In [1]:

```
import tensorflow
from tensorflow import keras
import matplotlib.pyplot as plt
```

```
/Users/lufy/Developer/projects/digit-recognizer/venv/lib/python3.9/site-pa
ckages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports
OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.
3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

In [2]:

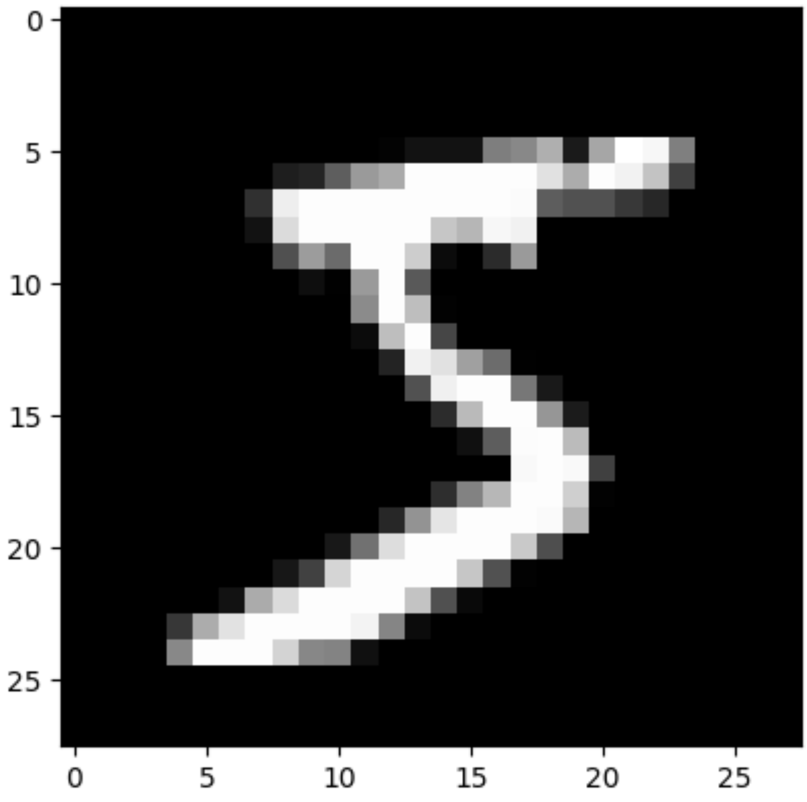
```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

In [ ]:

```
# Training image
plt.imshow(x_train[0], cmap="gray")
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x1586a7490>
```



In [4]:

```
# Pixel value harulai [0, 1] ko range ma lyaune (normalization)
x_train = x_train / 255
x_test = x_test / 255
```

In [5]:

```
# Create keras sequential model
model = keras.Sequential()

# First layer of the neural network, this is where the image data goes.
# The image is 28*28 pixels, so we need to flatten the image into 784 val
# 784 nodes hune vayo yo layer ma
model.add(keras.layers.Flatten(input_shape=(28,28)))

# Second layer of the neural network, this layer is hidden and contains 1
model.add(keras.layers.Dense(128, activation="relu"))

# Last layer containing 10 nodes. 10 ota number 0, 1, 2, .. 9 vako le
# 10 ota final nodes
# classification ko lagi "softmax" use garne
model.add(keras.layers.Dense(10, activation="softmax"))
```

```
/Users/lufy/Developer/projects/digit-recognizer/venv/lib/python3.9/site-pa
ckages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass
an `input_shape`/`input_dim` argument to a layer. When using Sequential mo
dels, prefer using an `Input(shape)` object as the first layer in the mode
l instead.
  super().__init__(**kwargs)
```

In [6]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	P
flatten (Flatten)	(None, 784)	
dense (Dense)	(None, 128)	1
dense_1 (Dense)	(None, 10)	

Total params: 101,770 (397.54 KB)

Trainable params: 101,770 (397.54 KB)

Non-trainable params: 0 (0.00 B)

In [7]:

```
# Model lai compile hanne, ani train garna milxa
model.compile(loss=keras.losses.SparseCategoricalCrossentropy, optimizer=
```

