Handwritten Numbers Classification Model - Convolutional Neural Network

Import Libraries

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
from PIL import Image
import os
```

Load MNIST Data

In [5]:
```
# Download data set and separate into training and testing sets
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
```

In [6]:
```
# Display the first images in the training set
for i in range(5):
    # Create subplot for each image
    plt.subplot(1, 5, i + 1)

    # Display the image in a grayscale format
    plt.imshow(train_images[i], cmap='gray')

    # Set the title to the label of the image
    plt.title(f"Label: {train_labels[i]}")

    # Hide the axes
    plt.axis("off")
plt.show()
```

Create, Train, & Test Model

In [8]:
```
# Normalize the data
train_images = train_images / 255.0
test_images = test_images / 255.0
```

In [9]:
```
# Reshape the data to include channel dimension (needed for CNN)
train_images = train_images.reshape((-1, 28, 28, 1))
test_images = test_images.reshape((-1, 28, 28, 1))
```

In [10]:
```
# Build the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),

    # 10 classes for digits 0-9
    layers.Dense(10, activation='softmax')
])
```

```
C:\Users\emili\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserW
arning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential model
s, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [11]:
```
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [12]:
```
# Train the model
model.fit(train_images, train_labels, epochs=5, validation_split=0.1)
```

```
Epoch 1/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8941 - loss: 0.3522 - val_accuracy: 0.98
32 - val_loss: 0.0574
Epoch 2/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9827 - loss: 0.0538 - val_accuracy: 0.98
53 - val_loss: 0.0513
Epoch 3/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9881 - loss: 0.0360 - val_accuracy: 0.98
82 - val_loss: 0.0413
Epoch 4/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9908 - loss: 0.0271 - val_accuracy: 0.98
90 - val_loss: 0.0408
Epoch 5/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.9938 - loss: 0.0195 - val_accuracy: 0.9
905 - val_loss: 0.0377
```

Out[12]:
```
<keras.src.callbacks.history.History at 0x192bb0e6c50>
```

In [13]:
```
# Evaluate on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9876 - loss: 0.0436
Test accuracy: 0.9899
```

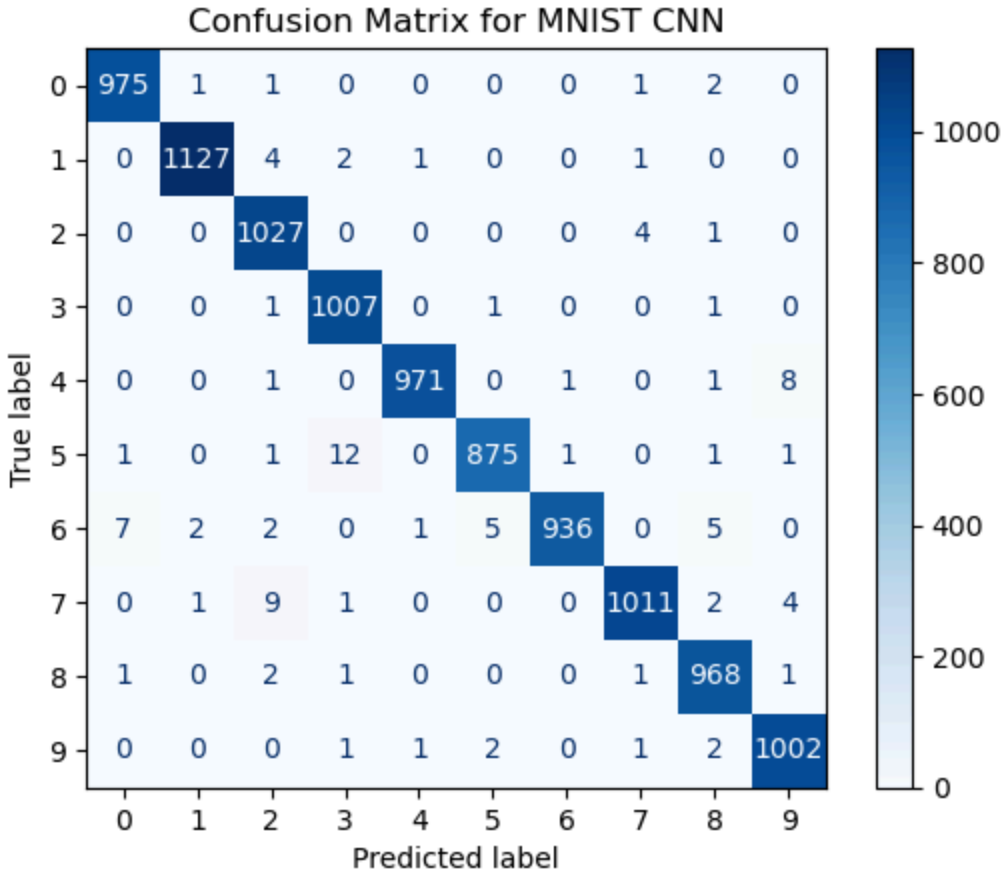
Visualize Model Accuracy

In [15]:
```
# Predict the test labels
y_pred_probs = model.predict(test_images)
y_pred = np.argmax(y_pred_probs, axis=1)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

In [16]:
```
# Create confusion matrix
cm = confusion_matrix(test_labels, y_pred)
```

In [17]:
```
# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.arange(10))
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix for MNIST CNN")
plt.show()
```



Test Model Using External Images

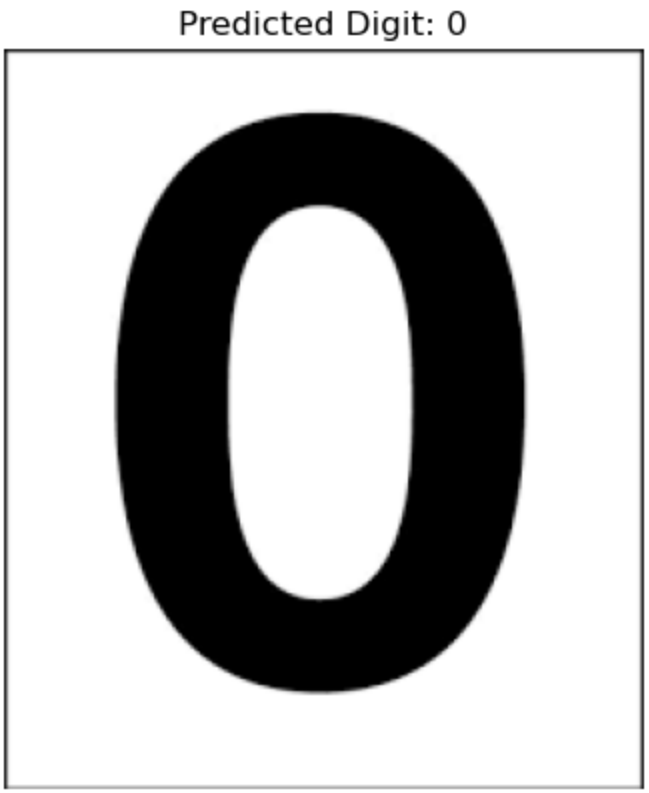
In [19]:
```
# Define the full image directory path
image_dir = r"C:\Users\emili\OneDrive\04 - GitHub Projects\Masters in Data Science\10 Handwritter
image_files = ['0.jpg', '1.jpg', '2.jpg', '6.jpg', '7.jpg', '8.jpg']
```

In [20]:
```
# Function to scale pictures
def preprocess_image(img_path):
    # Convert to grascale
    img = Image.open(img_path).convert("L")

    # Resize
    img = img.resize((28, 28))

    # Transform to numpy array
```

```
    img_array = np.array(img)

    # Invert for MNIST-style white-on-black
    img_array = 255 - img_array

    # Normalize data
    img_array = img_array.astype('float32') / 255

    # Reshape for CNN model
    img_array = img_array.reshape(1, 28, 28, 1)
    return img_array
```

In [21]:
```
# Loop through and predict each image
for file in image_files:
    # Joins safely the file path with file name that is being analyzed
    img_path = os.path.join(image_dir, file)

    # Uses the prepricess_image funtion to scale the image
    processed_img = preprocess_image(img_path)

    # Uses model to predict numeric value in picture
    prediction = model.predict(processed_img)

    # Obtain the predicted label
    predicted_label = np.argmax(prediction)

    # Display original image and prediction
    plt.imshow(Image.open(img_path), cmap='gray')
    plt.title(f"Predicted Digit: {predicted_label}")
    plt.axis('off')
    plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

Predicted Digit: 1

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

Predicted Digit: 2

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

Predicted Digit: 0

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

Predicted Digit: 7

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```

Predicted Digit: 3