In [41]:
```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```

In [42]:
```
# Set the number of categories and define the shape of input images
num_categories = 10
input_shape = (28, 28, 1)
```

In [43]:
```
# Load the MNIST dataset and split it into training and testing sets
(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()
```

In [44]:
```
# Normalize the pixel values of the images to a range between 0 and 1
train_images = train_images.astype("float32") / 255.0
test_images = test_images.astype("float32") / 255.0
```

In [45]:
```
# Expand the dimensions of the images to include the color channel
train_images = np.expand_dims(train_images, -1)
test_images = np.expand_dims(test_images, -1)
```

In [46]:
```
# Convert class labels to categorical matrices
train_labels = keras.utils.to_categorical(train_labels, num_categories)
test_labels = keras.utils.to_categorical(test_labels, num_categories)
```

In [47]:
```
# Define a modified convolutional neural network architecture
model = keras.Sequential([
    keras.Input(shape=input_shape),
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(128, kernel_size=(3, 3), activation="relu"),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(256, activation="relu"),
    layers.Dense(num_categories, activation="softmax")
])
```

In [48]:
```
# Display the summary of the model architecture
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_4 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_5 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_8 (Conv2D)           (None, 3, 3, 128)         73856

 flatten_2 (Flatten)         (None, 1152)              0

 dropout_2 (Dropout)         (None, 1152)              0

 dense_4 (Dense)             (None, 256)               295168

 dense_5 (Dense)             (None, 10)                2570

=================================================================
Total params: 390410 (1.49 MB)
Trainable params: 390410 (1.49 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [49]:
```
# Compile the model with specified optimizer and loss function
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

In [50]:
```
# Set the batch size and number of epochs for training
batch_size = 128
epochs = 15
```

In [51]:
```
# Train the model using the training data
model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/15
469/469 [==============================] - 20s 41ms/step - loss: 0.2338 - accuracy: 0.9268
Epoch 2/15
469/469 [==============================] - 19s 40ms/step - loss: 0.0647 - accuracy: 0.9803
Epoch 3/15
469/469 [==============================] - 19s 40ms/step - loss: 0.0501 - accuracy: 0.9840
Epoch 4/15
469/469 [==============================] - 20s 43ms/step - loss: 0.0431 - accuracy: 0.9866
Epoch 5/15
469/469 [==============================] - 20s 43ms/step - loss: 0.0342 - accuracy: 0.9900
Epoch 6/15
469/469 [==============================] - 20s 42ms/step - loss: 0.0304 - accuracy: 0.9904
Epoch 7/15
469/469 [==============================] - 20s 42ms/step - loss: 0.0272 - accuracy: 0.9915
Epoch 8/15
469/469 [==============================] - 20s 42ms/step - loss: 0.0228 - accuracy: 0.9926
Epoch 9/15
469/469 [==============================] - 20s 42ms/step - loss: 0.0224 - accuracy: 0.9929
Epoch 10/15
469/469 [==============================] - 19s 40ms/step - loss: 0.0198 - accuracy: 0.9934
Epoch 11/15
469/469 [==============================] - 19s 41ms/step - loss: 0.0172 - accuracy: 0.9943
Epoch 12/15
469/469 [==============================] - 20s 42ms/step - loss: 0.0164 - accuracy: 0.9945
Epoch 13/15
469/469 [==============================] - 20s 42ms/step - loss: 0.0144 - accuracy: 0.9951
Epoch 14/15
469/469 [==============================] - 22s 47ms/step - loss: 0.0144 - accuracy: 0.9950
Epoch 15/15
469/469 [==============================] - 20s 42ms/step - loss: 0.0122 - accuracy: 0.9963
```

Out[51]:
```
<keras.src.callbacks.History at 0x1e7cd216390>
```

In [52]:
```
# Evaluate the model performance on the test data
evaluation = model.evaluate(test_images, test_labels, verbose=0)
print("Test loss:", evaluation[0])
print("Test accuracy:", evaluation[1])
```

```
Test loss: 0.021707022562623024
Test accuracy: 0.9937000274658203
```

In [53]:
```
# Save the trained model for future use
model.save('D:\PRABIN CNN')
```

```
INFO:tensorflow:Assets written to: D:\PRABIN CNN\assets
```

In [54]:
```
model.save('MNIST_baseline_model.h5')
```

```
C:\Users\prabi\anaconda3\Lib\site-packages\keras\src\engine\training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

In [55]:
```
model_path = 'D:\Hand-digit/my_model.h5'  # Replace with your desired path
```

In [56]:
```
model.save(model_path)
```

In [57]:
```
from tensorflow.keras.models import load_model

model = load_model('D:\Hand-digit/my_model.h5')  # Replace with your model file
```

In [58]:
```
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import numpy as np

img = load_img('C:/Users/prabi/Downloads/sample_digit_7.png', color_mode='grayscale', target_size=(28, 28))


# Convert the image to an array and normalize
img_array = img_to_array(img) / 255.0

# Reshape the array for the model
img_array = img_array.reshape((1, 28, 28, 1))  # The first 1 is for the batch size
```

In [59]:
```
prediction = model.predict(img_array)
```

```
1/1 [==============================] - 0s 58ms/step
```

In [60]:
```
predicted_class = np.argmax(prediction, axis=1)
```

In [61]:
```
print(f"The model predicts: {predicted_class[0]}")
```

```
The model predicts: 7
```

In [ ]:
```

```