```python
'''

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
'''
```

```
'\n\nBuild the Image classification model by dividing the model into
following 4 stages:\na. Loading and preprocessing the image data\nb.
Defining the model’s architecture\nc. Training the model\nd.
Estimating the model’s performance\n'
```

```python
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
train_data_dir = 'mnist-jpg/train'
test_data_dir = '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
labels
    color_mode='grayscale',
    shuffle=True,
)
```

```
Found 60000 images belonging to 10 classes.
Found 10000 images belonging to 10 classes.
```

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

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

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

```
#b. Defining the model's architecture
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.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:\ProgramData\anaconda3\Lib\site-packages\keras\src\layers\
convolutional\base_conv.py:113: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
#c. Training the model
model.fit(x_train, y_train, epochs=5, batch_size=64,
validation_data=(x_test, y_test))
```

```
Epoch 1/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 20s 59ms/step - accuracy: 0.8496 - loss:
0.5394 - val_accuracy: 0.9370 - val_loss: 0.2147
Epoch 2/5
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 6s 40ms/step - accuracy: 0.9400 - loss:
0.2005 - val_accuracy: 0.9565 - val_loss: 0.1484
Epoch 3/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 7s 41ms/step - accuracy: 0.9636 - loss:
0.1256 - val_accuracy: 0.9660 - val_loss: 0.1020
Epoch 4/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 7s 41ms/step - accuracy: 0.9759 - loss:
0.0873 - val_accuracy: 0.9690 - val_loss: 0.0918
Epoch 5/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 6s 40ms/step - accuracy: 0.9843 - loss:
0.0615 - val_accuracy: 0.9710 - val_loss: 0.1001

<keras.src.callbacks.history.History at 0x23f7c3030e0>
```

```
#d. Estimating the model’s performance

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

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9710 - loss:
0.1001
Loss:  0.10011710971593857
Accuracy:  0.9710000157356262
```

```
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]))
```