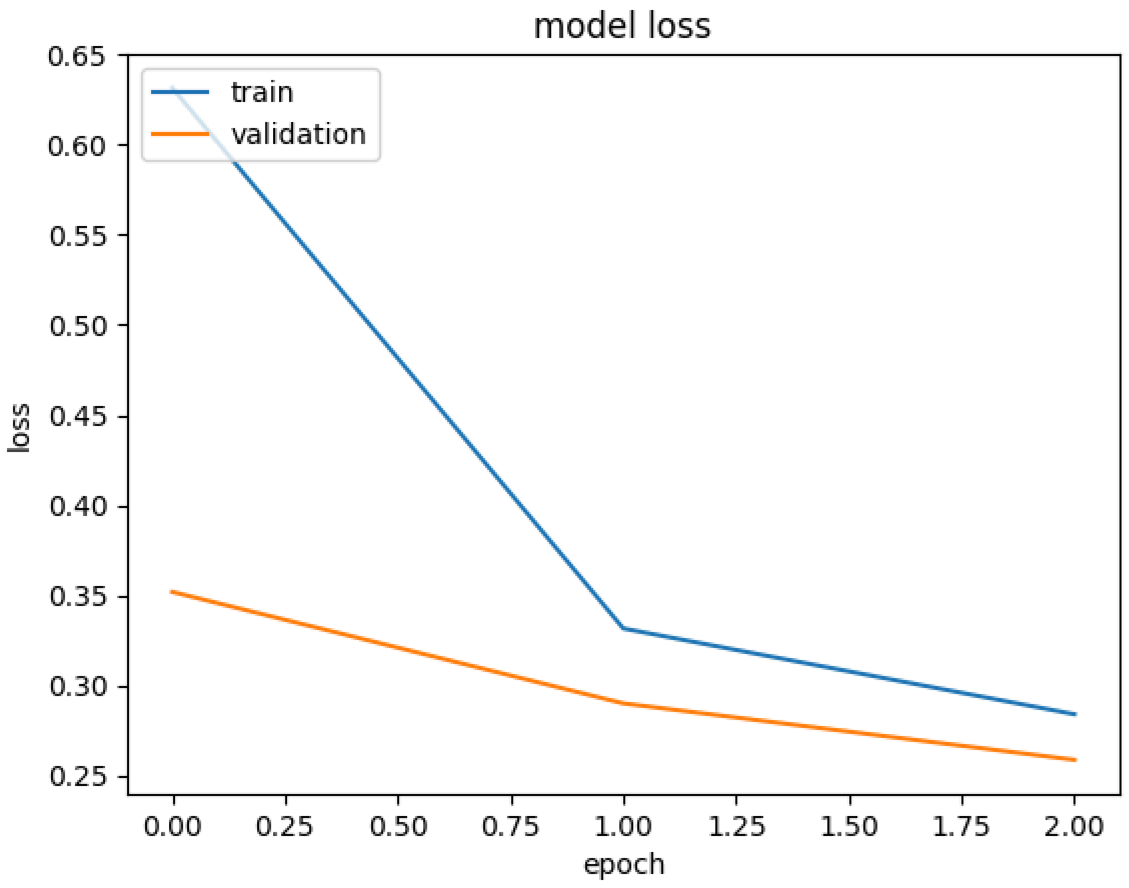


```
Predicted Value: [5.0470990e-04 3.6435162e-07 5.7552425e-05 2.9780556e-04 1.989
0675e-02
 9.0380585e-01 2.3467431e-03 9.1786347e-07 7.1932137e-02 1.1632276e-03]
```

In [12]:
```
# 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 right')
plt.show()
```



In [14]:
```
# 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 left")
plt.show()
```


model loss
loss
epoch
train
validation
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

```
In [ ]:
```

```
In [ ]:
```