Load the MNIST dataset correctly :

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Data Preprocessing

Reshape the data to add the channel dimension (since images are grayscale, we have only 1 channel):

In [ ]:
```
# Reshape the data to match the input shape for CNN: (28, 28, 1)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Model Building :

Build the CNN architecture using Keras. A basic CNN might include layers like convolution, pooling, flattening, and dense layers.

In [ ]:
```
model = models.Sequential()

# Add convolutional layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output
model.add(layers.Flatten())

# Add dense layers
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Model Compilation :

Compile the model with the appropriate loss function, optimizer, and metrics.

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

Model Training :

Train the model on the training data.

In [ ]:
```
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
```

Model Evaluation :

After training, evaluate the model on the test data to check the accuracy.

In [ ]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Visualization of Results :

In [ ]:
```
predictions = model.predict(x_test)

# Plot some test images with their predicted and actual labels
for i in range(5):
    plt.imshow(x_test[i].reshape(28, 28), cmap=plt.cm.binary)
    plt.title(f'Predicted: {np.argmax(predictions[i])}, Actual: {np.argmax(y_test[i])
    plt.show()
```

Additional Steps:

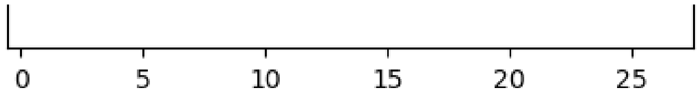
Data Augmentation: You can try augmenting the training data (rotation, shifting, etc.) to improve performance.
Hyperparameter Tuning: Experiment with different optimizers, batch sizes, and learning rates to improve accuracy.
This process will help you understand how CNNs work and their application in image recognition tasks like digit classification.

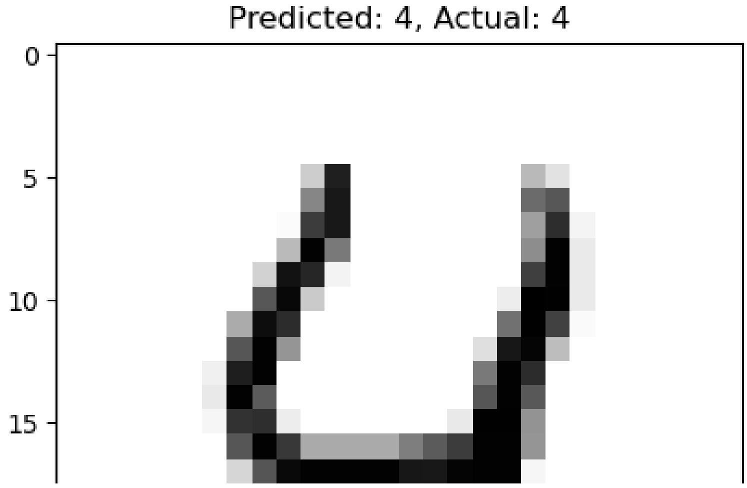
In [ ]:

Build a CNN with Convolutional layers :

In [ ]:

In [6]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape the data to match the input shape for CNN: (28, 28, 1)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

model = models.Sequential()

# Add convolutional layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output
model.add(layers.Flatten())

# Add dense layers
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))


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

model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

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

predictions = model.predict(x_test)

# Plot some test images with their predicted and actual labels
for i in range(5):
    plt.imshow(x_test[i].reshape(28, 28), cmap=plt.cm.binary)
    plt.title(f'Predicted: {np.argmax(predictions[i])}, Actual: {np.argmax(y_test[i])
    plt.show()
```






```
In [ ]:
```

Include MaxPooling layers :

In [8]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess the data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

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

# Build the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')

# Model summary
model.summary()
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 34s 41ms/step - accuracy: 0.8612 - loss: 0.4544 - val_a
ccuracy: 0.9760 - val_loss: 0.0762
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9825 - loss: 0.0569 - val_a
ccuracy: 0.9849 - val_loss: 0.0531
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 37ms/step - accuracy: 0.9864 - loss: 0.0420 - val_a
ccuracy: 0.9847 - val_loss: 0.0510
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9899 - loss: 0.0331 - val_a
ccuracy: 0.9883 - val_loss: 0.0404
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9929 - loss: 0.0235 - val_a
ccuracy: 0.9856 - val_loss: 0.0472
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9930 - loss: 0.0228 - val_a
ccuracy: 0.9893 - val_loss: 0.0416
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9939 - loss: 0.0175 - val_a
ccuracy: 0.9905 - val_loss: 0.0378
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9949 - loss: 0.0140 - val_a
ccuracy: 0.9883 - val_loss: 0.0432
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 38ms/step - accuracy: 0.9966 - loss: 0.0110 - val_a
ccuracy: 0.9898 - val_loss: 0.0385
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9962 - loss: 0.0107 - val_a
ccuracy: 0.9915 - val_loss: 0.0363
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9892 - loss: 0.0323
Test accuracy: 0.9923999905586243
```

