In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import mnist
```

In [ ]:
```
(X_train,y_train), (X_test,y_test)= mnist.load_data()
```

In [ ]:
```
X_train= X_train/255
X_test= X_test/255
```

In [ ]:
```
model= Sequential([Conv2D(32,(3,3),input_shape= (28,28,1),activation='relu'),
                   MaxPool2D(pool_size=(2,2),padding='same'),
                   Conv2D(32,(3,3),activation='relu'),
                   MaxPool2D(pool_size=(2,2),padding='same'),
                   Flatten(),
                   Dense(1152, activation='relu'),
                   Dense(64, activation='relu'),
                   Dense(32, activation='relu'),
                   Dense(10, activation='softmax')])
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_22 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_22 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 conv2d_23 (Conv2D)          (None, 11, 11, 32)        9248

 max_pooling2d_23 (MaxPooli  (None, 6, 6, 32)          0
 ng2D)

 flatten_10 (Flatten)        (None, 1152)              0

 dense_34 (Dense)            (None, 1152)              1328256

 dense_35 (Dense)            (None, 64)                73792

 dense_36 (Dense)            (None, 32)                2080

 dense_37 (Dense)            (None, 10)                330

=================================================================
Total params: 1414026 (5.39 MB)
Trainable params: 1414026 (5.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

In [ ]:
```
history= model.fit(X_train, y_train,epochs=10,verbose='auto',validation_split=0.2,use_multiprocessing=True)
```

```
Epoch 1/10
1500/1500 [==============================] - 42s 27ms/step - loss: 0.1529 - accuracy: 0.9531 - val_loss: 0.0682 - val_accuracy:
0.9794
Epoch 2/10
1500/1500 [==============================] - 44s 29ms/step - loss: 0.0478 - accuracy: 0.9858 - val_loss: 0.0442 - val_accuracy:
0.9872
Epoch 3/10
1500/1500 [==============================] - 43s 29ms/step - loss: 0.0344 - accuracy: 0.9892 - val_loss: 0.0512 - val_accuracy:
0.9849
Epoch 4/10
1500/1500 [==============================] - 43s 29ms/step - loss: 0.0240 - accuracy: 0.9924 - val_loss: 0.0404 - val_accuracy:
0.9896
Epoch 5/10
1500/1500 [==============================] - 43s 29ms/step - loss: 0.0205 - accuracy: 0.9937 - val_loss: 0.0405 - val_accuracy:
0.9901
Epoch 6/10
1500/1500 [==============================] - 44s 29ms/step - loss: 0.0171 - accuracy: 0.9949 - val_loss: 0.0477 - val_accuracy:
0.9890
Epoch 7/10
1500/1500 [==============================] - 44s 29ms/step - loss: 0.0133 - accuracy: 0.9962 - val_loss: 0.0383 - val_accuracy:
0.9912
Epoch 8/10
1500/1500 [==============================] - 43s 28ms/step - loss: 0.0118 - accuracy: 0.9962 - val_loss: 0.0484 - val_accuracy:
0.9893
Epoch 9/10
1500/1500 [==============================] - 44s 29ms/step - loss: 0.0115 - accuracy: 0.9966 - val_loss: 0.0460 - val_accuracy:
0.9899
Epoch 10/10
1500/1500 [==============================] - 44s 29ms/step - loss: 0.0078 - accuracy: 0.9975 - val_loss: 0.0422 - val_accuracy:
0.9919
```

In [ ]:
```
Loss, Accuracy =model.evaluate(X_test, y_test)
print("Test Loss:", Loss)
print("Test Accuracy:", Accuracy)
```

