

DEEP LEARNING

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
```

```
#Load the MNIST dataset
```

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

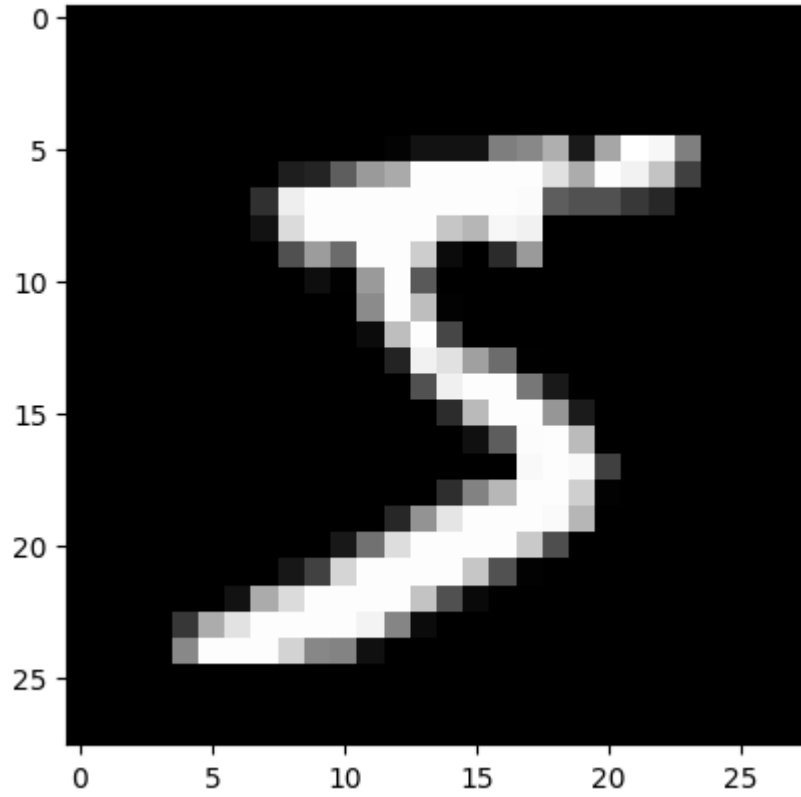
```
➡ Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz  
11490434/11490434 ————— 0s 0us/step
```

