```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
```

```
train_data = pd.read_csv('/content/Train.csv')
print("Shape of train_data:", train_data.shape)
X = train_data.iloc[:, 1:]
y = train_data.iloc[:, 0]

print("Shape of X after separating features:", X.shape)
```

```
Shape of train_data: (42000, 785)
Shape of X after separating features: (42000, 784)
```

```
if not isinstance(X, pd.DataFrame):
    X = pd.DataFrame(X)
X = X.apply(pd.to_numeric, errors='coerce')
X = X.fillna(0)
X = X.values / 255.0
X = X.reshape(-1, 28, 28, 1)
print("Shape of X after reshaping:", X.shape)
```

```
Shape of X after reshaping: (42000, 28, 28, 1)
```

```
y = to_categorical(y, num_classes=10)
print("Shape of y after one-hot encoding:", y.shape)
```

```
Shape of y after one-hot encoding: (42000, 10)
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
print("X_train shape:", X_train.shape)
```

```
X_train shape: (33600, 28, 28, 1)
```

```
model = Sequential([
    Flatten(input_shape=(28, 28, 1)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_c
  super().__init__(**kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 10)	650

```
Total params: 109,386 (427.29 KB)
Trainable params: 109,386 (427.29 KB)
Non-trainable params: 0 (0.00 B)
```

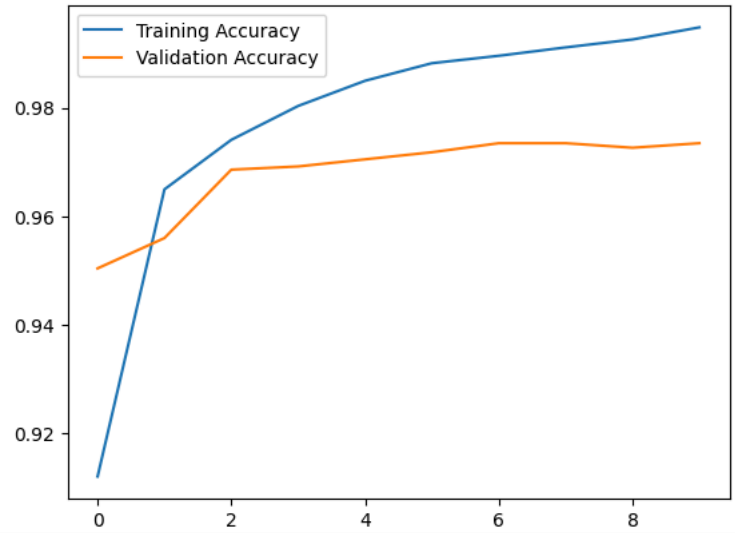
```
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
```

```
Epoch 1/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8412 - loss: 0.5485 - val_accuracy: 0.9504 - val_loss: 0.1684
Epoch 2/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9619 - loss: 0.1261 - val_accuracy: 0.9560 - val_loss: 0.1363
Epoch 3/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9743 - loss: 0.0834 - val_accuracy: 0.9686 - val_loss: 0.1077
Epoch 4/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9818 - loss: 0.0594 - val_accuracy: 0.9692 - val_loss: 0.1025
Epoch 5/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9855 - loss: 0.0462 - val_accuracy: 0.9705 - val_loss: 0.0982
```

```
Epoch 6/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9901 - loss: 0.0318 - val_accuracy: 0.9718 - val_loss: 0.1017
Epoch 7/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9909 - loss: 0.0271 - val_accuracy: 0.9735 - val_loss: 0.1032
Epoch 8/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9932 - loss: 0.0219 - val_accuracy: 0.9735 - val_loss: 0.1031
Epoch 9/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9925 - loss: 0.0196 - val_accuracy: 0.9726 - val_loss: 0.1158
Epoch 10/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9961 - loss: 0.0132 - val_accuracy: 0.9735 - val_loss: 0.1173
```