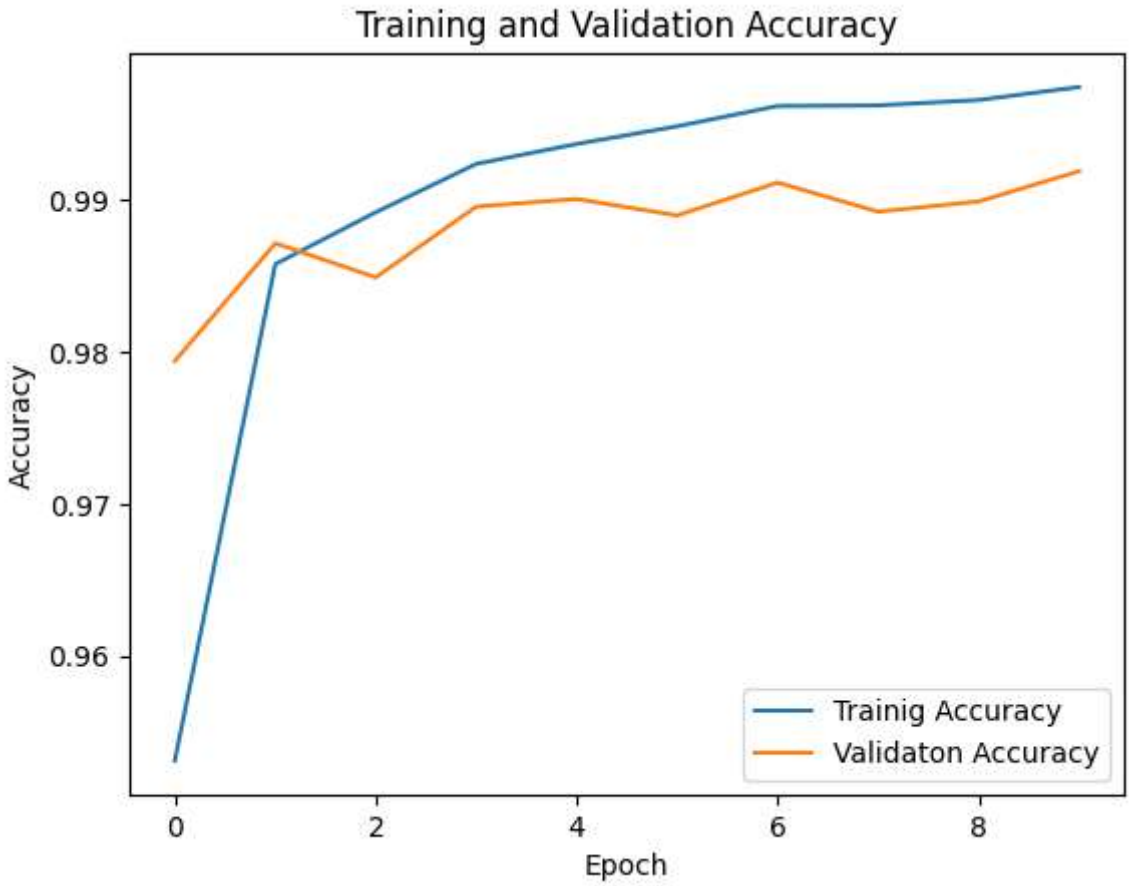
```
313/313 [==============================] - 2s 6ms/step - loss: 0.0362 - accuracy: 0.9911
Test Loss: 0.03615691885352135
Test Accuracy: 0.991100013256073
```

In [ ]:
```
plt.plot(history.history['accuracy'], label='Trainig Accuracy')
plt.plot(history.history['val_accuracy'],label='Validaton Accuracy')
```

```
# Adding labels and title
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')

# Adding a legend to the plot
plt.legend()

# Display the plot
plt.show()
```

