#intial building of the model using a basic neural network using the MNIST dataset

```
import os

import numpy as np

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train , y_train) ,(x_test,y_test) = mnist.load_data()


x_train = tf.keras.utils.normalize(x_train , axis = 1)
x_test = tf.keras.utils.normalize(x_test,axis=1)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape = (28,28)))
model.add(tf.keras.layers.Dense(128,activation='relu'))
model.add(tf.keras.layers.Dense(64,activation='relu'))
model.add(tf.keras.layers.Dense(32,activation='relu'))
model.add(tf.keras.layers.Dense(10,activation='softmax'))

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

model.fit(x_train,y_train,epochs=3)

model.save('handwritten.keras')
```

#Testing the basic model

```
model = tf.keras.models.load_model('handwritten.keras')
loss,accuracy = model.evaluate(x_test,y_test)
print(accuracy)
print(loss)
```

```
model = tf.keras.models.load_model('handwritten_cnn.keras')
loss,accuracy = model.evaluate(x_test,y_test)
print(accuracy)
print(loss)
```

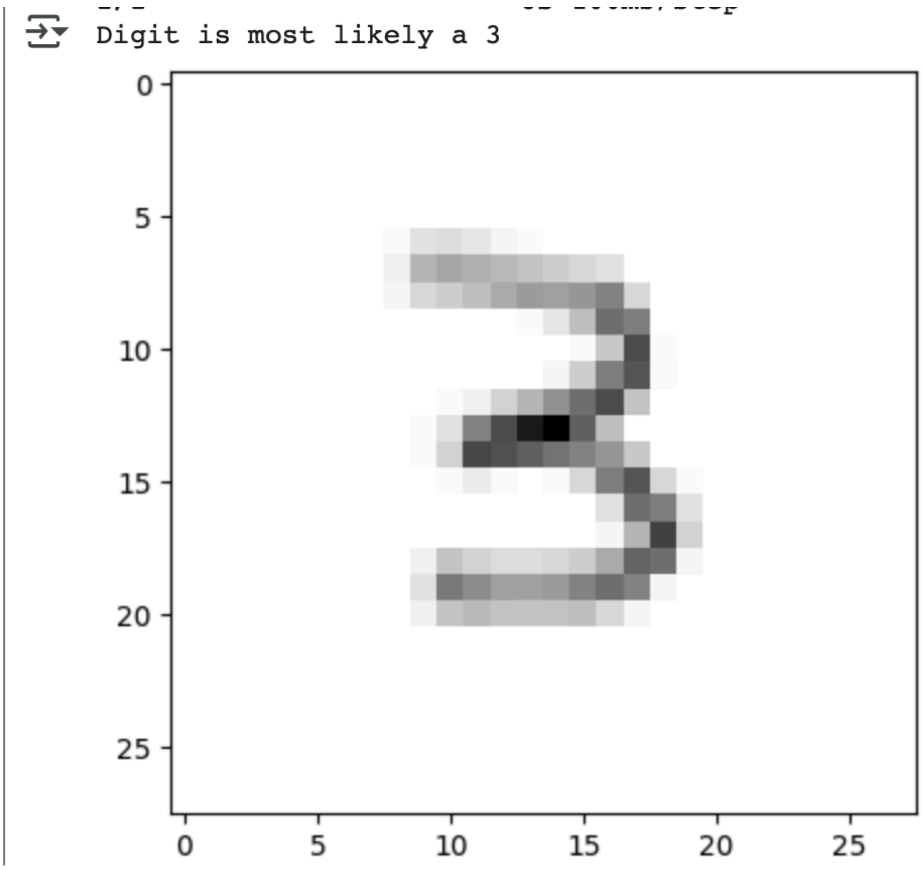
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9659 - loss: 0.1181
0.9711999893188477
0.09777520596981049
```

 # Analysing images from image files and predicting and displaying them

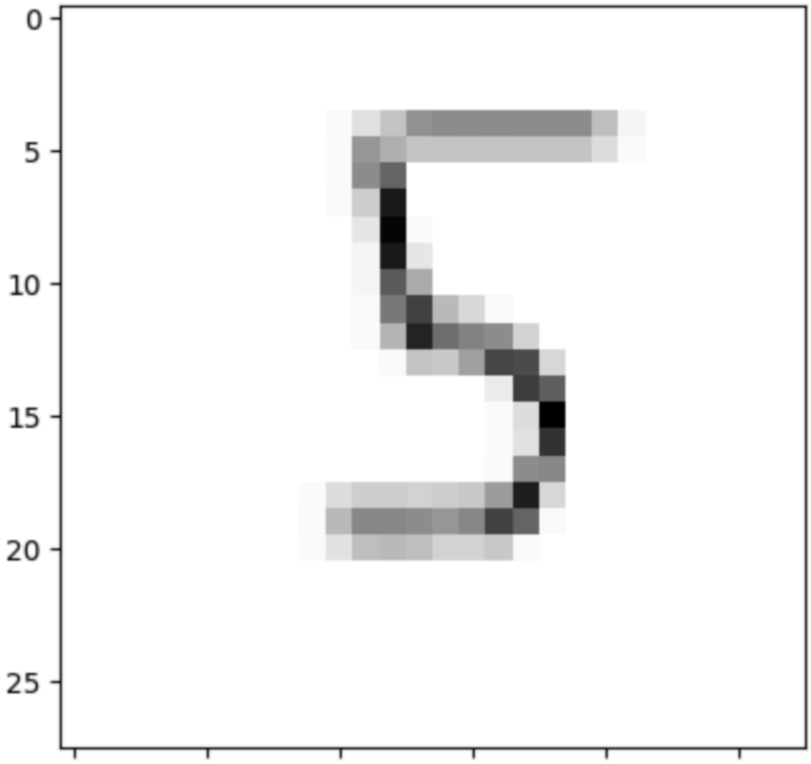
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
model = tf.keras.models.load_model('handwritten.keras')
ImgNum = 1

while os.path.isfile(f"Images/Image{ImgNum}.png"):
  try:
    inputIMG = cv2.imread(f"Images/Image{ImgNum}.png")[:,:,0]
    inputIMG = np.invert(np.array([inputIMG]))
    result = model.predict(inputIMG)
    print(f"Digit is most likely a {np.argmax(result)}")
    plt.imshow(inputIMG[0],cmap=plt.cm.binary)
    plt.show()
  except:
    print("ERROR!")
  finally:
    ImgNum += 1
```