In [8]:

```
# Model lai train garne, epochs = num of times to train
# validation split = 0.2 vaneko purai data ma 20% chai
# validation garna ko lagi chuttyayeko
history = model.fit(x_train, y_train, epochs=25, validation_split=0.2)
```

```
Epoch 1/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8647 - loss: 0.48
12 - val_accuracy: 0.9542 - val_loss: 0.1628
Epoch 2/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9610 - loss: 0.13
19 - val_accuracy: 0.9650 - val_loss: 0.1163
Epoch 3/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9737 - loss: 0.08
96 - val_accuracy: 0.9704 - val_loss: 0.0989
Epoch 4/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9817 - loss: 0.06
36 - val_accuracy: 0.9722 - val_loss: 0.0974
Epoch 5/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9850 - loss: 0.04
95 - val_accuracy: 0.9744 - val_loss: 0.0904
Epoch 6/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9894 - loss: 0.03
88 - val_accuracy: 0.9750 - val_loss: 0.0894
Epoch 7/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9919 - loss: 0.02
86 - val_accuracy: 0.9751 - val_loss: 0.0863
Epoch 8/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9929 - loss: 0.02
39 - val_accuracy: 0.9745 - val_loss: 0.0932
Epoch 9/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9954 - loss: 0.01
67 - val_accuracy: 0.9700 - val_loss: 0.1135
Epoch 10/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9955 - loss: 0.01
49 - val_accuracy: 0.9749 - val_loss: 0.1052
Epoch 11/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9964 - loss: 0.01
21 - val_accuracy: 0.9757 - val_loss: 0.0940
Epoch 12/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9977 - loss: 0.00
97 - val_accuracy: 0.9765 - val_loss: 0.1024
Epoch 13/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9973 - loss: 0.00
94 - val_accuracy: 0.9759 - val_loss: 0.1074
Epoch 14/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9982 - loss: 0.00
73 - val_accuracy: 0.9762 - val_loss: 0.1094
Epoch 15/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9983 - loss: 0.00
70 - val_accuracy: 0.9729 - val_loss: 0.1184
Epoch 16/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9980 - loss: 0.00
71 - val_accuracy: 0.9758 - val_loss: 0.1172
Epoch 17/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9983 - loss: 0.00
60 - val_accuracy: 0.9747 - val_loss: 0.1197
Epoch 18/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9980 - loss: 0.00
67 - val_accuracy: 0.9762 - val_loss: 0.1119
Epoch 19/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9986 - loss: 0.00
50 - val_accuracy: 0.9777 - val_loss: 0.1146
Epoch 20/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9995 - loss: 0.00
28 - val_accuracy: 0.9725 - val_loss: 0.1460
```

```
Epoch 21/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9991 - loss: 0.00
41 - val_accuracy: 0.9774 - val_loss: 0.1230
Epoch 22/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9994 - loss: 0.00
27 - val_accuracy: 0.9746 - val_loss: 0.1394
Epoch 23/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9978 - loss: 0.00
61 - val_accuracy: 0.9731 - val_loss: 0.1476
Epoch 24/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9985 - loss: 0.00
47 - val_accuracy: 0.9753 - val_loss: 0.1432
Epoch 25/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9996 - loss: 0.00
19 - val_accuracy: 0.9725 - val_loss: 0.1574
```

In [9]:

```python
# Predict the test inputs
y_probabilities = model.predict(x_test)
y_predictions = y_probabilities.argmax(axis=1)
print("test y values:", y_test)
print("test y predictions:", y_predictions)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 408us/step
test y values: [7 2 1 ... 4 5 6]
test y predictions: [7 2 1 ... 4 5 6]
```

In [10]:

```python
# Aba accuracy mesure garne model ko
# accuray improve garna layer badauna milyo, epoch value ajai dherai haln
# tara overfitting huna sakxa
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_predictions)
```

Out[10]:

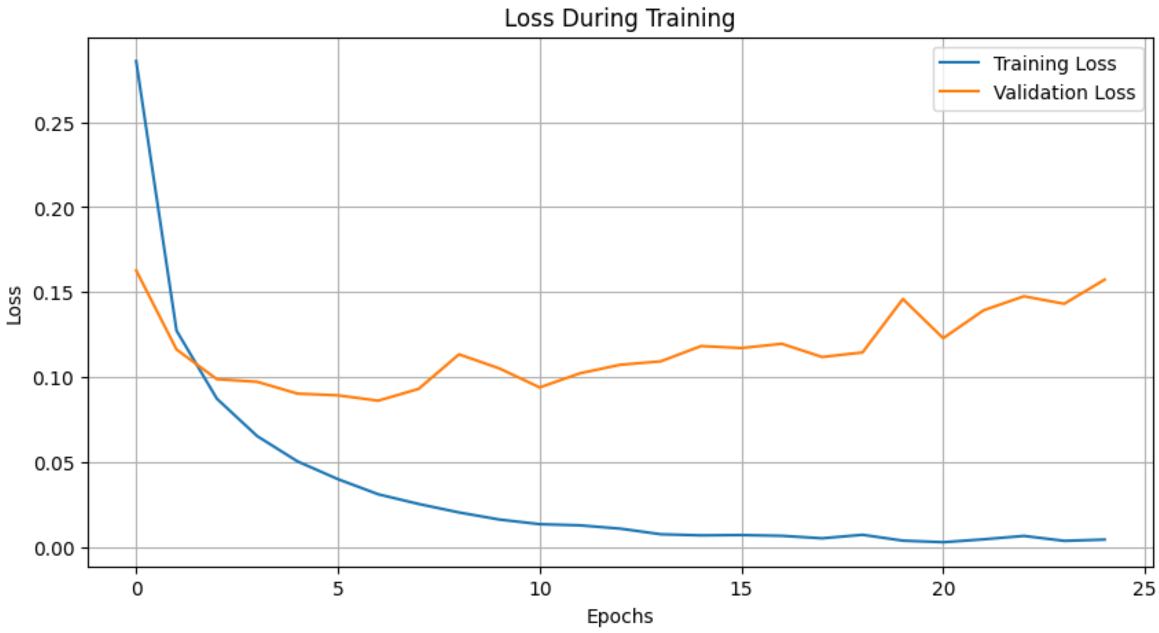
```
0.9748
```

In [18]:

