Name: Subodh Chaudhari

Roll no: 13

Batch: BE IT B1

Importing necessary libraries

In [1]:
```
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
%matplotlib inline
```

Load the training and testing data (MNIST)

In [2]:
```
#importing dataset and splitting into train and test data
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [3]:
```
#to se length of traning dataset
len(x_train)
```

Out[3]:
```
60000
```

In [4]:
```
#to see the lengh of testing data
len(x_test)
```

Out[4]:
```
10000
```

In [5]:
```
x_train.shape
```

Out[5]:
```
(60000, 28, 28)
```

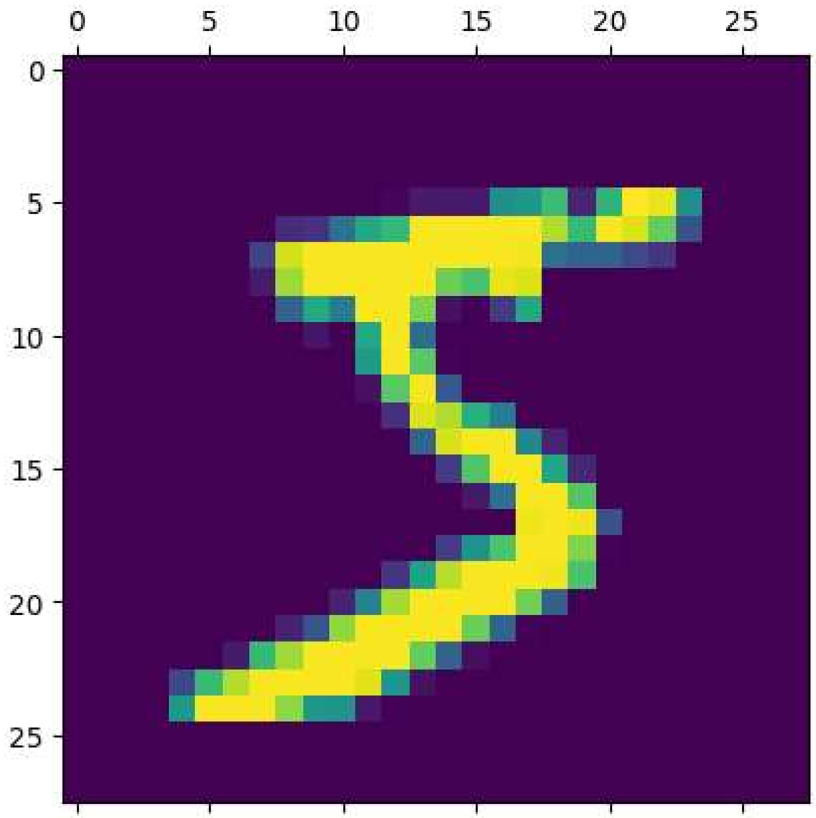
In [6]:
```
#we want to see first image

x_train[0]

#It is showing image of matrix of size 28*28 pixels(Total 784 features)
#each feature represents the intensity between 0 to 255
```

```
Out[6]: array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253,
        190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
          0,   0],
```

```
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
        148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
        253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
        253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
        195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
         11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

In [7]:
```
#to see how first image look
plt.matshow(x_train[0])
```

Out[7]: <matplotlib.image.AxesImage at 0x258deab2590>



In [8]:
```
#normalize the images by scaling pixel intensities to the range 0,1
#Normalization is a technique for organizing data in a database.

x_train = x_train / 255
x_test = x_test / 255

#here 255 is maximum value of intensity that's why it is divided by 255
```

In [9]:

```
x_train[0]
```

```
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ]])
```

Creating the model

The ReLU function is one of the most popular activation functions. It stands for “rectified linear unit”. Mathematically this function is defined as: y = max(0,x)The ReLU function returns “0” if the input is negative and is linear if the input is positive.

The softmax function is another activation function. It changes input values into values that reach from 0 to 1.