```
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()
```

```
263/263 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9739 - loss: 0.1290
Validation Accuracy: 97.35%
```

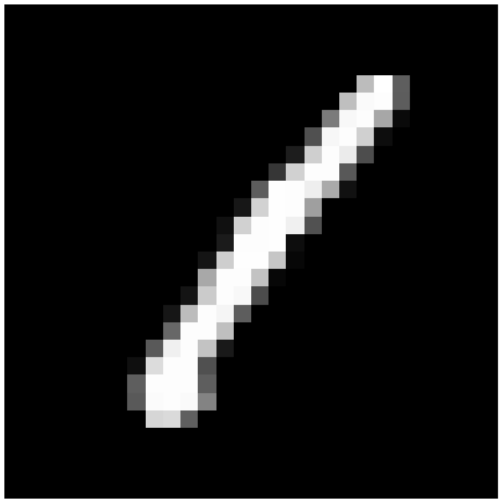


Default title text

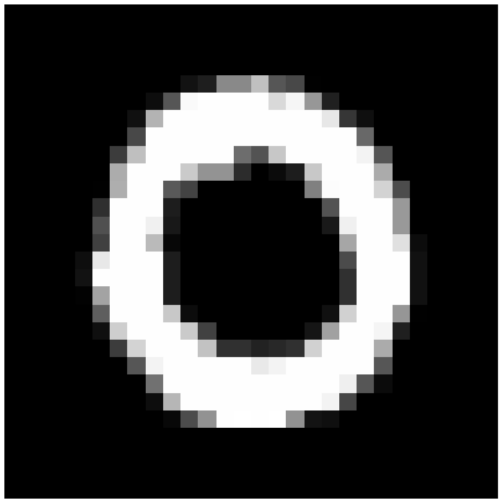
```
# @title Default title text
test_data = pd.read_csv('/content/Train.csv')
X_test = test_data.iloc[:, 1:].values  # Select feature columns (pixels)
X_test = X_test / 255.0  # Normalize pixel values before reshaping
X_test = X_test.reshape(-1, 28, 28, 1)  # Reshape for the model
predictions = model.predict(X_test)
predicted_labels = np.argmax(predictions, axis=1)
for i in range(5):
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Predicted: {predicted_labels[i]}")
    plt.axis('off')
    plt.show()
```

```
1313/1313 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
```

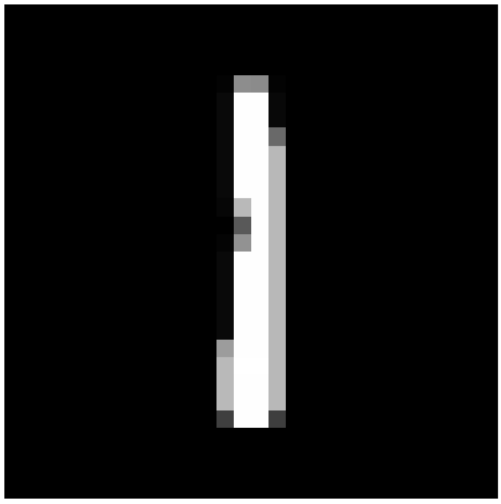
Predicted: 1



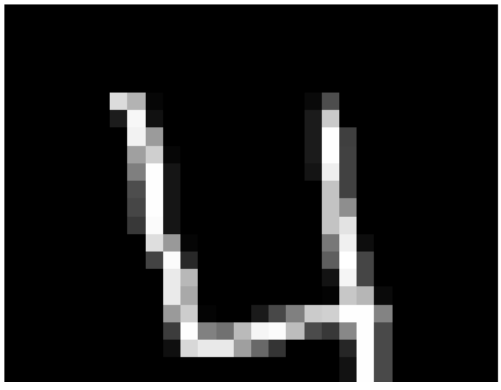
Predicted: 0



Predicted: 1



Predicted: 4



Predicted: 0