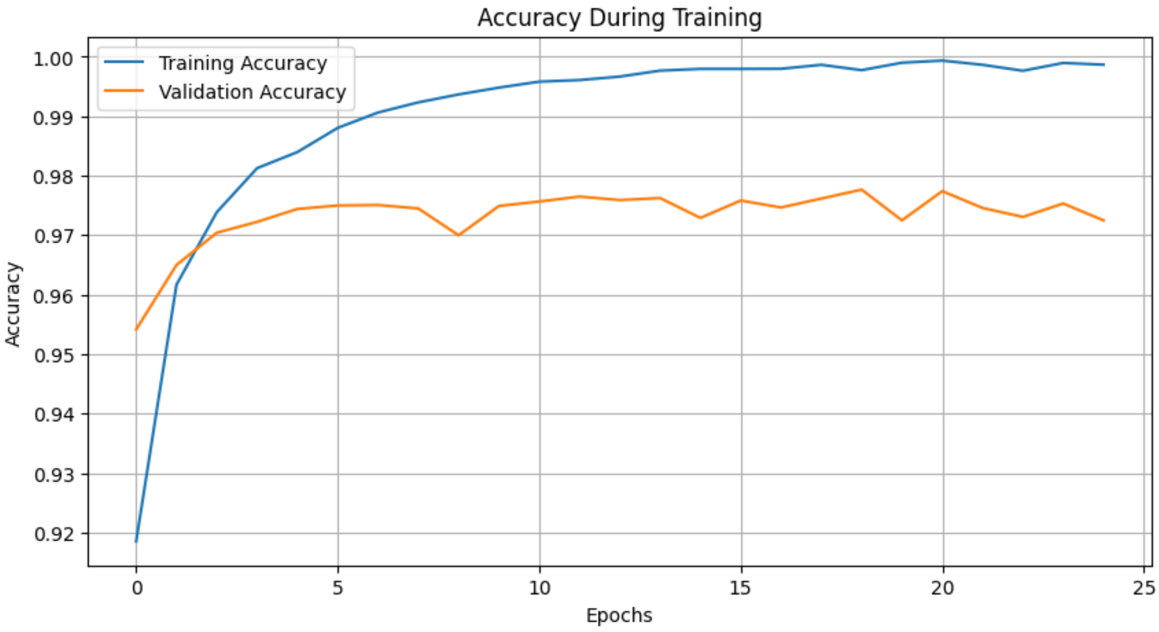
```python
# Anaylze the training process

# Plot training and validation loss
plt.figure(figsize=(10, 5)) # figure wide banauxa
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("Loss During Training")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()

# Plot training and validation accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.title("Accuracy During Training")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



Loss During Training



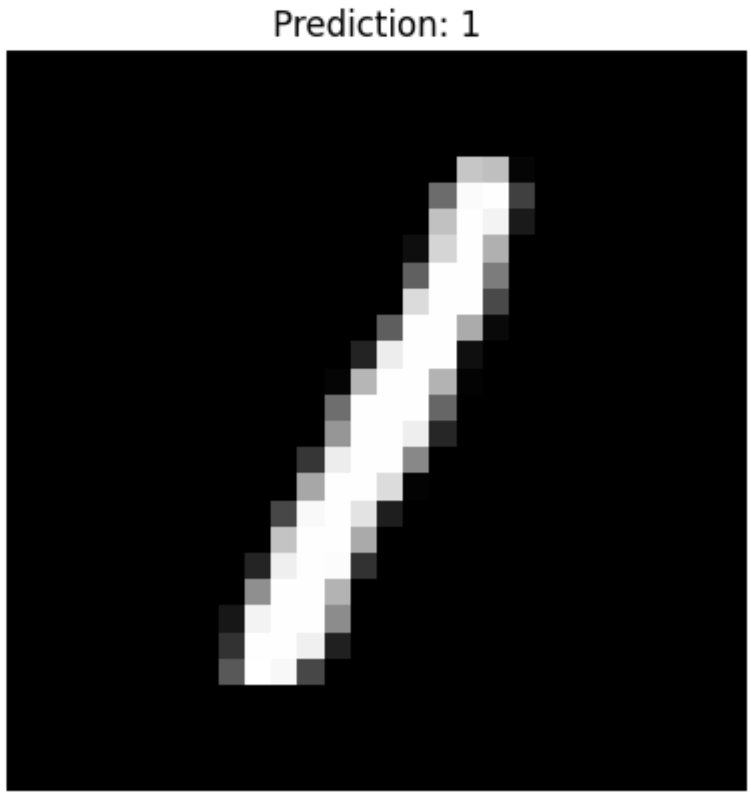
Accuracy During Training

In [ ]:

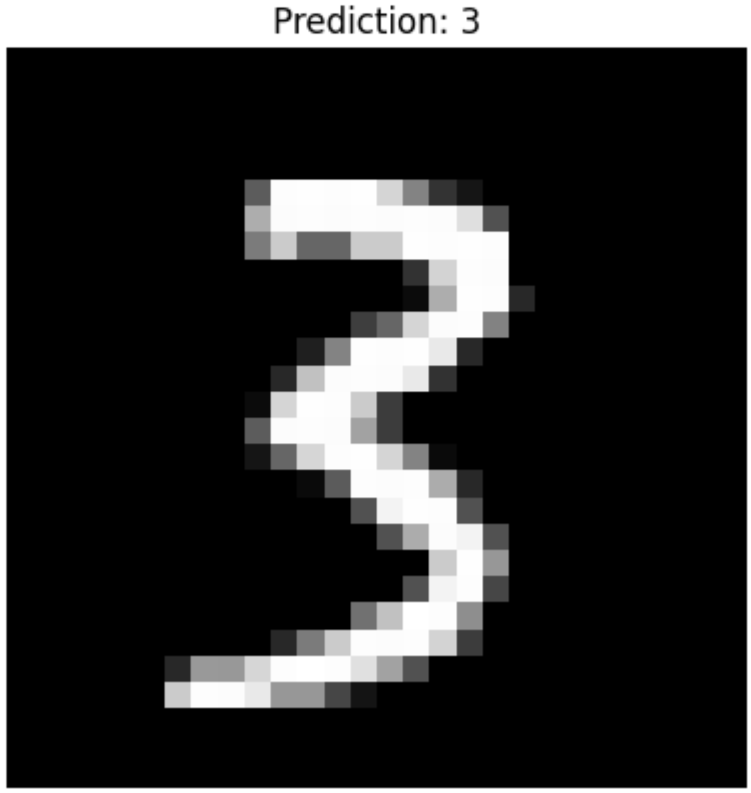
```
# Test our model's output with 10 random test inputs
from random import randrange