In [10]:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),   #Input layer
    keras.layers.Dense(128, activation='relu'),   #hidden layer abs
    keras.layers.Dense(10, activation='softmax')  #output layer
])
```

In [11]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 784)               0         
                                                                 
 dense (Dense)               (None, 128)               100480    
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compile the model

In [12]:
```
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Train the model

In [13]:
```
history=model.fit(x_train, y_train,validation_data=(x_test,y_test),epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 0.6334 - acc
uracy: 0.8361 - val_loss: 0.3499 - val_accuracy: 0.9045
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3318 - acc
uracy: 0.9074 - val_loss: 0.2875 - val_accuracy: 0.9201
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2849 - acc
uracy: 0.9192 - val_loss: 0.2588 - val_accuracy: 0.9271
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2563 - acc
uracy: 0.9279 - val_loss: 0.2367 - val_accuracy: 0.9326
Epoch 5/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.2353 - acc
uracy: 0.9342 - val_loss: 0.2220 - val_accuracy: 0.9359
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2182 - acc
uracy: 0.9385 - val_loss: 0.2069 - val_accuracy: 0.9406
Epoch 7/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2037 - acc
uracy: 0.9423 - val_loss: 0.1959 - val_accuracy: 0.9434
Epoch 8/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1906 - acc
uracy: 0.9464 - val_loss: 0.1836 - val_accuracy: 0.9488
Epoch 9/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1791 - acc
uracy: 0.9497 - val_loss: 0.1773 - val_accuracy: 0.9500
Epoch 10/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1689 - acc
uracy: 0.9525 - val_loss: 0.1687 - val_accuracy: 0.9514
```

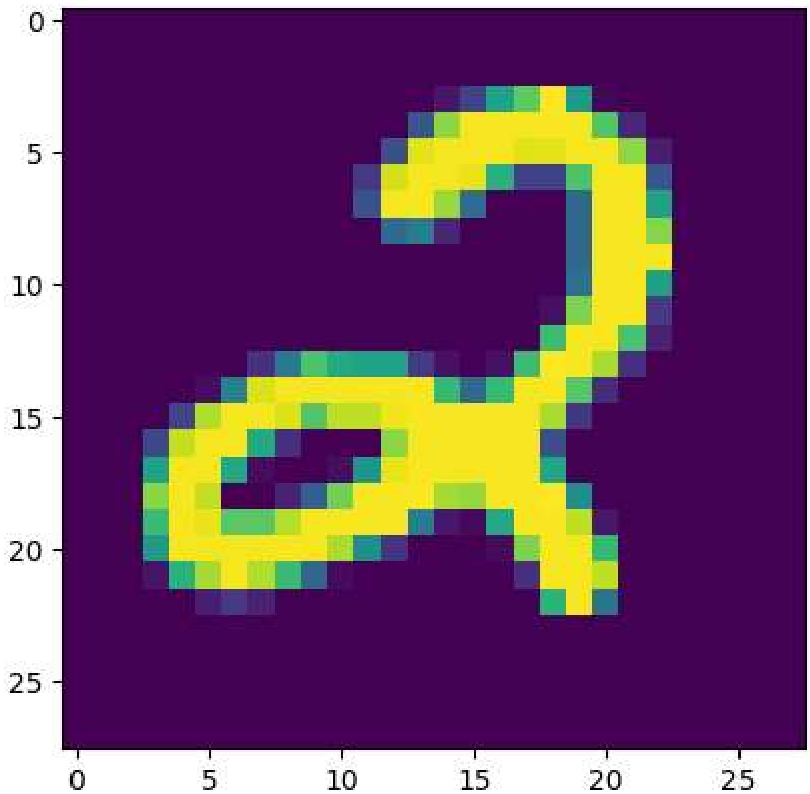
Evaluate the model

In [14]:
```
test_loss,test_acc=model.evaluate(x_test,y_test)
print("Loss=%.3f" %test_loss)
print("Accuracy=%.3f" %test_acc)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1687 - accur
acy: 0.9514
Loss=0.169
Accuracy=0.951
```