Model: "sequential_3"

Layer (type)	Output Shape	Param
conv2d_9 (Conv2D)	(None, 26, 26, 32)	3
max_pooling2d_6 (MaxPooling2D)	(None, 13, 13, 32)	
conv2d_10 (Conv2D)	(None, 11, 11, 64)	18,4
max_pooling2d_7 (MaxPooling2D)	(None, 5, 5, 64)	
conv2d_11 (Conv2D)	(None, 3, 3, 64)	36,9
flatten_3 (Flatten)	(None, 576)	
dense_6 (Dense)	(None, 64)	36,9
dense_7 (Dense)	(None, 10)	6

Total params: 279,968 (1.07 MB)

Trainable params: 93,322 (364.54 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 186,646 (729.09 KB)

In [ ]:

```

```

Add Dense layers and the correct output layer with 10 neurons and softmax activation :

In [10]:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess the data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

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

# Build the CNN model with MaxPooling layers
model = models.Sequential()

# 1st Convolutional layer with MaxPooling
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional layer with MaxPooling
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output
model.add(layers.Flatten())

# Fully connected layer with 64 units
model.add(layers.Dense(64, activation='relu'))

# Output layer with 10 units (for 10 classes) and softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 31s 38ms/step - accuracy: 0.8458 - loss: 0.5024 - val_a
ccuracy: 0.9809 - val_loss: 0.0637
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9805 - loss: 0.0590 - val_a
ccuracy: 0.9863 - val_loss: 0.0491
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 39ms/step - accuracy: 0.9878 - loss: 0.0380 - val_a
ccuracy: 0.9868 - val_loss: 0.0468
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 38ms/step - accuracy: 0.9909 - loss: 0.0276 - val_a
ccuracy: 0.9901 - val_loss: 0.0368
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 38ms/step - accuracy: 0.9926 - loss: 0.0228 - val_a
ccuracy: 0.9862 - val_loss: 0.0473
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9933 - loss: 0.0192 - val_a
ccuracy: 0.9899 - val_loss: 0.0405
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9963 - loss: 0.0130 - val_a
ccuracy: 0.9886 - val_loss: 0.0448
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9963 - loss: 0.0111 - val_a
ccuracy: 0.9905 - val_loss: 0.0372
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9960 - loss: 0.0113 - val_a
ccuracy: 0.9891 - val_loss: 0.0446
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 38ms/step - accuracy: 0.9963 - loss: 0.0100 - val_a
ccuracy: 0.9906 - val_loss: 0.0434
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9904 - loss: 0.0353
Test accuracy: 0.9926000237464905
```

In [ ]:

Use the 'adam' optimizer :

In [11]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

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

# Build the CNN model
model = models.Sequential()

# 1st Convolutional layer with 32 filters, followed by MaxPooling
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional layer with 64 filters, followed by MaxPooling
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional layer with 64 filters
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output from the convolutional layers
model.add(layers.Flatten())

# Fully connected (Dense) layer with 64 units and ReLU activation
model.add(layers.Dense(64, activation='relu'))

# Fully connected (Dense) layer with 128 units and ReLU activation (optional)
model.add(layers.Dense(128, activation='relu'))

# Output layer with 10 neurons (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'])

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 34s 41ms/step - accuracy: 0.8381 - loss: 0.5137 - val_a
ccuracy: 0.9777 - val_loss: 0.0765
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 32s 42ms/step - accuracy: 0.9813 - loss: 0.0617 - val_a
ccuracy: 0.9802 - val_loss: 0.0654
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 31s 41ms/step - accuracy: 0.9869 - loss: 0.0433 - val_a
ccuracy: 0.9872 - val_loss: 0.0426
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9906 - loss: 0.0299 - val_a
ccuracy: 0.9871 - val_loss: 0.0428
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9919 - loss: 0.0244 - val_a
ccuracy: 0.9898 - val_loss: 0.0389
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 38ms/step - accuracy: 0.9931 - loss: 0.0212 - val_a
ccuracy: 0.9893 - val_loss: 0.0410
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 42s 39ms/step - accuracy: 0.9939 - loss: 0.0170 - val_a
ccuracy: 0.9891 - val_loss: 0.0419
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9951 - loss: 0.0123 - val_a
ccuracy: 0.9886 - val_loss: 0.0418
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 39ms/step - accuracy: 0.9958 - loss: 0.0130 - val_a
ccuracy: 0.9906 - val_loss: 0.0375
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 39ms/step - accuracy: 0.9957 - loss: 0.0117 - val_a
ccuracy: 0.9890 - val_loss: 0.0479
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9850 - loss: 0.0586
Test accuracy: 0.9887999892234802
```