for i in range(10):
    input_image = x_test[randrange(len(x_test))]
    plt.imshow(input_image, cmap="gray")
    probabilities = model.predict(input_image.reshape(1, 28, 28))
    prediction = probabilities.argmax(axis=1)[0] # one with highest proba
    plt.title(f"Prediction: {prediction}")
    plt.axis('off')
    plt.show()
```

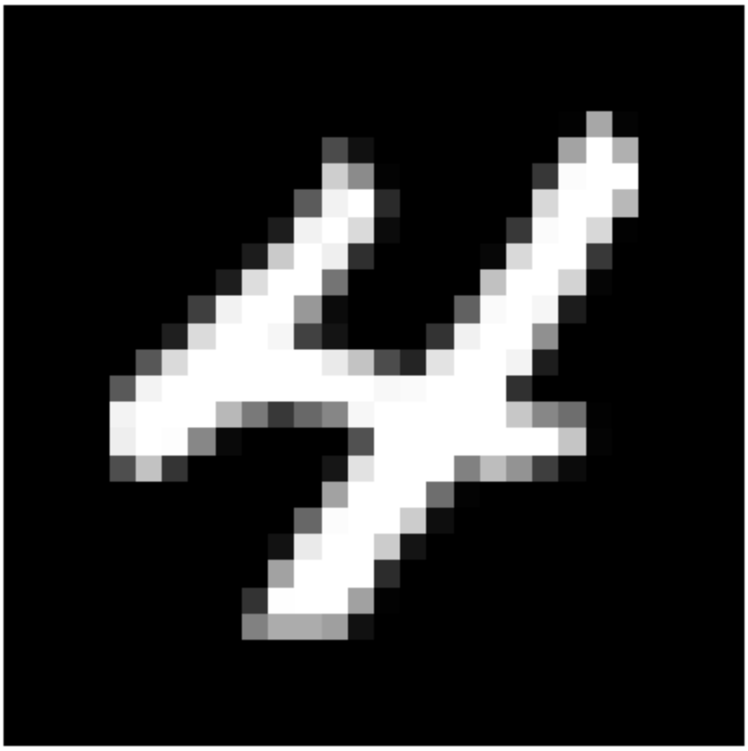
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

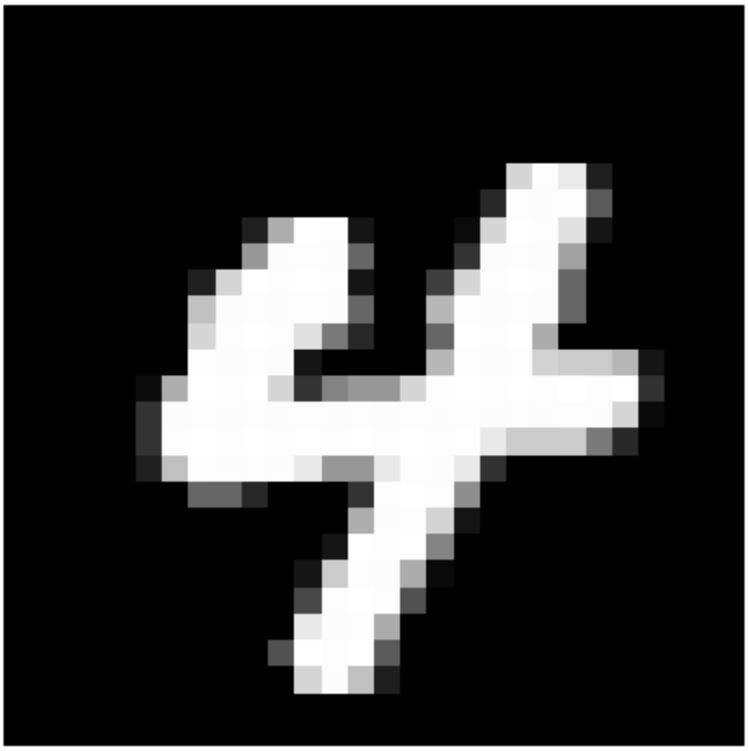


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```



Prediction: 4

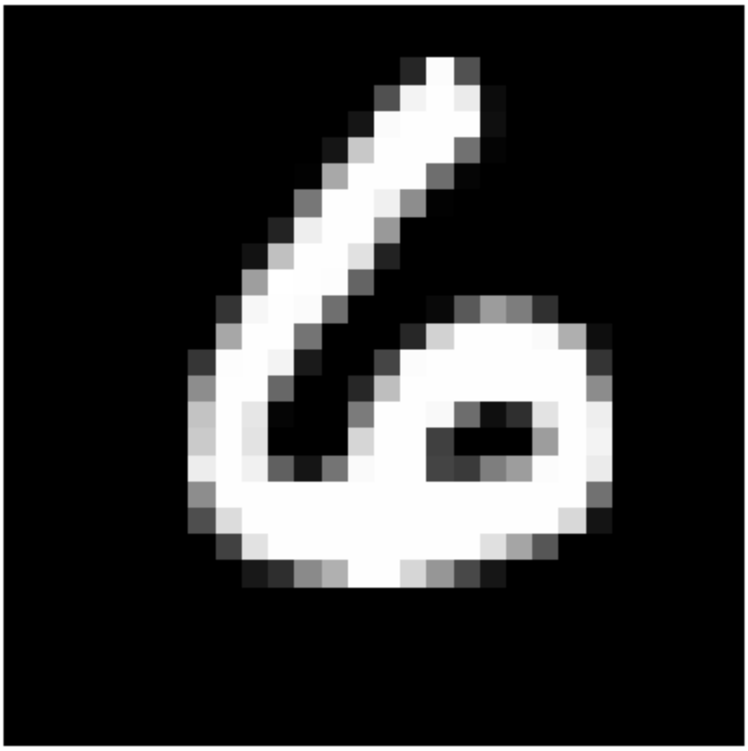