```
#display the first 5 images and label from training set
```

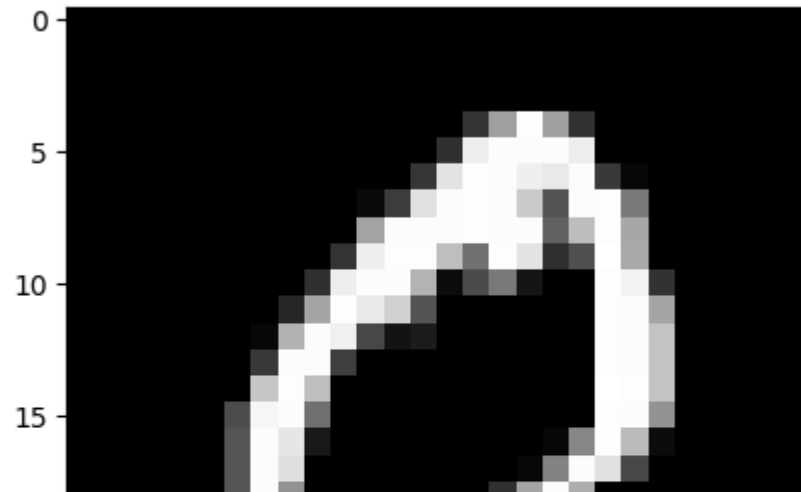
```
for i in range(5):
    plt.imshow(x_train[i],cmap='gray')
    plt.title("Label"+ str(y_train[i]))
    plt.show()
```

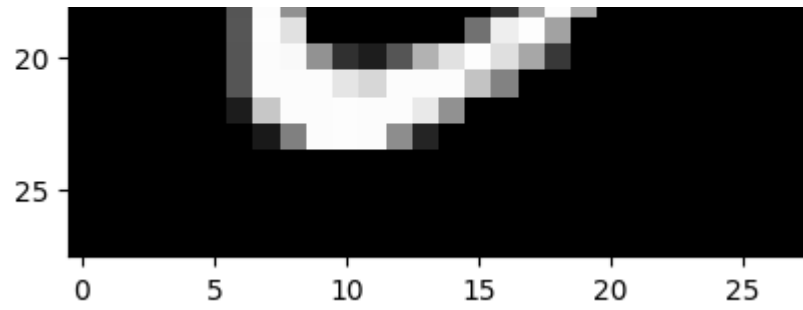


Label5

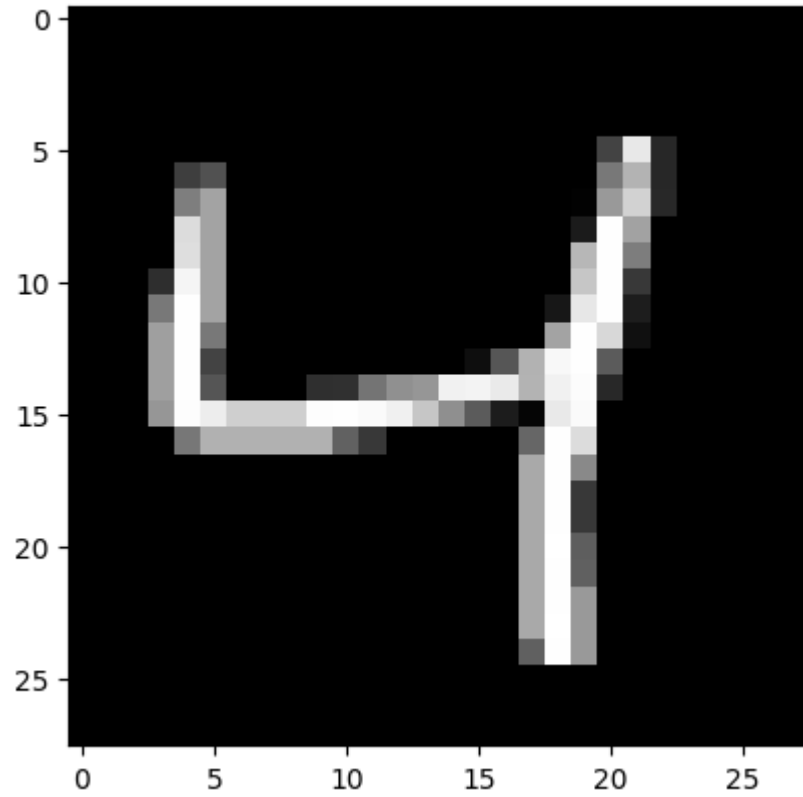


Label0



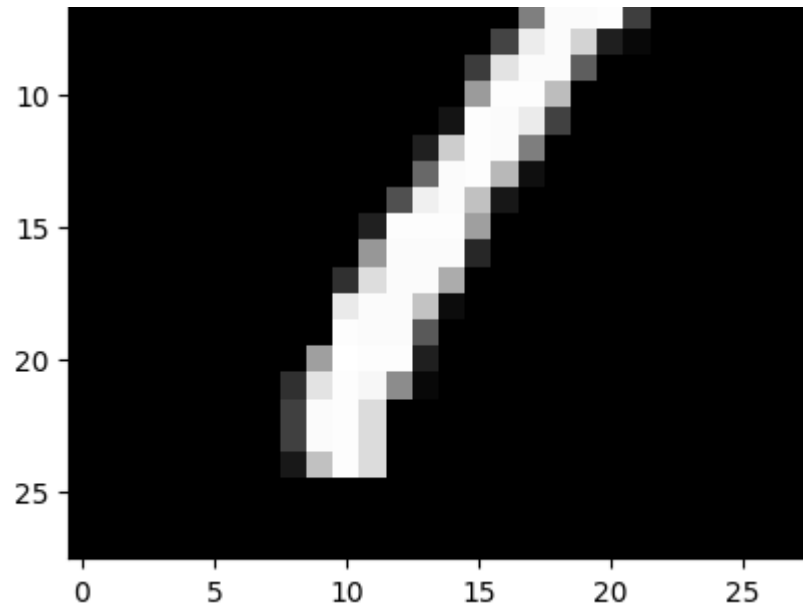


Label4

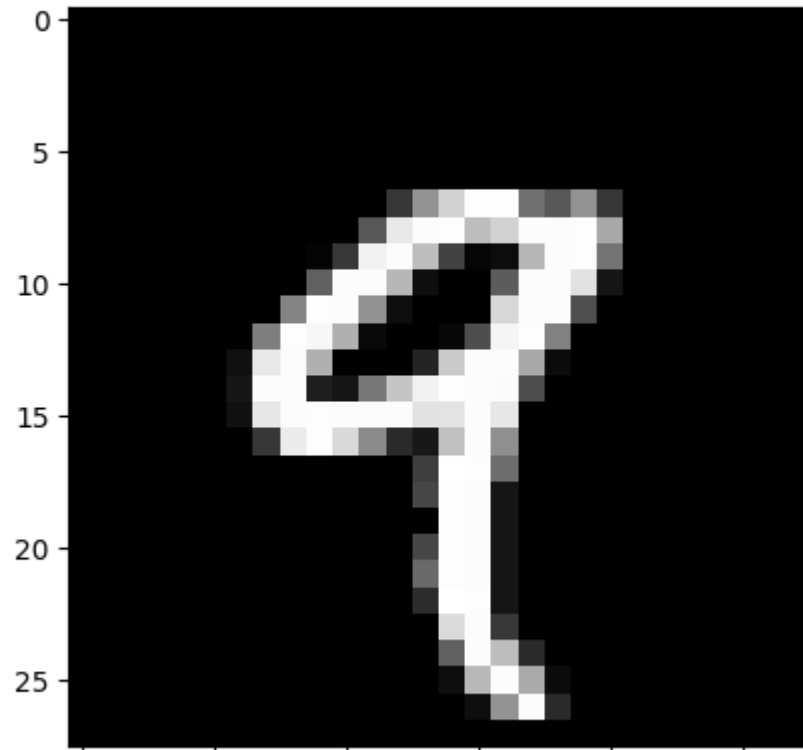


Label1





Label9



0 5 10 15 20 25

