In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Drop
from tensorflow.keras.utils import to_categorical
```

In [2]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [3]:
```
# Reshape to (28, 28, 1) for CNN input and normalize the pixel values (0-255)
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') / 255.0
```

In [4]:
```
# One-hot encode the labels
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

In [5]:
```
# Build the CNN model
model = Sequential()

# First convolutional layer + pooling
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 2
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolutional layer + pooling
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

In [6]:
```
# Flatten the output for the fully connected layers
model.add(Flatten())

# Fully connected layer with 128 neurons
model.add(Dense(128, activation='relu'))

# Dropout layer to prevent overfitting
model.add(Dropout(0.5))

# Output layer with 10 neurons for the 10 digit classes (0-9)
model.add(Dense(10, activation='softmax'))
```

In [7]:
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_sp

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)
```

```
Epoch 1/10
375/375 - 39s - loss: 0.3561 - accuracy: 0.8916 - val_loss: 0.0798 - val_accu
racy: 0.9768 - 39s/epoch - 105ms/step
Epoch 2/10
375/375 - 36s - loss: 0.1113 - accuracy: 0.9675 - val_loss: 0.0566 - val_accu
racy: 0.9848 - 36s/epoch - 97ms/step
Epoch 3/10
375/375 - 36s - loss: 0.0809 - accuracy: 0.9756 - val_loss: 0.0450 - val_accu
racy: 0.9876 - 36s/epoch - 96ms/step
Epoch 4/10
375/375 - 36s - loss: 0.0648 - accuracy: 0.9806 - val_loss: 0.0394 - val_accu
racy: 0.9883 - 36s/epoch - 95ms/step
Epoch 5/10
375/375 - 36s - loss: 0.0554 - accuracy: 0.9836 - val_loss: 0.0370 - val_accu
racy: 0.9895 - 36s/epoch - 95ms/step
Epoch 6/10
375/375 - 35s - loss: 0.0470 - accuracy: 0.9859 - val_loss: 0.0401 - val_accu
racy: 0.9882 - 35s/epoch - 94ms/step
Epoch 7/10
375/375 - 35s - loss: 0.0418 - accuracy: 0.9871 - val_loss: 0.0387 - val_accu
racy: 0.9892 - 35s/epoch - 93ms/step
Epoch 8/10
375/375 - 36s - loss: 0.0370 - accuracy: 0.9887 - val_loss: 0.0348 - val_accu
racy: 0.9904 - 36s/epoch - 96ms/step
Epoch 9/10
375/375 - 35s - loss: 0.0326 - accuracy: 0.9900 - val_loss: 0.0343 - val_accu
racy: 0.9913 - 35s/epoch - 94ms/step
Epoch 10/10
375/375 - 34s - loss: 0.0300 - accuracy: 0.9903 - val_loss: 0.0396 - val_accu
racy: 0.9898 - 34s/epoch - 92ms/step
313/313 - 3s - loss: 0.0263 - accuracy: 0.9923 - 3s/epoch - 10ms/step
```

In [8]:
```
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
Test Accuracy: 0.9923
```