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```



Prediction: 4

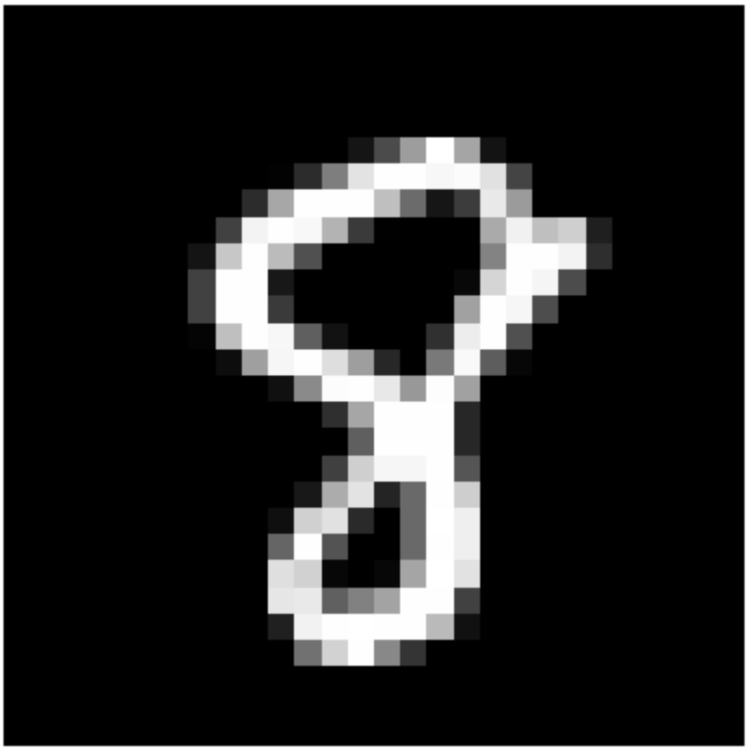
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

Prediction: 6

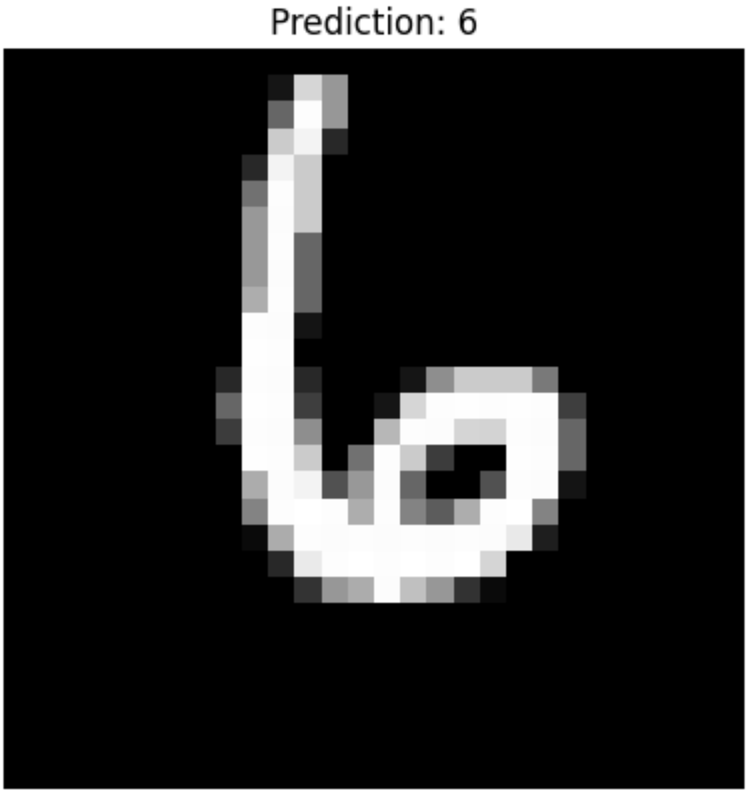


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

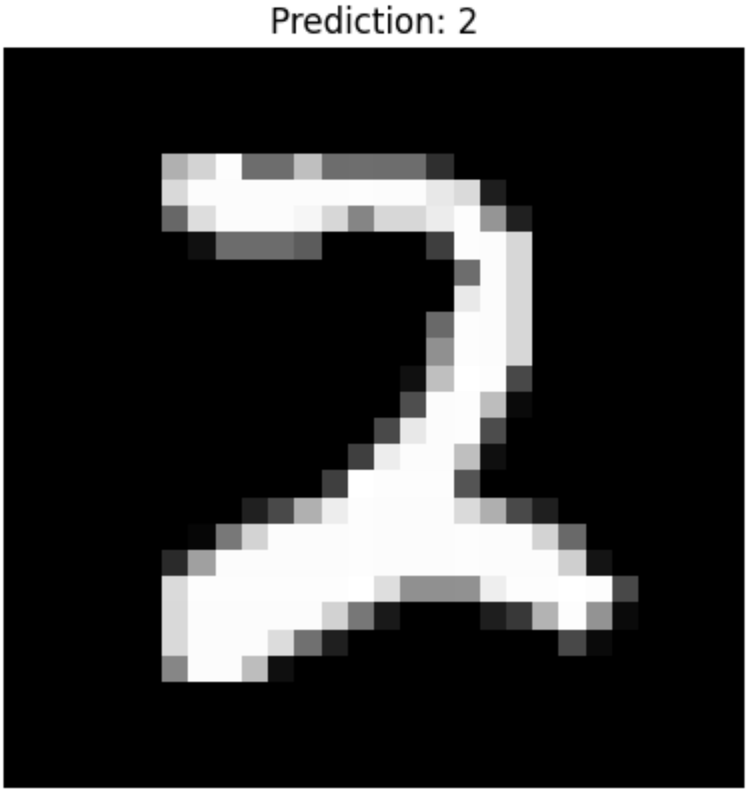
Prediction: 8



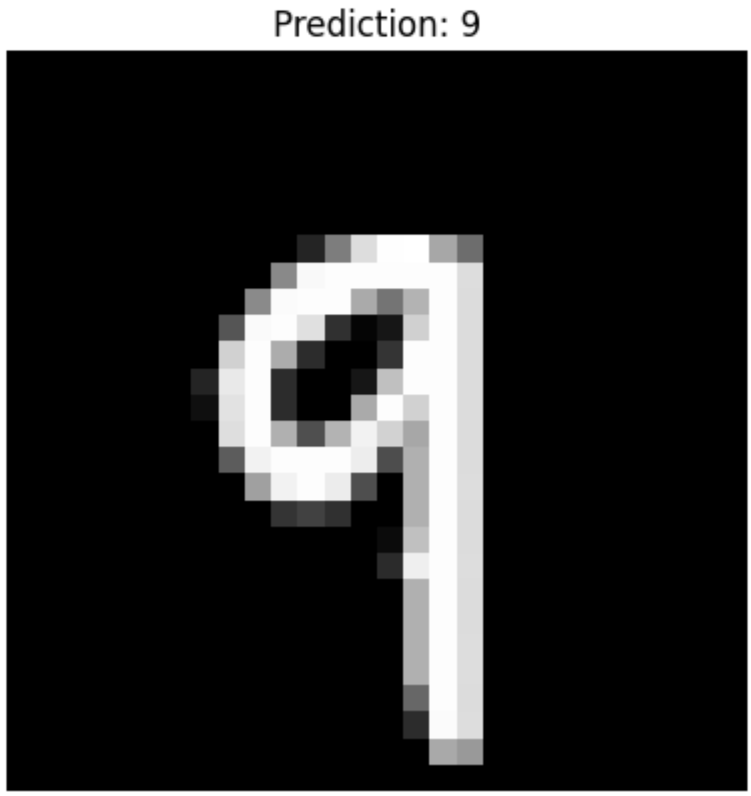
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```



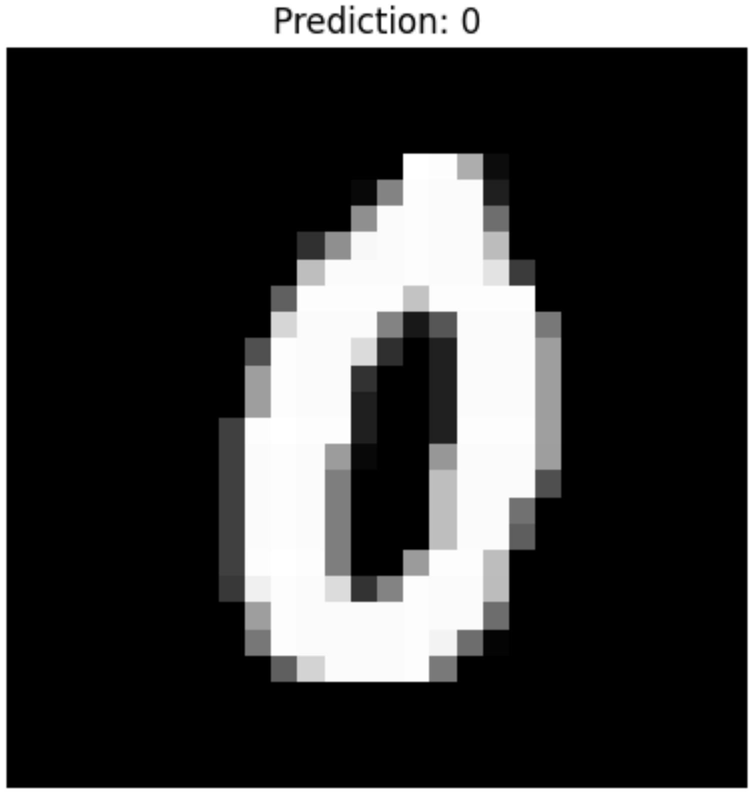
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```


Prediction: 0