```
#Normalize the images (from [0,255] to [0,1])
```

```
x_train=x_train.astype('float32')/255.0  
x_test=x_test.reshape((x_test.shape[0],28,28,1))
```

```
#Check the shapes of the data  
print(f'Training data shape:{x_train.shape},Label shape:{y_train.shape}')
```

```
➞ Training data shape:(60000, 28, 28),Label shape:(60000,)
```

```
print(f'Test data shape:{x_test.shape},Label shape:{y_test.shape}')
```

```
➞ Test data shape:(10000, 28, 28, 1),Label shape:(10000,)
```

```
#one-hot encode the labels
```

```
y_train=tf.keras.utils.to_categorical(y_train,10)  
y_test=tf.keras.utils.to_categorical(y_test,10)
```

```
#Buid the CNN model
```

```
model= models.Sequential()
```

```
#First convolutional layer with 32 filters, 3x3 kernel size, and ReLu activation
```

```
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
```

```
➞ /usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`  
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
##second convolutional layer with 64 filters, 3x3 kernel size, and ReLu activation
```

```
model.add(layers.Conv2D(64,(3,3),activation='relu'))

#maxpooling layer to downsample by 2x2

model.add(layers.MaxPooling2D((2,2)))

#dropout layer for regularization

model.add(layers.Dropout(0.25))

#flatten the feature maps into a 1D feature vector

model.add(layers.Flatten())

#fully connected dense layer with 128 units and relu activation

model.add(layers.Dense(128,activation='relu'))

#Dropout layer to prevent overfitting

model.add(layers.Dropout(0.5))

#Output layer with 10 units (one for each class) and softmax activation

model.add(layers.Dense(10,activation='softmax'))

#Compile the model

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

#display the summary of model
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
conv2d_1 (Conv2D)	(None, 24, 24, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 12, 12, 64)	0
dropout (Dropout)	(None, 12, 12, 64)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 10)	92,170
dense_1 (Dense)	(None, 128)	1,408
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 113,684 (444.08 KB)

Trainable params: 113,684 (444.08 KB)

```
#train the model
```

```
history=model.fit(x_train,y_train,epochs=5,batch_size=64,validation_data=(x_test,y_test))
```

