In [20]:
```python
# Importing required libraries
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
# Loading the MNIST dataset
(x_train, x_test), (y_train, y_test) = mnist.load_data()
```

In [22]:
```python
# Reshaping the input data to fit the model
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

In [24]:
```python
# Normalizing the input data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

In [25]:
```python
# Converting the target variable to categorical
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

In [26]:
```python
# Creating a sequential model
model = Sequential()

# Adding the first convolutional layer
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))

# Adding the second convolutional layer
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

# Adding a pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Adding a flattening layer
model.add(Flatten())

# Adding a dense layer
model.add(Dense(128, activation='relu'))

# Adding an output layer
model.add(Dense(num_classes, activation='softmax'))
```

In [27]:
```python
# Compiling the model
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy
```

In [28]:
```python
# Training the model
history = model.fit(x_train, y_train, batch_size=128, epochs=10, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

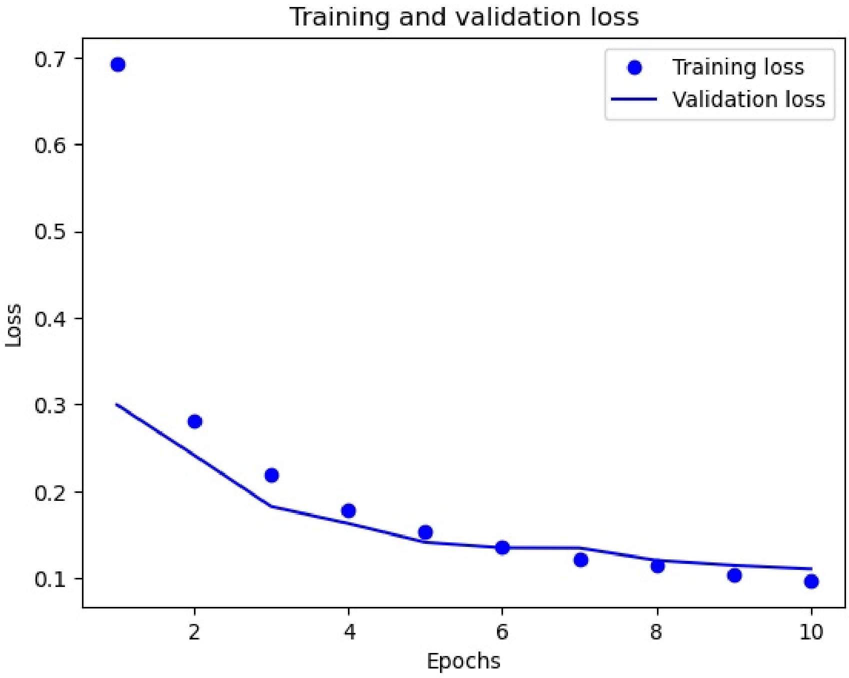
```
Epoch 1/10
469/469 [==============================] - 81s 169ms/step - loss: 0.6929 - accuracy: 0.7909 - val_loss: 0.2997
- val_accuracy: 0.9133
Epoch 2/10
469/469 [==============================] - 83s 176ms/step - loss: 0.2809 - accuracy: 0.9169 - val_loss: 0.2418
- val_accuracy: 0.9269
Epoch 3/10
469/469 [==============================] - 79s 169ms/step - loss: 0.2192 - accuracy: 0.9352 - val_loss: 0.1826
- val_accuracy: 0.9451
Epoch 4/10
469/469 [==============================] - 76s 162ms/step - loss: 0.1786 - accuracy: 0.9460 - val_loss: 0.1631
- val_accuracy: 0.9525
Epoch 5/10
469/469 [==============================] - 76s 161ms/step - loss: 0.1536 - accuracy: 0.9545 - val_loss: 0.1414
- val_accuracy: 0.9579
Epoch 6/10
469/469 [==============================] - 75s 161ms/step - loss: 0.1367 - accuracy: 0.9588 - val_loss: 0.1350
- val_accuracy: 0.9603
Epoch 7/10
469/469 [==============================] - 74s 157ms/step - loss: 0.1224 - accuracy: 0.9632 - val_loss: 0.1347
- val_accuracy: 0.9608
Epoch 8/10
469/469 [==============================] - 75s 159ms/step - loss: 0.1155 - accuracy: 0.9646 - val_loss: 0.1207
- val_accuracy: 0.9648
Epoch 9/10
469/469 [==============================] - 77s 165ms/step - loss: 0.1044 - accuracy: 0.9675 - val_loss: 0.1148
- val_accuracy: 0.9656
Epoch 10/10
469/469 [==============================] - 80s 171ms/step - loss: 0.0971 - accuracy: 0.9703 - val_loss: 0.1106
- val_accuracy: 0.9653
Test loss: 0.11059534549713135
Test accuracy: 0.9653000235557556
```