In [ ]:

```

```

Set the loss function to 'categorical_crossentropy' :

In [12]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

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

# Build the CNN model
model = models.Sequential()

# 1st Convolutional layer with 32 filters and MaxPooling
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional layer with 64 filters and MaxPooling
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional layer with 64 filters
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output from the convolutional layers
model.add(layers.Flatten())

# Fully connected (Dense) layer with 64 units and ReLU activation
model.add(layers.Dense(64, activation='relu'))

# Fully connected (Dense) layer with 128 units and ReLU activation (optional)
model.add(layers.Dense(128, activation='relu'))

# Output layer with 10 neurons (one for each class) and softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model using 'adam' optimizer
model.compile(optimizer='adam',  # Adam optimizer
              loss='categorical_crossentropy',  # Multi-class classification loss
              metrics=['accuracy'])  # Track accuracy during training and testing

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 34s 39ms/step - accuracy: 0.8360 - loss: 0.5240 - val_a
ccuracy: 0.9802 - val_loss: 0.0654
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9809 - loss: 0.0620 - val_a
ccuracy: 0.9856 - val_loss: 0.0504
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9884 - loss: 0.0383 - val_a
ccuracy: 0.9864 - val_loss: 0.0464
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9906 - loss: 0.0315 - val_a
ccuracy: 0.9857 - val_loss: 0.0509
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 38ms/step - accuracy: 0.9915 - loss: 0.0261 - val_a
ccuracy: 0.9892 - val_loss: 0.0390
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 38ms/step - accuracy: 0.9933 - loss: 0.0203 - val_a
ccuracy: 0.9885 - val_loss: 0.0410
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 38ms/step - accuracy: 0.9943 - loss: 0.0181 - val_a
ccuracy: 0.9901 - val_loss: 0.0367
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 30s 40ms/step - accuracy: 0.9943 - loss: 0.0168 - val_a
ccuracy: 0.9886 - val_loss: 0.0450
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9965 - loss: 0.0101 - val_a
ccuracy: 0.9873 - val_loss: 0.0471
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9967 - loss: 0.0106 - val_a
ccuracy: 0.9898 - val_loss: 0.0457
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9860 - loss: 0.0435
Test accuracy: 0.989799976348877
```

In [ ]:

```

```

Track accuracy as the metric :

In [13]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

y_train = tf.keras.utils.to_categorical(y_train, 10)  # One-hot encode the labels
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the CNN model
model = models.Sequential()

# 1st Convolutional layer with 32 filters and MaxPooling
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional layer with 64 filters and MaxPooling
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional layer with 64 filters
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output from the convolutional layers
model.add(layers.Flatten())

# Fully connected (Dense) layer with 64 units and ReLU activation
model.add(layers.Dense(64, activation='relu'))

# Optional: Fully connected (Dense) layer with 128 units and ReLU activation
model.add(layers.Dense(128, activation='relu'))

# Output layer with 10 neurons (one for each class) and softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model using 'adam' optimizer and 'categorical_crossentropy' loss
model.compile(optimizer='adam',               # Adam optimizer
              loss='categorical_crossentropy', # Loss function for multi-class classi
              metrics=['accuracy'])            # Use accuracy as the evaluation metri

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 34s 40ms/step - accuracy: 0.8424 - loss: 0.5126 - val_a
ccuracy: 0.9789 - val_loss: 0.0741
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 46s 47ms/step - accuracy: 0.9800 - loss: 0.0639 - val_a
ccuracy: 0.9833 - val_loss: 0.0543
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9868 - loss: 0.0433 - val_a
ccuracy: 0.9845 - val_loss: 0.0488
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 35ms/step - accuracy: 0.9896 - loss: 0.0310 - val_a
ccuracy: 0.9863 - val_loss: 0.0487
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9919 - loss: 0.0249 - val_a
ccuracy: 0.9888 - val_loss: 0.0420
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9934 - loss: 0.0206 - val_a
ccuracy: 0.9875 - val_loss: 0.0426
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 35ms/step - accuracy: 0.9941 - loss: 0.0166 - val_a
ccuracy: 0.9888 - val_loss: 0.0387
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 35ms/step - accuracy: 0.9942 - loss: 0.0186 - val_a
ccuracy: 0.9889 - val_loss: 0.0417
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 35ms/step - accuracy: 0.9958 - loss: 0.0132 - val_a
ccuracy: 0.9893 - val_loss: 0.0468
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 27s 36ms/step - accuracy: 0.9966 - loss: 0.0101 - val_a
ccuracy: 0.9862 - val_loss: 0.0561
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9836 - loss: 0.0581
Test accuracy: 0.9868999719619751
```