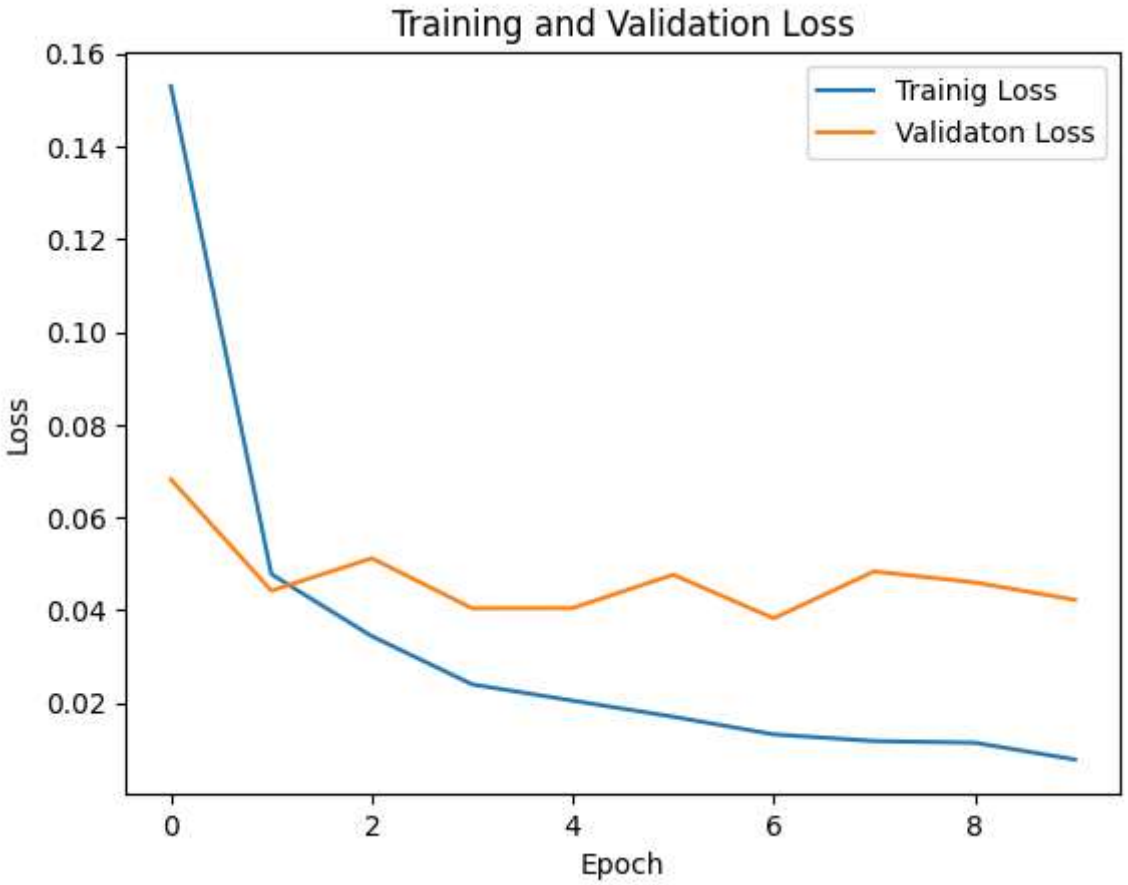



In [ ]:
```
plt.plot(history.history['loss'], label='Trainig Loss')
plt.plot(history.history['val_loss'],label='Validaton Loss')
```

```
# Adding labels and title
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')

# Adding a legend to the plot
plt.legend()

# Display the plot
plt.show()
```