In [29]:
```python
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320       
                                                                 
 conv2d_3 (Conv2D)           (None, 24, 24, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 12, 12, 64)       0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 9216)              0         
                                                                 
 dense_2 (Dense)             (None, 128)               1179776   
                                                                 
 dense_3 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

In [30]:

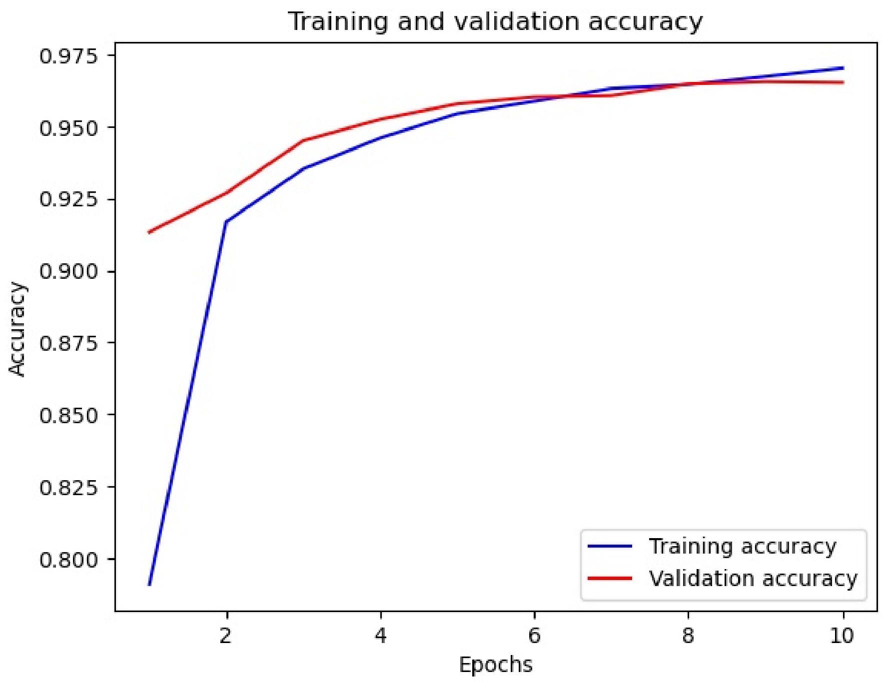
```python
import matplotlib.pyplot as plt

# Plot the training and validation loss over each epoch
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(train_loss) + 1)

plt.plot(epochs, train_loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [31]:

```python
import matplotlib.pyplot as plt

# get the training and validation accuracy values from the history object
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# create a range of x-axis values for the chart
epochs = range(1, len(acc) + 1)

# plot the training and validation accuracy values as two separate lines
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')

# set the chart title and axis labels
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

# add a legend to the chart
plt.legend()
```

```
# display the chart
plt.show()
```



In [32]:
```
import numpy as np
# Generate predictions on test data using the trained model
y_pred = model.predict(x_test)

# Convert predictions to integer labels
y_pred_labels = np.argmax(y_pred, axis=1)

# Print sample predictions and actual labels
print("Sample predictions:", y_pred_labels[:10])
print("Actual labels:", y_test[:10])
```

```
313/313 [==============================] - 6s 18ms/step
Sample predictions: [7 2 1 0 4 1 4 9 5 9]
Actual labels: [[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

In [33]:
```
import sklearn.metrics
from sklearn.metrics import confusion_matrix, classification_report

# Generate integer labels for test data
y_test_labels = np.argmax(y_test, axis=1)

# Generate integer labels for predictions
y_pred_labels = np.argmax(y_pred, axis=1)

# Compute confusion matrix and classification report
cm = confusion_matrix(y_test_labels, y_pred_labels)
report = classification_report(y_test_labels, y_pred_labels)

# Print confusion matrix and classification report
print("Confusion Matrix:\n", cm)
print("\nClassification Report:\n", report)
```

```
Confusion Matrix:
 [[ 967    0    0    2    1    1    5    1    1    2]
 [   0 1115    2    4    0    2    3    5    3    1]
 [   4    2  987   17    5    1    5    5    4    2]
 [   0    0    2  988    1    2    1    5    2    9]
 [   0    0    2    0  944    1    8    1    1   25]
 [   2    0    0   21    2  853    7    2    1    4]
 [   3    1    0    1    9    6  937    0    1    0]
 [   0    2    9    6    3    0    0  988    1   19]
 [   4    0    2   25    7    6    6    3  902   19]
 [   3    1    0   13   12    6    0    2    0  972]]

Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.98      0.99      1135
           2       0.98      0.96      0.97      1032
           3       0.92      0.98      0.95      1010
           4       0.96      0.96      0.96       982
           5       0.97      0.96      0.96       892
           6       0.96      0.98      0.97       958
           7       0.98      0.96      0.97      1028
           8       0.98      0.93      0.95       974
           9       0.92      0.96      0.94      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.96      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```