In [ ]:

```

```

Train the model :

In [14]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

y_train = tf.keras.utils.to_categorical(y_train, 10)  # One-hot encode the labels
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the CNN model
model = models.Sequential()

# 1st Convolutional layer with 32 filters and MaxPooling
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional layer with 64 filters and MaxPooling
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional layer with 64 filters
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output from the convolutional layers
model.add(layers.Flatten())

# Fully connected (Dense) layer with 64 units and ReLU activation
model.add(layers.Dense(64, activation='relu'))

# Optional: Fully connected (Dense) layer with 128 units and ReLU activation
model.add(layers.Dense(128, activation='relu'))

# Output layer with 10 neurons (one for each class) and softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model using 'adam' optimizer, 'categorical_crossentropy' loss, and trac
model.compile(optimizer='adam',                # Adam optimizer
              loss='categorical_crossentropy', # Loss function for multi-class classi
              metrics=['accuracy'])            # Track accuracy as the metric

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 35s 42ms/step - accuracy: 0.8256 - loss: 0.5350 - val_a
ccuracy: 0.9716 - val_loss: 0.0934
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.9800 - loss: 0.0649 - val_a
ccuracy: 0.9797 - val_loss: 0.0632
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9876 - loss: 0.0413 - val_a
ccuracy: 0.9843 - val_loss: 0.0536
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9891 - loss: 0.0350 - val_a
ccuracy: 0.9883 - val_loss: 0.0402
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9908 - loss: 0.0284 - val_a
ccuracy: 0.9867 - val_loss: 0.0471
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 35ms/step - accuracy: 0.9932 - loss: 0.0211 - val_a
ccuracy: 0.9894 - val_loss: 0.0400
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9938 - loss: 0.0183 - val_a
ccuracy: 0.9891 - val_loss: 0.0396
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9952 - loss: 0.0148 - val_a
ccuracy: 0.9851 - val_loss: 0.0552
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9943 - loss: 0.0167 - val_a
ccuracy: 0.9898 - val_loss: 0.0399
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 27s 36ms/step - accuracy: 0.9964 - loss: 0.0109 - val_a
ccuracy: 0.9883 - val_loss: 0.0454
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9866 - loss: 0.0500
Test accuracy: 0.9902999997138977
```

In [ ]:

Track both training and validation accuracy/loss :

In [15]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

y_train = tf.keras.utils.to_categorical(y_train, 10)  # One-hot encode the labels
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the CNN model
model = models.Sequential()

# 1st Convolutional layer with 32 filters and MaxPooling
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional layer with 64 filters and MaxPooling
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional layer with 64 filters
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output from the convolutional layers
model.add(layers.Flatten())

# Fully connected (Dense) layer with 64 units and ReLU activation
model.add(layers.Dense(64, activation='relu'))

# Optional: Fully connected (Dense) layer with 128 units and ReLU activation
model.add(layers.Dense(128, activation='relu'))

# Output layer with 10 neurons (one for each class) and softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model using 'adam' optimizer, 'categorical_crossentropy' loss, and trac
model.compile(optimizer='adam',               # Adam optimizer
              loss='categorical_crossentropy', # Loss function for multi-class classi
              metrics=['accuracy'])            # Track accuracy as the metric

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 34s 40ms/step - accuracy: 0.8407 - loss: 0.5087 - val_a
ccuracy: 0.9769 - val_loss: 0.0797
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 26s 34ms/step - accuracy: 0.9801 - loss: 0.0666 - val_a
ccuracy: 0.9832 - val_loss: 0.0537
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 39ms/step - accuracy: 0.9850 - loss: 0.0472 - val_a
ccuracy: 0.9860 - val_loss: 0.0498
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9879 - loss: 0.0350 - val_a
ccuracy: 0.9866 - val_loss: 0.0452
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 27s 36ms/step - accuracy: 0.9909 - loss: 0.0287 - val_a
ccuracy: 0.9876 - val_loss: 0.0430
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 27s 36ms/step - accuracy: 0.9928 - loss: 0.0222 - val_a
ccuracy: 0.9892 - val_loss: 0.0395
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 44s 41ms/step - accuracy: 0.9947 - loss: 0.0174 - val_a
ccuracy: 0.9890 - val_loss: 0.0410
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 41ms/step - accuracy: 0.9953 - loss: 0.0144 - val_a
ccuracy: 0.9891 - val_loss: 0.0428
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 30s 40ms/step - accuracy: 0.9949 - loss: 0.0150 - val_a
ccuracy: 0.9903 - val_loss: 0.0432
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9960 - loss: 0.0119 - val_a
ccuracy: 0.9885 - val_loss: 0.0455
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9868 - loss: 0.0443
Test accuracy: 0.9905999898910522
```