In [ ]:
```
y_pred= model.predict(X_test)
```

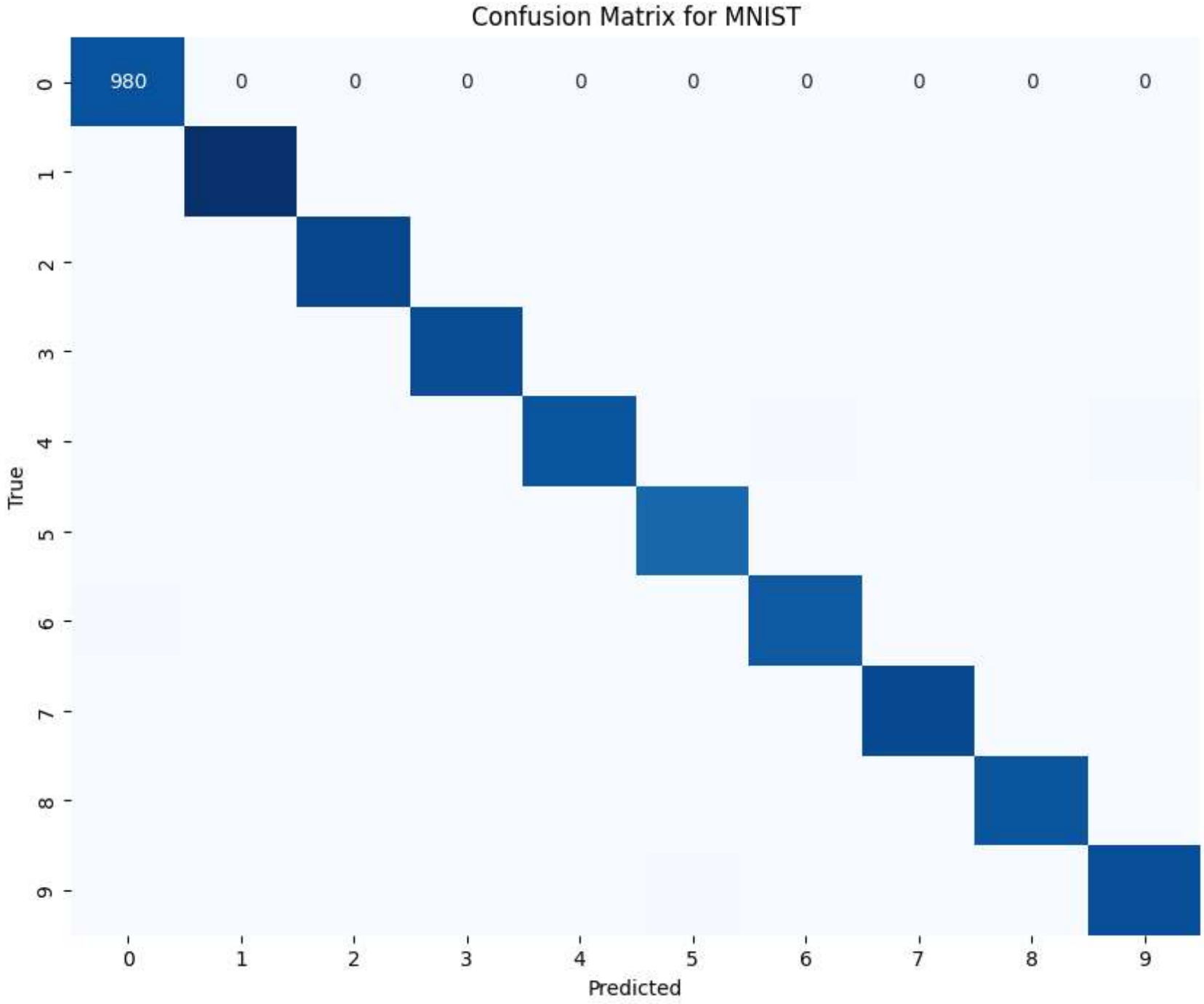
```
313/313 [==============================] - 2s 6ms/step
```

In [ ]:

```python
from sklearn.metrics import confusion_matrix
y_pred_classes = y_pred.argmax(axis=1)
y_test_classes = y_test


cm = confusion_matrix(y_test_classes, y_pred_classes)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=range(10), yticklabels=range(10))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for MNIST')
plt.show()
```


Confusion Matrix for MNIST
980
0
0
0
0
0
0
0
0
0
True
Predicted
0
1
2
3
4
5
6
7
8
9

Model Testing:

In [ ]:
```
plt.figure(figsize=(2,2))
plt.imshow(X_test[1],cmap='gray')
plt.axis('off')
plt.show()
predicted_label=y_pred[1].argmax()
print('Predicted Label:',predicted_label)
```

```
Predicted Label: 2
```