Making Prediction on New Data

In [15]:
```
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
```



In [16]:
```
#we use predict() on new data
predicted_value=model.predict(x_test)
print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]))
```

```
313/313 [==============================] - 1s 3ms/step
Handwritten number in the image is= 2
```

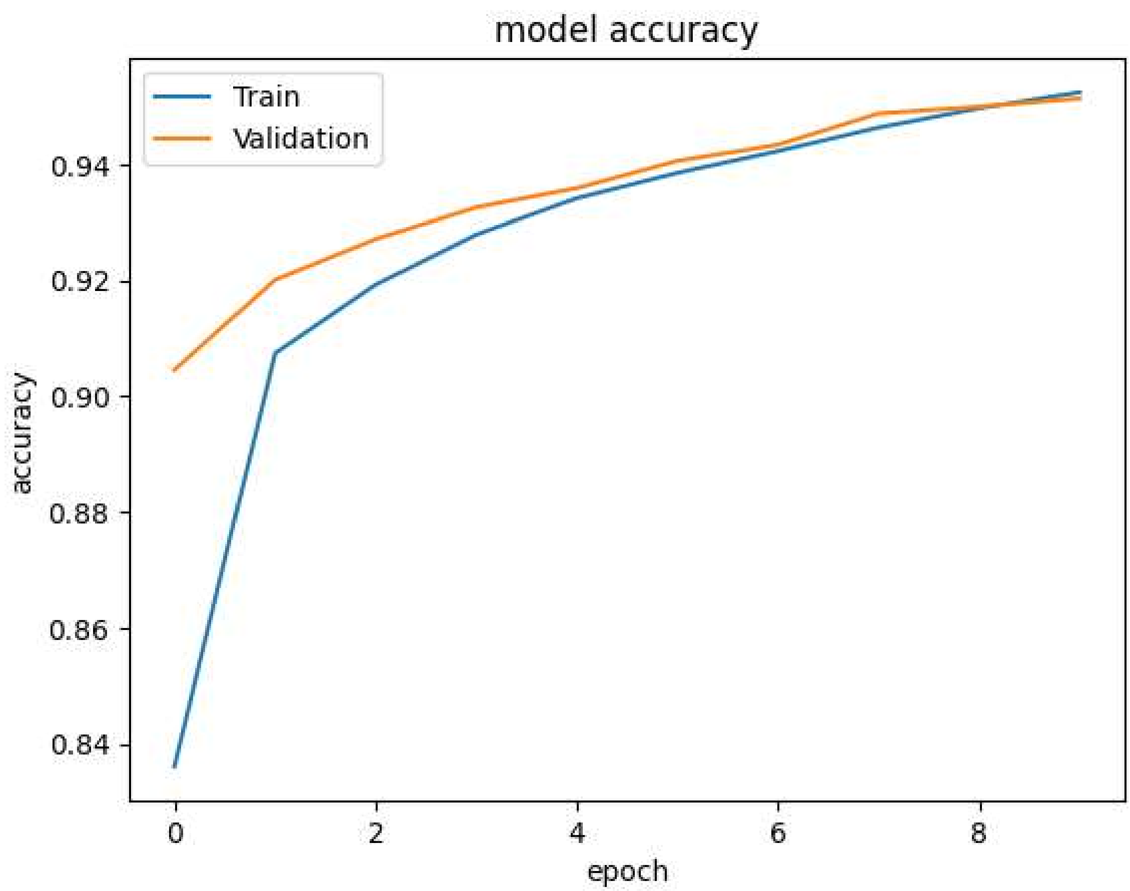
Plot graph for Accuracy and Loss

In [17]:
```
history.history??
```

In [18]:
```
history.history.keys()
```

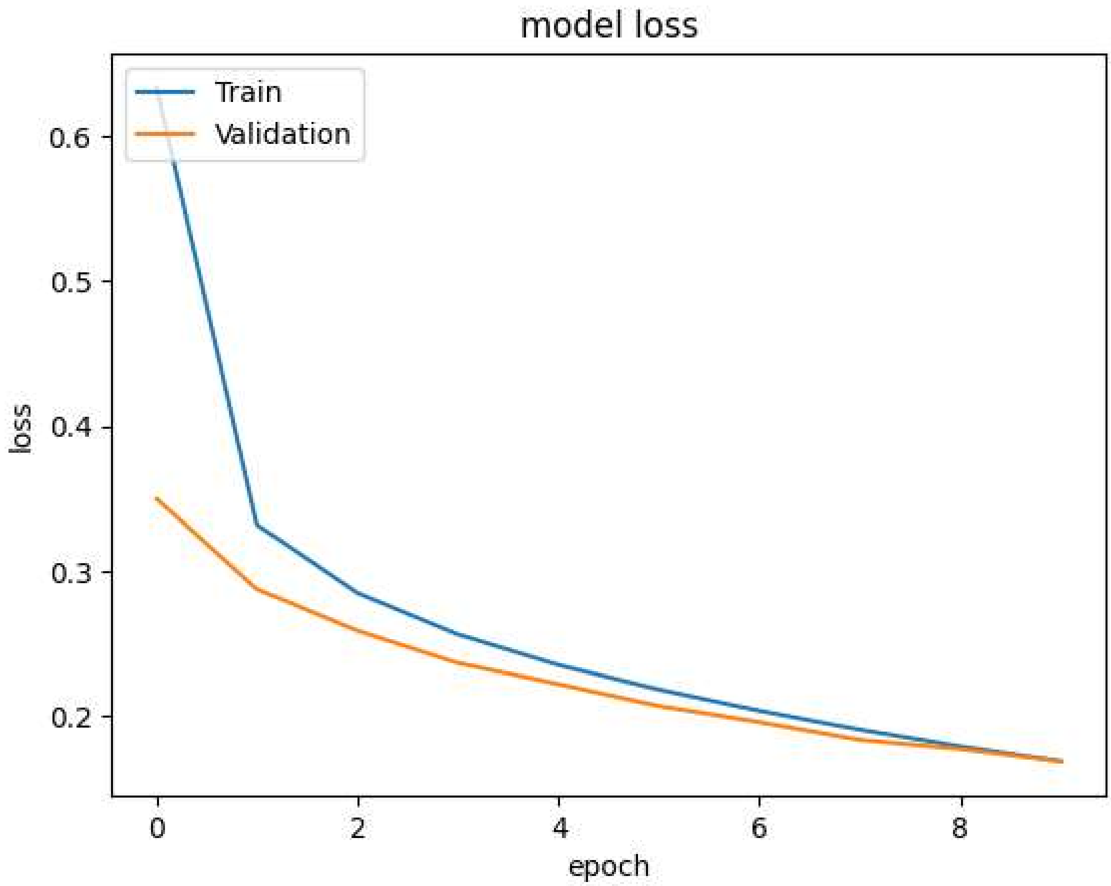
Out[18]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [19]:
```
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()
```