#output

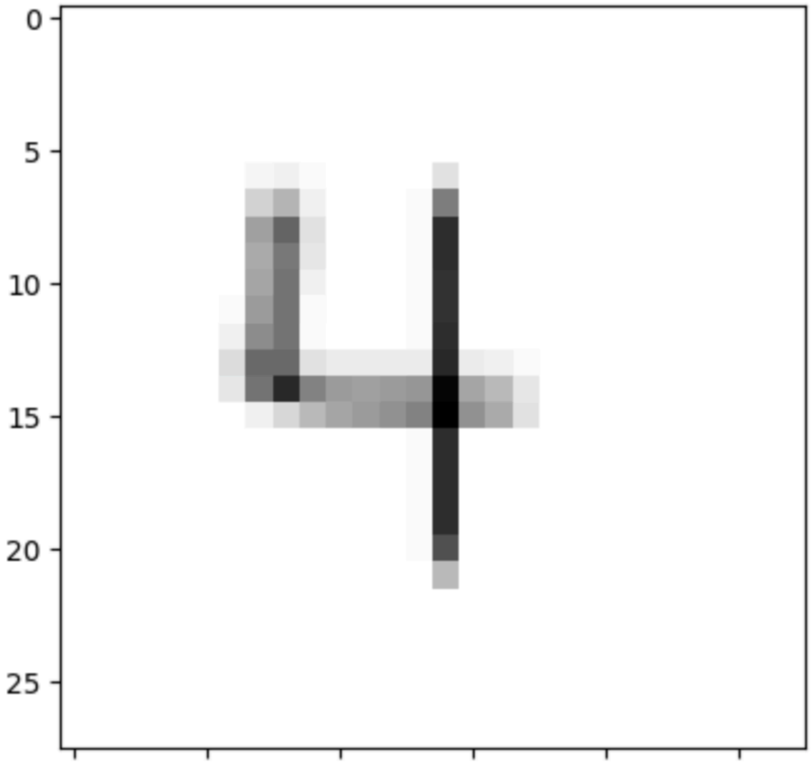


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Digit is most likely a 5
```

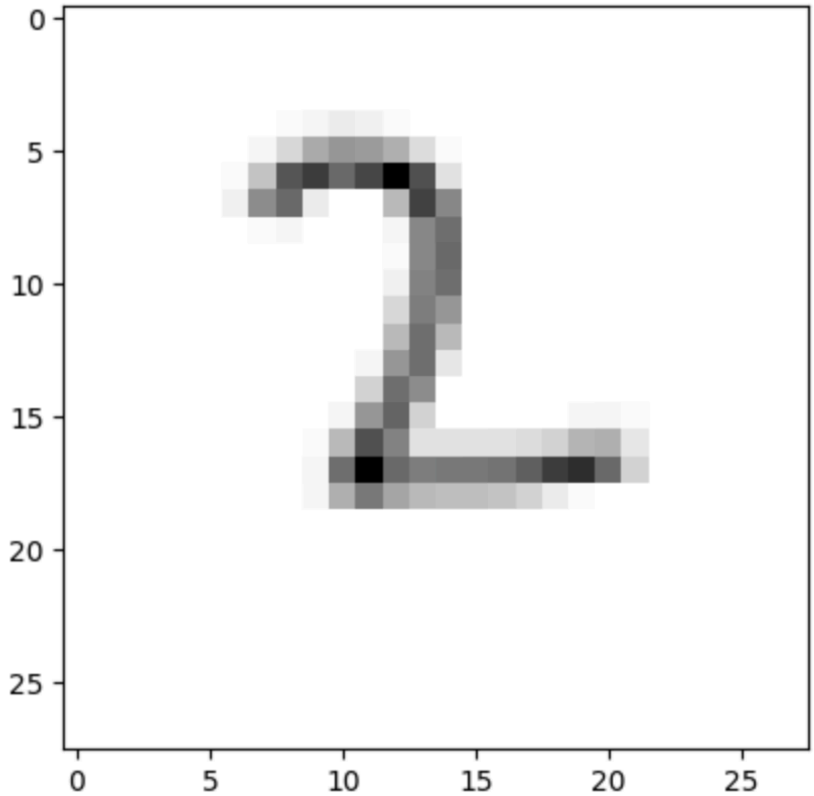


Correct predictions

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Digit is most likely a 1
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
Digit is most likely a 3
```