Epoch 1/5
 938/938 ————— 185s 195ms/step - accuracy: 0.7037 - loss: 1.0348 - val_accuracy: 0.8866 - val_loss: 0.4024
 Epoch 2/5
 938/938 ————— 205s 218ms/step - accuracy: 0.9615 - loss: 0.1496 - val_accuracy: 0.8874 - val_loss: 0.5758
 Epoch 3/5
 938/938 ————— 184s 196ms/step - accuracy: 0.9787 - loss: 0.0810 - val_accuracy: 0.8885 - val_loss: 0.6742
 Epoch 4/5
 938/938 ————— 174s 166ms/step - accuracy: 0.9853 - loss: 0.0611 - val_accuracy: 0.8900 - val_loss: 0.7277
 Epoch 5/5
 938/938 ————— 194s 158ms/step - accuracy: 0.9851 - loss: 0.0538 - val_accuracy: 0.8896 - val_loss: 0.7506


```
#Evaluate the model on the test set
```

```
test_loss,test_acc=model.evaluate(x_test,y_test,verbose=2)
print(f'Test accuracy:{test_acc:.4f}')
```

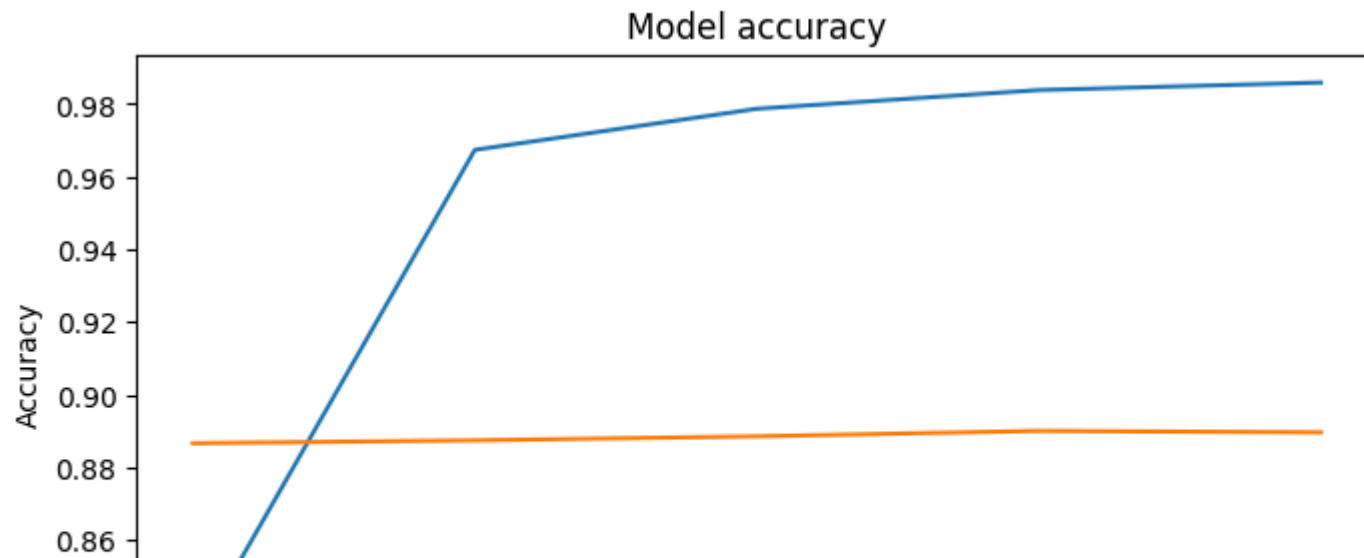
```
➞ 313/313 - 6s - 18ms/step - accuracy: 0.8896 - loss: 0.7506
    Test accuracy:0.8896
```

```
history.history['accuracy']
```

```
➞ [0.8315333127975464,
    0.9673500061035156,
    0.9786666631698608,
    0.9838166832923889,
    0.985883355140686]
```

```
#PLOT training and validation accuracy values
```

```
plt.figure(figsize=(8,4))
plt.plot(history.history['accuracy'],label="Train accuracy")
plt.plot(history.history['val_accuracy'],label="Validation accuracy")
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



#PLOT training and validation loss values

```
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'],label="Train loss")
plt.plot(history.history['val_loss'],label="Validation loss")
plt.title('Model Loss')
```