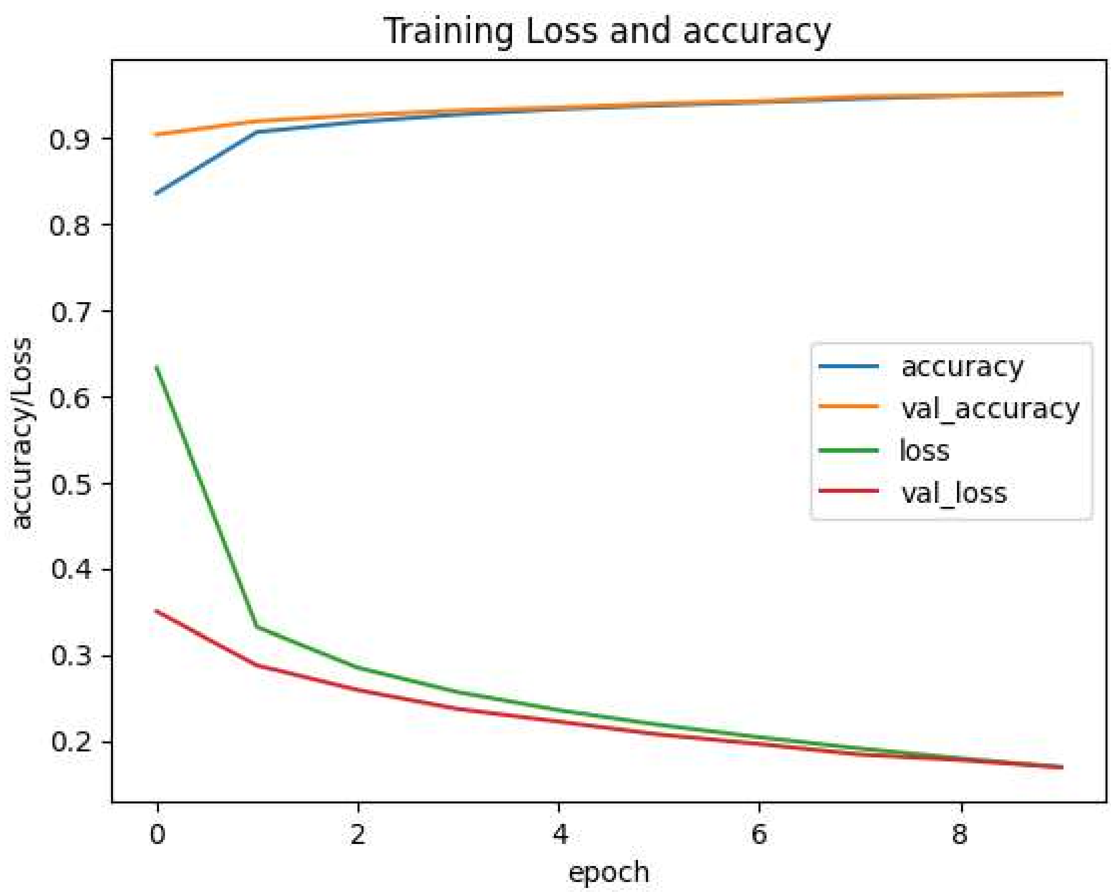
Graph represents model accuracy

In [20]:
```
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()
```



graph represents the model’s loss

In [21]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training Loss and accuracy')
plt.ylabel('accuracy/Loss')
plt.xlabel('epoch')
plt.legend(['accuracy', 'val_accuracy','loss','val_loss'])
plt.show()
```



Conclusion: With above code We can see, that throughout the epochs, our model accuracy increases and our model loss decreases,that is good since our model gains confidence with its predictions.

1. The two losses (loss and val_loss) are decreasing and the accuracy (accuracy and val_accuracy)are increasing. So this indicates the model is trained in a good way.
2. The val_accuracy is the measure of how good the predictions of your model are. So In this case, it looks like the model is well trained after 10 epochs

Save the model

In [22]:
```
pwd
```

Out[22]:
```
'C:\\Users\\subod\\OneDrive\\Documents'
```

In [23]:
```
keras_model_path='C:\\Users\\subod\\OneDrive\\Desktop\\DL'
#'DL.ipynb'
model.save(keras_model_path)
```

```
INFO:tensorflow:Assets written to: C:\Users\subod\OneDrive\Desktop\DL\assets
```

```
INFO:tensorflow:Assets written to: C:\Users\subod\OneDrive\Desktop\DL\assets
```

In [24]:
```
#use the save model
restored_keras_model = tf.keras.models.load_model(keras_model_path)
```