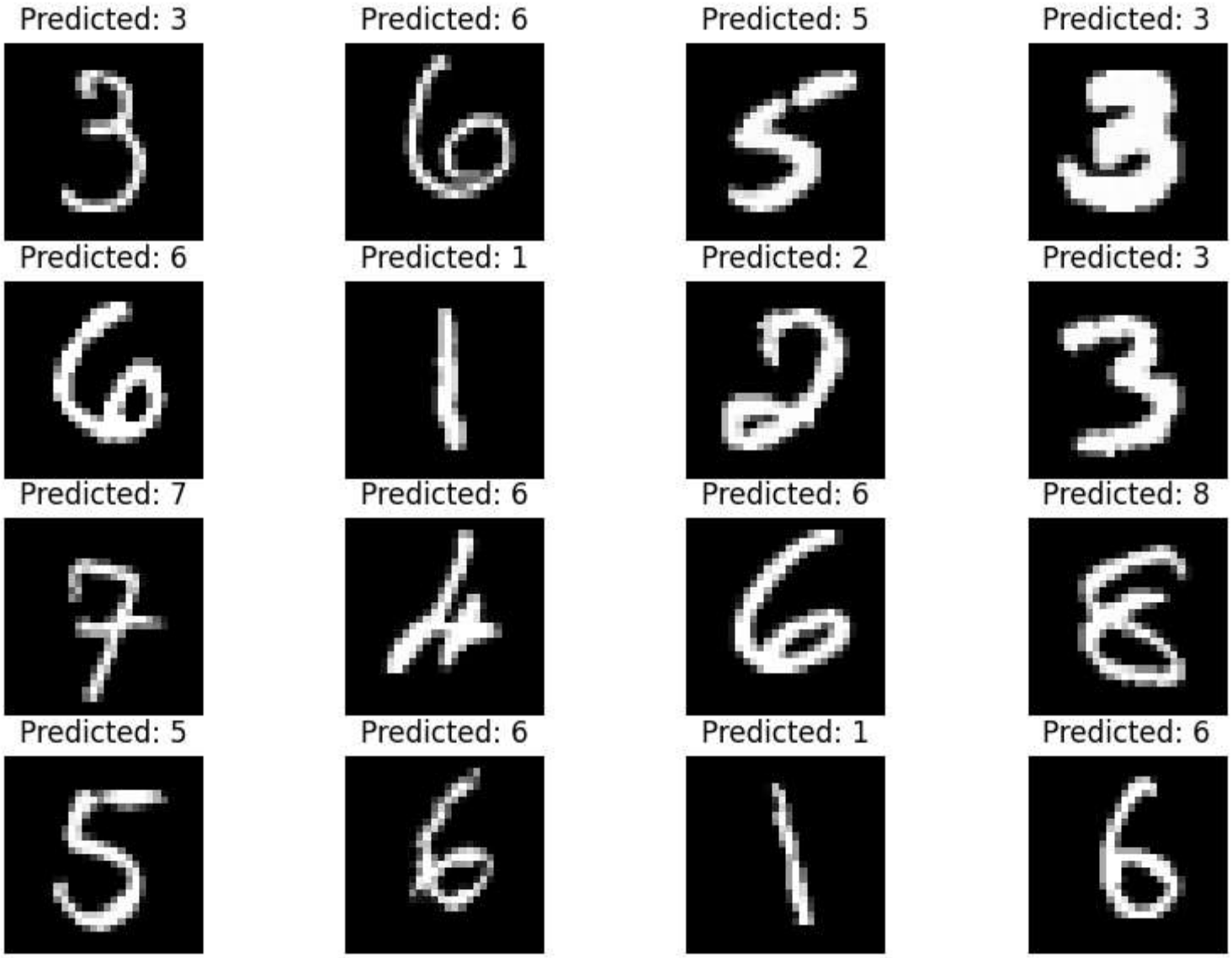
In [ ]:
```
predictions = model.predict(X_test)

image_indices = [1503, 100, 155,200,886,302,400,500,
                 550,247,384,1202,1521,2000,2504,3000]


plt.figure(figsize=(10, 7))
for i, index in enumerate(image_indices, 1):
    plt.subplot(4, 4, i)
    plt.imshow(X_test[index], cmap='gray')
    plt.axis('off')
    predicted_label = predictions[index].argmax()
    plt.title(f'Predicted: {predicted_label}')

plt.show()
```

```
313/313 [==============================] - 2s 6ms/step
```


Predicted: 3
Predicted: 6
Predicted: 5
Predicted: 3
Predicted: 6
Predicted: 1
Predicted: 2
Predicted: 3
Predicted: 7
Predicted: 6
Predicted: 6
Predicted: 8
Predicted: 5
Predicted: 6
Predicted: 1
Predicted: 6

In [ ]: