1. Build the Image classification model by dividing the model into following 4 stages:

   a. Loading and preprocessing the image data

   b. Defining the model’s architecture

   c. Training the model

   d. Estimating the model’s performance

In [2]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

a. Loading and preprocessing the image data

In [3]:

```
train_data_dir = 'Datasets/mnist-jpg/train'
test_data_dir = 'Datasets/mnist-jpg/test'

# Image data generator for training data
train_datagen = ImageDataGenerator(
rescale=1.0/255
)

# Image data generator for testing data
test_datagen = ImageDataGenerator(
rescale=1.0/255
)

# Create data generators
train_batch_size = 10000
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(28, 28),  # Resize images to 28x28
    batch_size=train_batch_size,
    class_mode='categorical',
    color_mode='grayscale',# Use 'categorical' for one-hot encoded labels
    shuffle=True,
)

# Load test data without labels (class_mode=None)
test_batch_size = 2000
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(28, 28),  # Resize images to 28x28
    batch_size=test_batch_size,
    class_mode='categorical',  # Use 'categorical' for one-hot encoded labe
    color_mode='grayscale',
    shuffle=True,
)
```

```
Found 60000 images belonging to 10 classes.
Found 60000 images belonging to 10 classes.
```

Selecting first batch containing 10000 images

In [4]:

```
x_train, y_train = train_generator[0]
x_test, y_test = test_generator[0]
```

In [5]:

```
print(x_train.shape, y_train.shape)
```

```
(10000, 28, 28, 1) (10000, 10)
```

b. Defining the model's architecture

In [6]:

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

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

c. Training the model

In [8]:

```
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_te
```

```
Epoch 1/5
157/157 [==============================] - 2s 11ms/step - loss: 0.5566 - a
ccuracy: 0.8428 - val_loss: 0.2578 - val_accuracy: 0.9315
Epoch 2/5
157/157 [==============================] - 1s 10ms/step - loss: 0.2051 - a
ccuracy: 0.9419 - val_loss: 0.1786 - val_accuracy: 0.9500
Epoch 3/5
157/157 [==============================] - 2s 10ms/step - loss: 0.1344 - a
ccuracy: 0.9626 - val_loss: 0.1204 - val_accuracy: 0.9730
Epoch 4/5
157/157 [==============================] - 2s 10ms/step - loss: 0.0943 - a
ccuracy: 0.9733 - val_loss: 0.1029 - val_accuracy: 0.9725
Epoch 5/5
157/157 [==============================] - 2s 10ms/step - loss: 0.0658 - a
ccuracy: 0.9828 - val_loss: 0.0822 - val_accuracy: 0.9795
```

Out[8]:

```
<keras.src.callbacks.History at 0x7f7b05b74690>
```

d. Estimating the model's performance

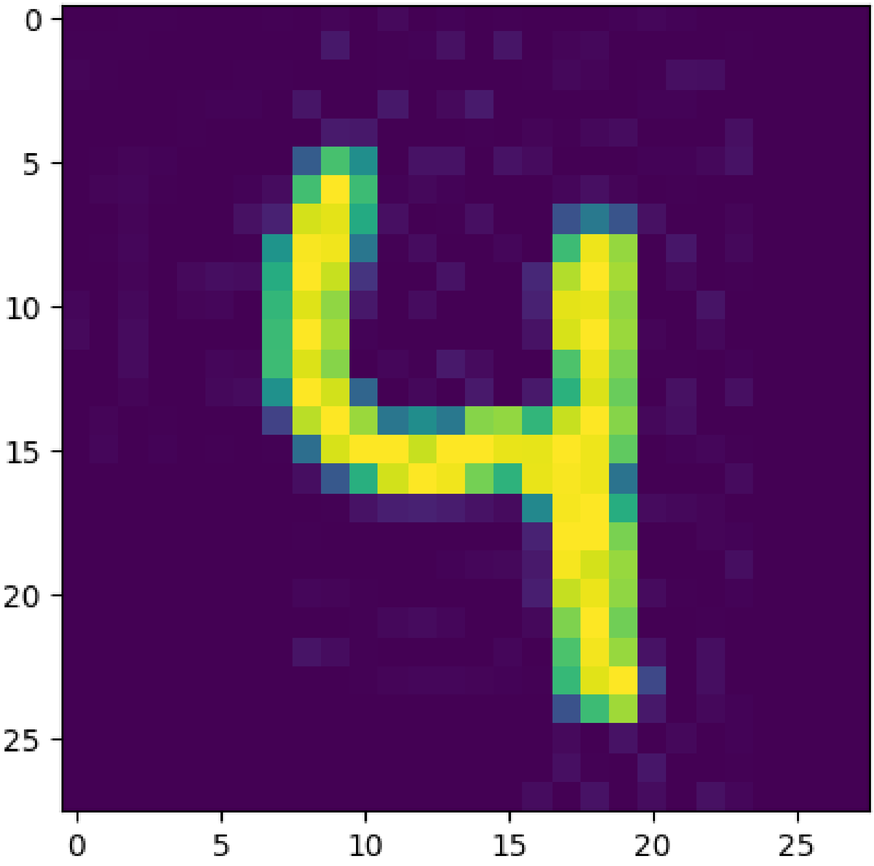
In [10]:

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Loss: ", test_loss)
print("Accuracy: ", test_accuracy)
```

```
63/63 [==============================] - 0s 2ms/step - loss: 0.0822 - accu
racy: 0.9795
Loss:  0.08217164129018784
Accuracy:  0.9794999957084656
```

```
In [12]: n = 30
         plt.imshow(x_test[n])
         predicted_value = model.predict(x_test)
         print("Actual Number: ",np.argmax(y_test[n]))
         print("Predicted Number: ", np.argmax(predicted_value[n]))
```

```
63/63 [==============================] - 0s 2ms/step
Actual Number:  4
Predicted Number:  4
```


0
5
10
15
20
25
0
5
10
15
20
25