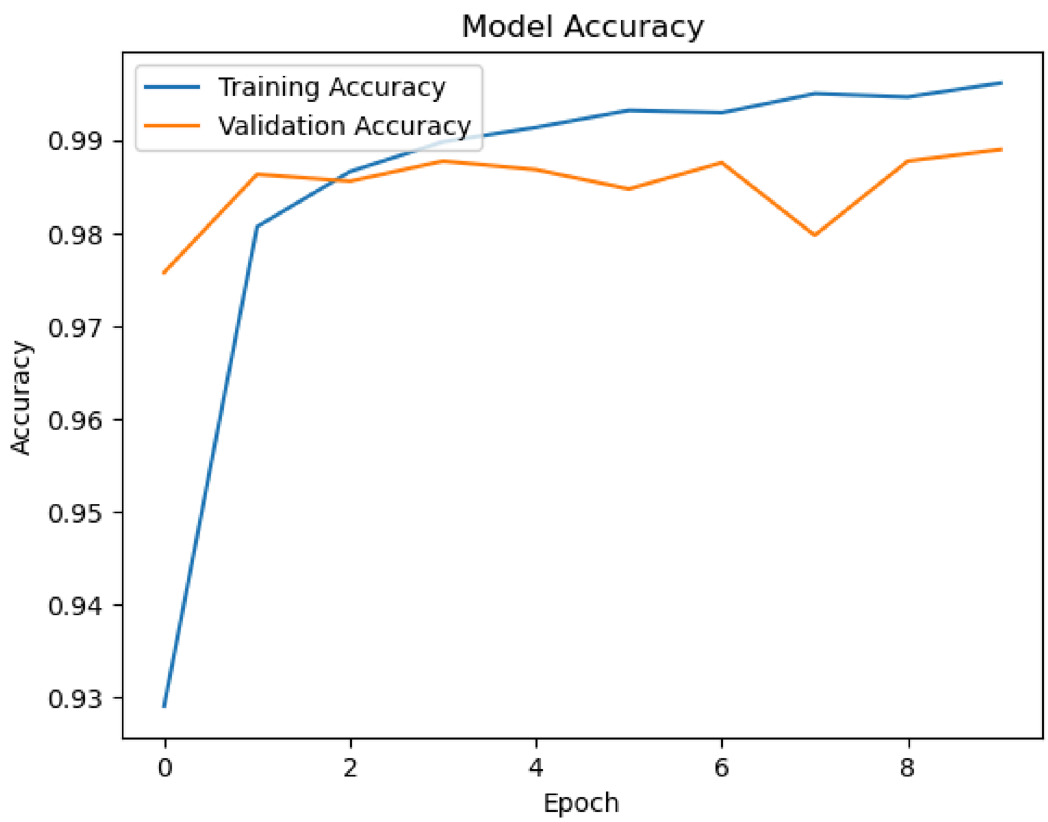
In [ ]:

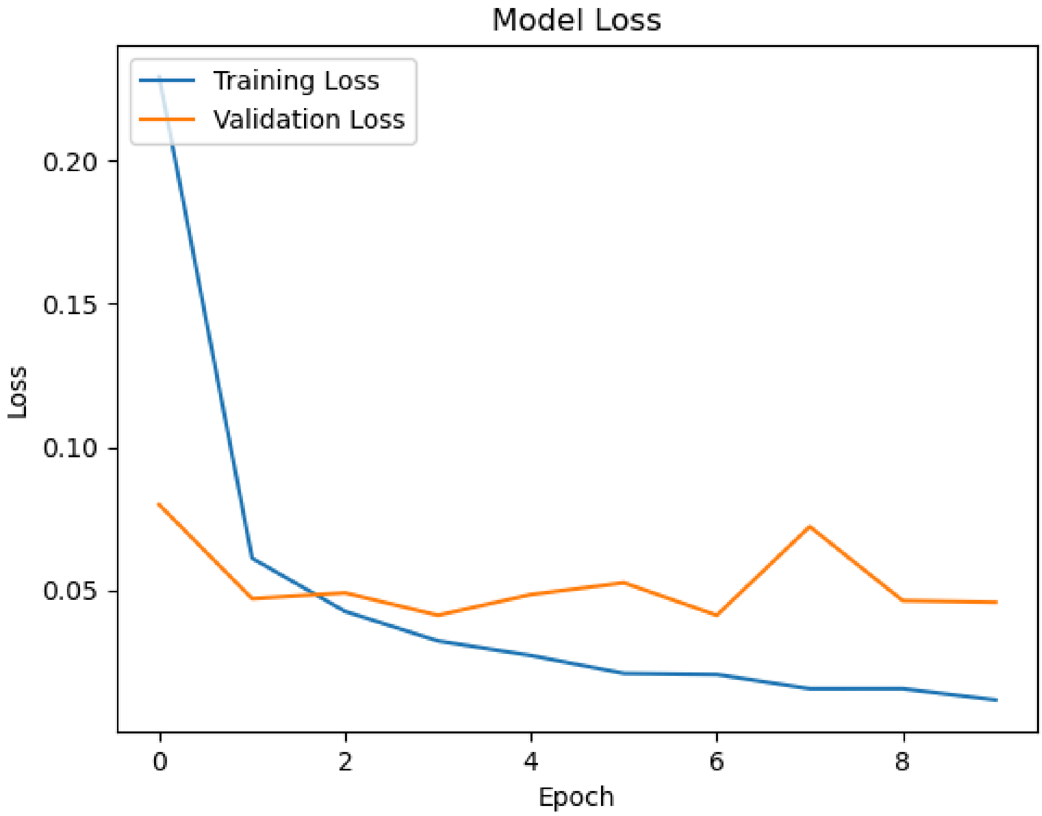
Evaluate the model on test data :

In [16]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
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.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

y_train = tf.keras.utils.to_categorical(y_train, 10)  # One-hot encode the labels
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the CNN model
model = models.Sequential()

# 1st Convolutional layer with 32 filters and MaxPooling
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional layer with 64 filters and MaxPooling
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional layer with 64 filters
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output from the convolutional layers
model.add(layers.Flatten())

# Fully connected (Dense) layer with 64 units and ReLU activation
model.add(layers.Dense(64, activation='relu'))

# Optional: Fully connected (Dense) layer with 128 units and ReLU activation
model.add(layers.Dense(128, activation='relu'))

# Output layer with 10 neurons (one for each class) and softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model using 'adam' optimizer, 'categorical_crossentropy' loss, and trac
model.compile(optimizer='adam',               # Adam optimizer
              loss='categorical_crossentropy', # Loss function for multi-class classi
              metrics=['accuracy'])            # Track accuracy as the metric

# Train the model and track training/validation accuracy & loss
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

```
plt.legend(loc='upper left')
plt.show()

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 31s 37ms/step - accuracy: 0.8398 - loss: 0.5140 - val_a
ccuracy: 0.9758 - val_loss: 0.0798
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9791 - loss: 0.0667 - val_a
ccuracy: 0.9863 - val_loss: 0.0471
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9870 - loss: 0.0410 - val_a
ccuracy: 0.9856 - val_loss: 0.0490
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 42s 38ms/step - accuracy: 0.9903 - loss: 0.0294 - val_a
ccuracy: 0.9877 - val_loss: 0.0413
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 38ms/step - accuracy: 0.9913 - loss: 0.0260 - val_a
ccuracy: 0.9868 - val_loss: 0.0485
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 37ms/step - accuracy: 0.9936 - loss: 0.0195 - val_a
ccuracy: 0.9847 - val_loss: 0.0526
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 28s 38ms/step - accuracy: 0.9934 - loss: 0.0184 - val_a
ccuracy: 0.9876 - val_loss: 0.0412
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9953 - loss: 0.0142 - val_a
ccuracy: 0.9797 - val_loss: 0.0722
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9953 - loss: 0.0140 - val_a
ccuracy: 0.9877 - val_loss: 0.0464
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 38ms/step - accuracy: 0.9963 - loss: 0.0104 - val_a
ccuracy: 0.9890 - val_loss: 0.0458
```






```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9867 - loss: 0.0417
Test accuracy: 0.9908000230789185
```

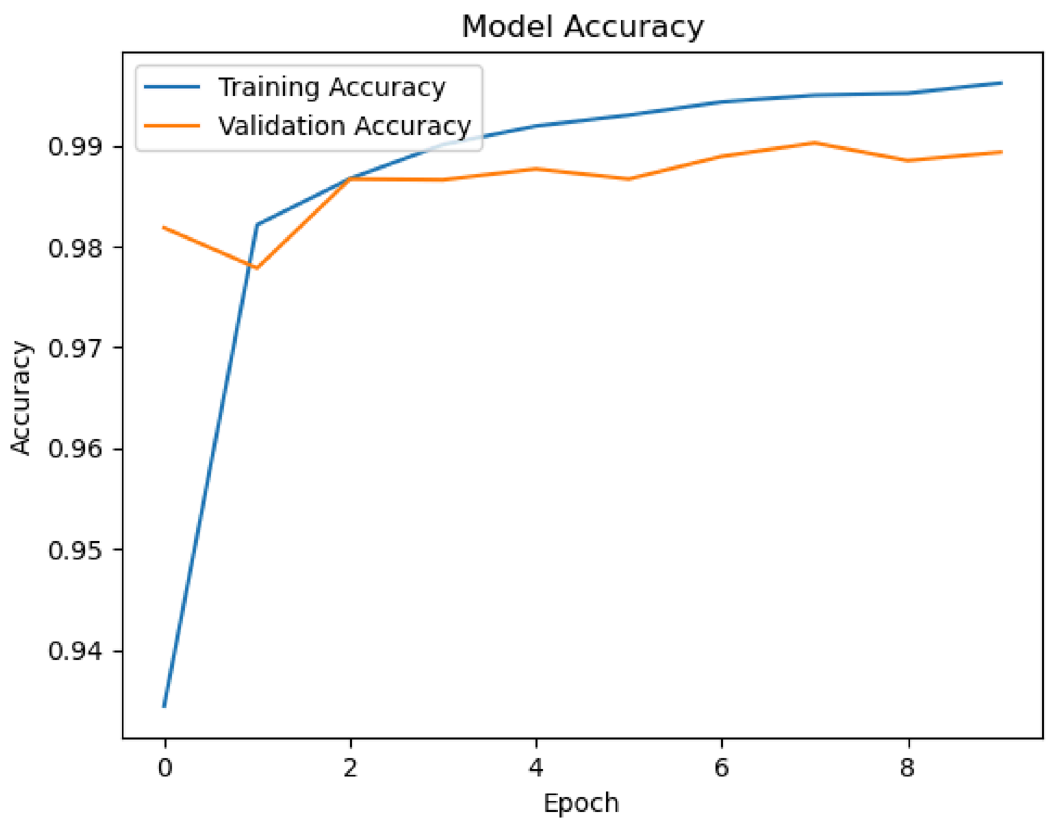
In [ ]:

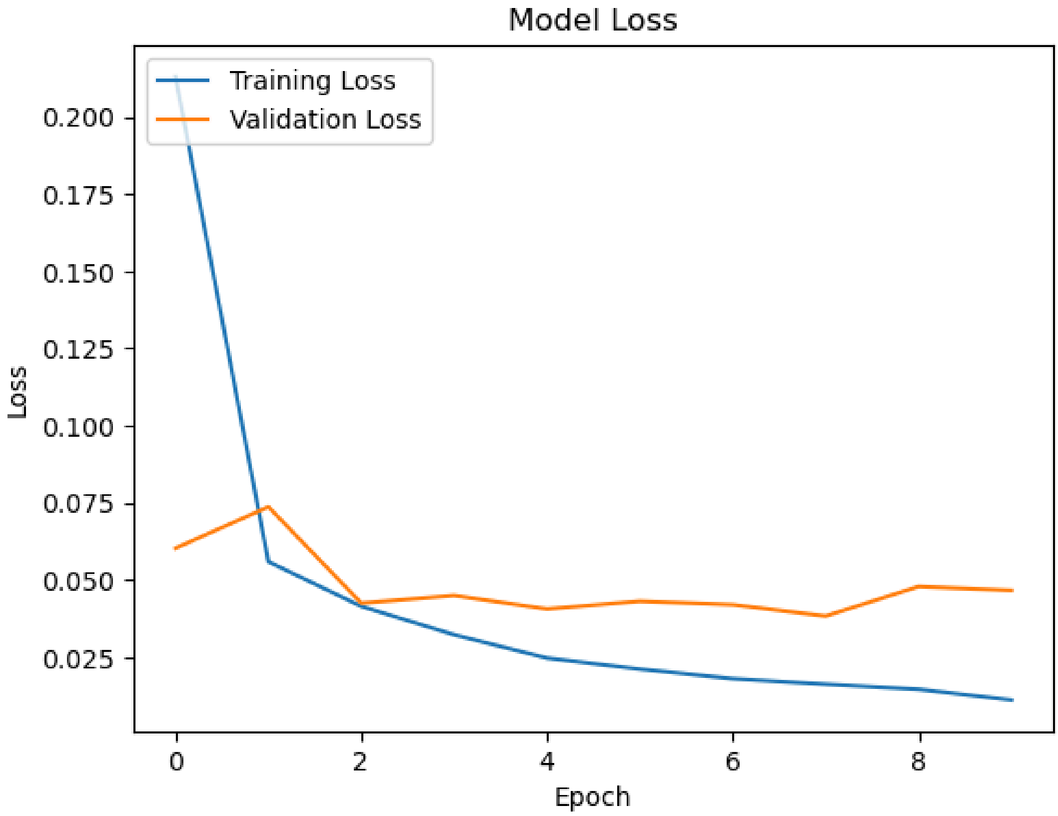
Plot accuracy and loss graphs :

In [17]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
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.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

y_train = tf.keras.utils.to_categorical(y_train, 10)  # One-hot encode the labels
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the CNN model
model = models.Sequential()

# 1st Convolutional layer with 32 filters and MaxPooling
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional layer with 64 filters and MaxPooling
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional layer with 64 filters
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output from the convolutional layers
model.add(layers.Flatten())

# Fully connected (Dense) layer with 64 units and ReLU activation
model.add(layers.Dense(64, activation='relu'))

# Optional: Fully connected (Dense) layer with 128 units and ReLU activation
model.add(layers.Dense(128, activation='relu'))

# Output layer with 10 neurons (one for each class) and softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model using 'adam' optimizer, 'categorical_crossentropy' loss, and trac
model.compile(optimizer='adam',                # Adam optimizer
              loss='categorical_crossentropy', # Loss function for multi-class classi
              metrics=['accuracy'])            # Track accuracy as the metric

# Train the model and track training/validation accuracy & loss
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

```
plt.legend(loc='upper left')
plt.show()

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.8449 - loss: 0.4984 - val_a
ccuracy: 0.9818 - val_loss: 0.0603
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 30s 40ms/step - accuracy: 0.9798 - loss: 0.0607 - val_a
ccuracy: 0.9778 - val_loss: 0.0737
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 39ms/step - accuracy: 0.9857 - loss: 0.0437 - val_a
ccuracy: 0.9867 - val_loss: 0.0425
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 42s 40ms/step - accuracy: 0.9907 - loss: 0.0300 - val_a
ccuracy: 0.9866 - val_loss: 0.0449
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 29s 39ms/step - accuracy: 0.9926 - loss: 0.0229 - val_a
ccuracy: 0.9877 - val_loss: 0.0406
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 30s 40ms/step - accuracy: 0.9931 - loss: 0.0206 - val_a
ccuracy: 0.9867 - val_loss: 0.0431
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 30s 40ms/step - accuracy: 0.9946 - loss: 0.0165 - val_a
ccuracy: 0.9889 - val_loss: 0.0420
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 30s 40ms/step - accuracy: 0.9955 - loss: 0.0143 - val_a
ccuracy: 0.9902 - val_loss: 0.0383
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.9959 - loss: 0.0118 - val_a
ccuracy: 0.9885 - val_loss: 0.0478
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 30s 40ms/step - accuracy: 0.9967 - loss: 0.0096 - val_a
ccuracy: 0.9893 - val_loss: 0.0466
```






```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9855 - loss: 0.0506
Test accuracy: 0.9896000027656555
```