Incorrect Predictions
Total incorrect predictions = 4/10

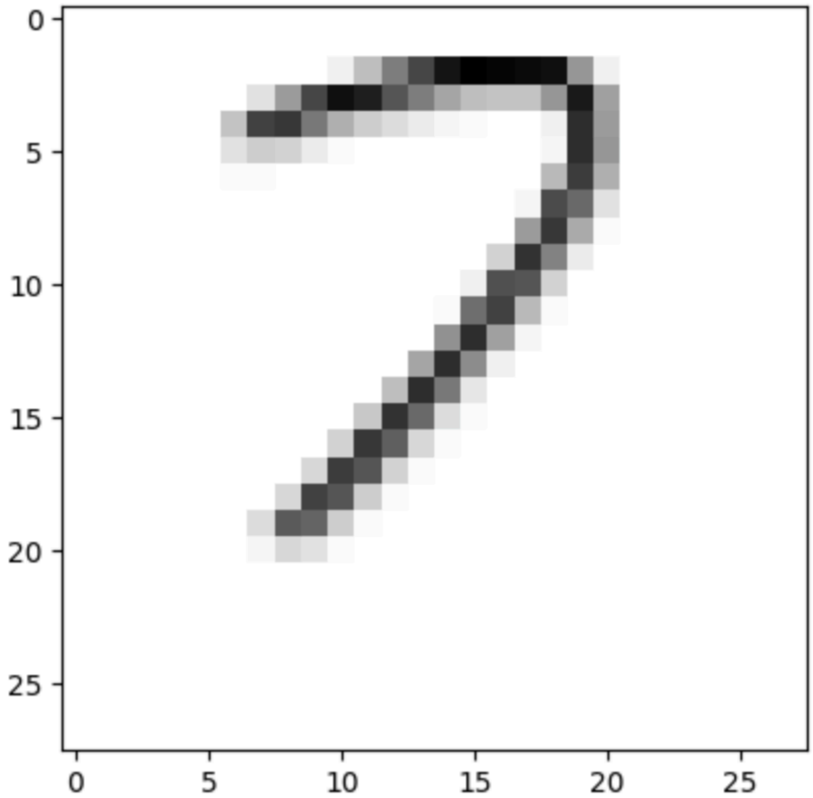
#Improving accuracy

model.fit(x_train,y_train,epochs=10)

Epochs increased to 10

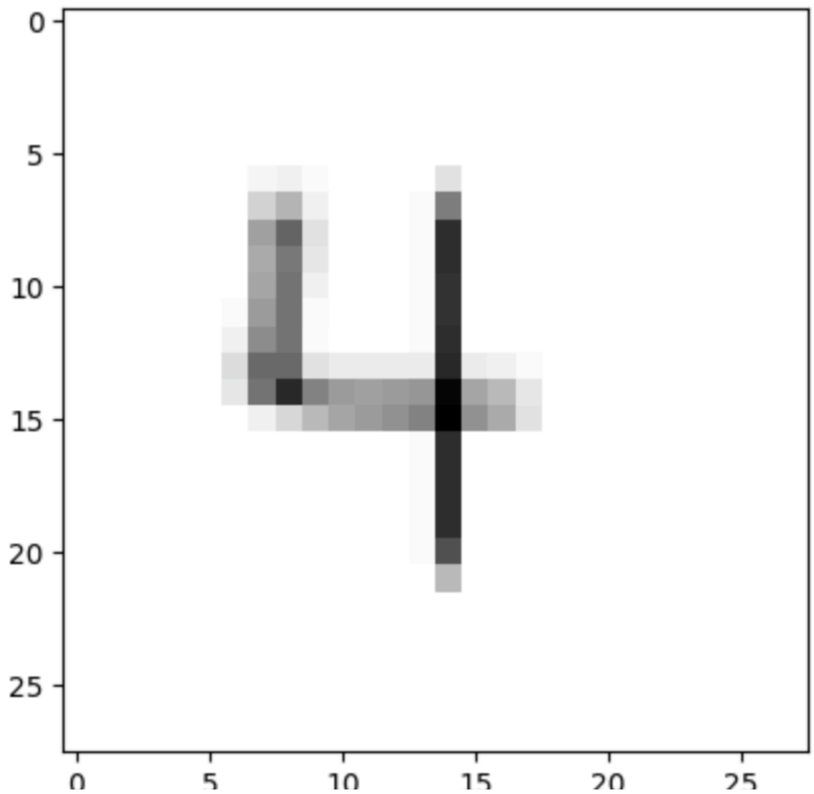
Before:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Digit is most likely a 2
```



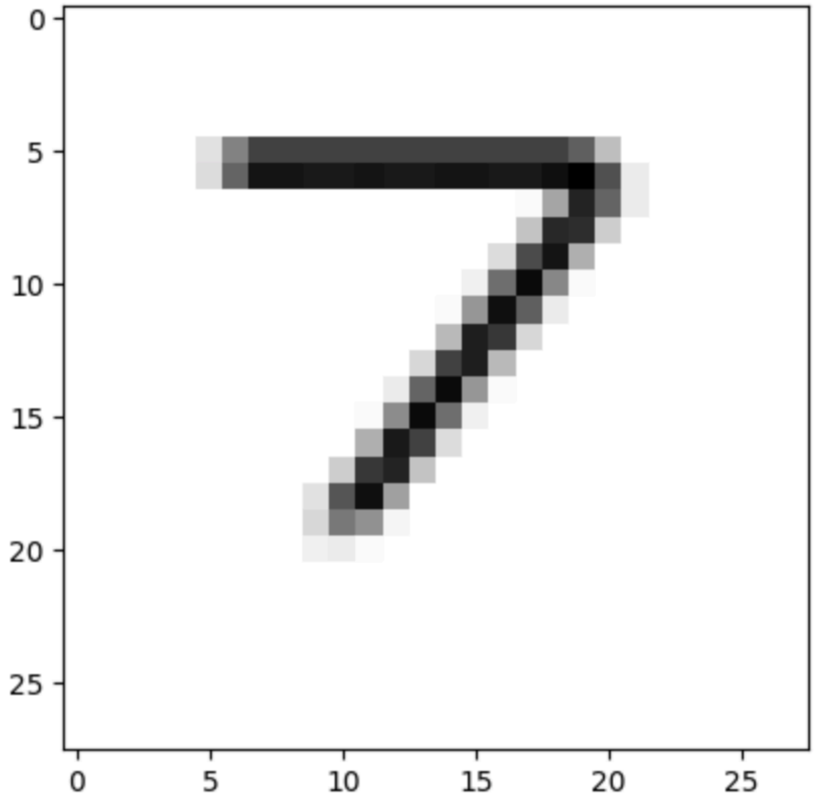

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Digit is most likely a 1
```

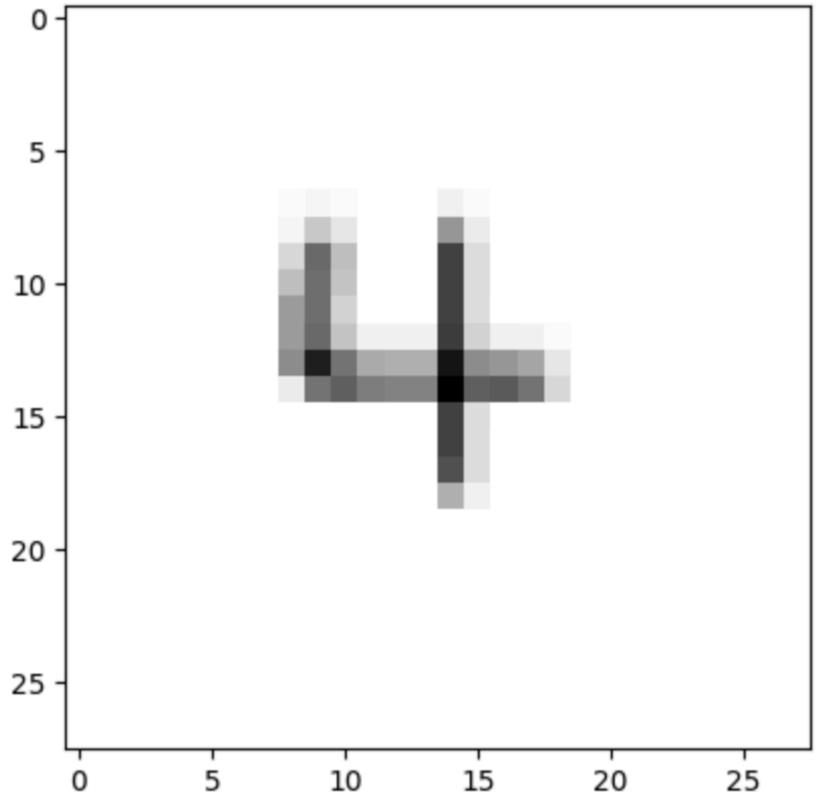


After:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Digit is most likely a 7
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
Digit is most likely a 4
```



#Adding Convolutional Layers

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

* while training the model again with the convolutional layers I realised that 10 Epochs might be overtraining the model.As convolutional neural networks are significantly more accurate with image comparisons I can afford to reduce the number of epochs for training to 4.

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

model.fit(x_train, y_train, epochs=4)
```

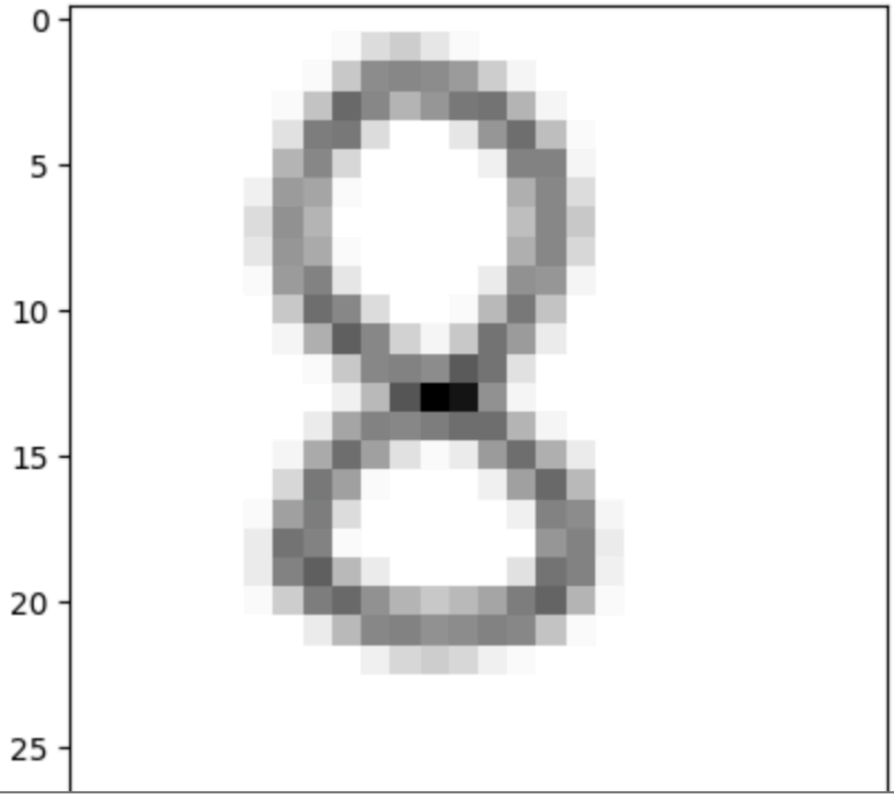
#Evaluation with Convolutional Layers

```python
model = tf.keras.models.load_model('handwritten_cnn.keras')
loss,accuracy = model.evaluate(x_test,y_test)
print(accuracy)
print(loss)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9876 - loss: 0.0365
0.9901999831199646
0.02873428724706173
```

#Running the CNN model with the same data

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Digit is most likely a 8
```



[ ] Start coding or generate with AI.

Total errors:1/10

Initial errors:4/10

#Final Code

```python
import os

import numpy as np

import tensorflow as tf


#rebuilding the model using a CNN

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()


x_train = x_train / 255.0
x_test = x_test / 255.0
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)


model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test Accuracy:", test_acc)

model.save('handwritten_cnn.keras')



#testing
model = tf.keras.models.load_model('handwritten.keras')
loss,accuracy = model.evaluate(x_test,y_test)
print(accuracy)
print(loss)

#using data and showing results
from genericpath import isfile
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
model = tf.keras.models.load_model('handwritten.keras')
ImgNum = 1

while os.path.isfile(f"Images/Image{ImgNum}.png"):
  try:
    inputIMG = cv2.imread(f"Images/Image{ImgNum}.png")[:,:,0]
    inputIMG = np.invert(np.array([inputIMG]))
    result = model.predict(inputIMG)
    print(f"Digit is most likely a {np.argmax(result)}")
    plt.imshow(inputIMG[0],cmap=plt.cm.binary)
    plt.show()
  except:
    print("ERROR!")
  finally:
    ImgNum += 1
```

Conclusion:

By using a CNN which uses filters to better analyse grid-like inputs such as Images in this case allowed me to be able to reduce the number of epochs therefore computational cost while training my model while simultaneously becoming more accurate as that was the initial issue with the Dense model.