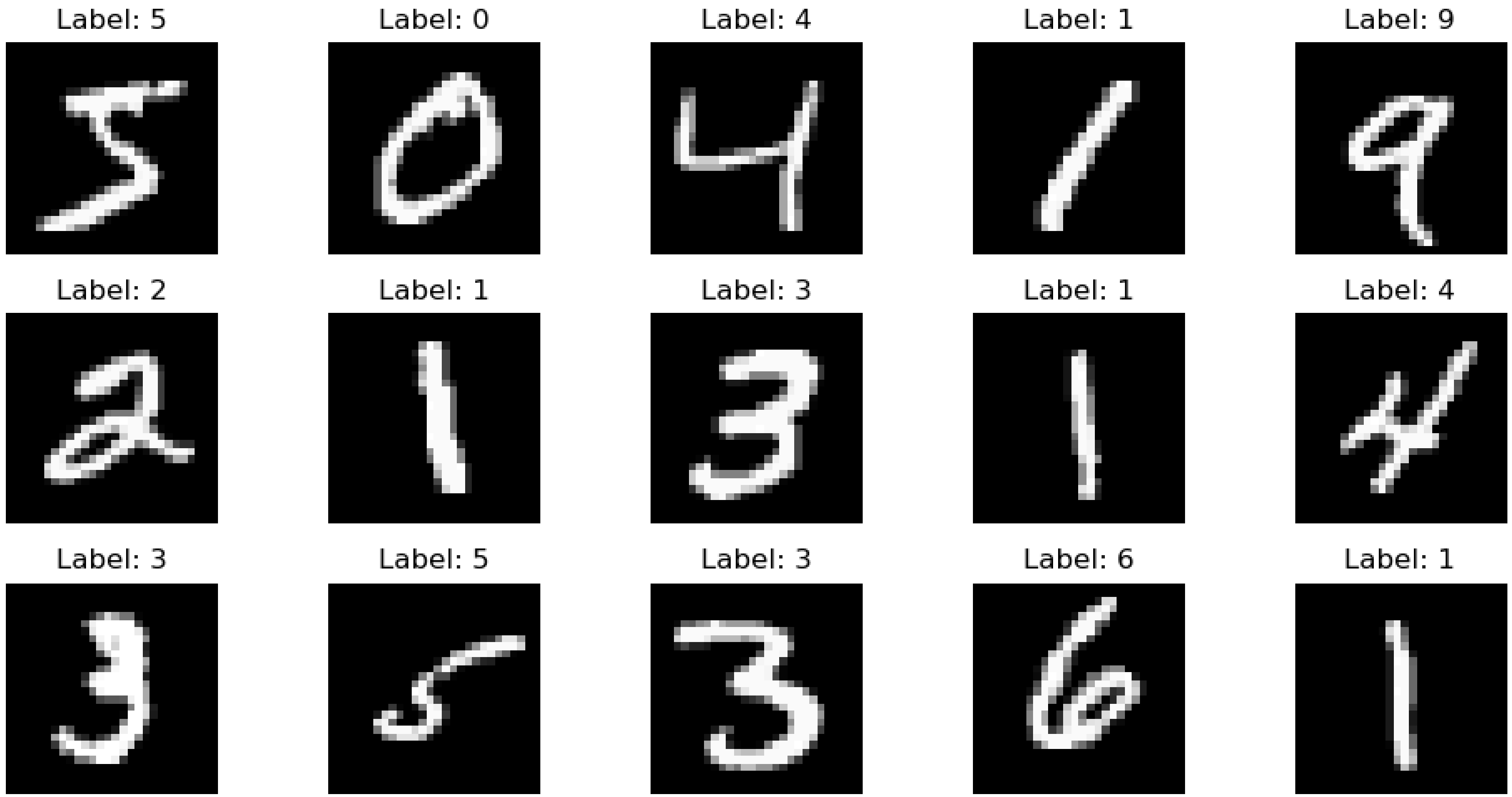
In [1]:
```
# Step 1: Import Required Libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam, SGD
import cv2
```

In [5]:
```
# Step 2: Load MNIST Data

# Load the dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Shape of dataset
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)

# Number of classes
print("Number of unique labels:", len(np.unique(y_train)))

# Visualize some sample digits
plt.figure(figsize=(10,5))
for i in range(15):
    plt.subplot(3,5,i+1)
    plt.imshow(X_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```
Training data shape: (60000, 28, 28)
Testing data shape: (10000, 28, 28)
Number of unique labels: 10
```



In [6]:
```
# Step 3A: Preprocess the Data

# Normalize pixel values (0–1 range)
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Reshape to add channel dimension (needed for CNN: 28x28x1)
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
```

```
y_test = to_categorical(y_test, 10)

print("After preprocessing:")
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

```
After preprocessing:
Training data shape: (60000, 28, 28, 1)
Testing data shape: (10000, 28, 28, 1)
```

In [7]:
```
# Step 3B: Build CNN Model (Function for neatness)

def build_cnn_model(optimizer='adam', learning_rate=0.001):
    if optimizer == 'adam':
        opt = Adam(learning_rate=learning_rate)
    elif optimizer == 'sgd':
        opt = SGD(learning_rate=learning_rate)

    model = Sequential([
        Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
        MaxPooling2D((2,2)),
        Conv2D(64, (3,3), activation='relu'),
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])

    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

In [8]:
```
## Train CNN Model + Plot Accuracy/Loss Graphs
```

In [9]:
```
# Step 4A: Train the CNN Model

# Build model with Adam optimizer
model_adam = build_cnn_model(optimizer='adam', learning_rate=0.001)

# Train the model
history_adam = model_adam.fit(
    X_train, y_train,
    validation_split=0.2,
```

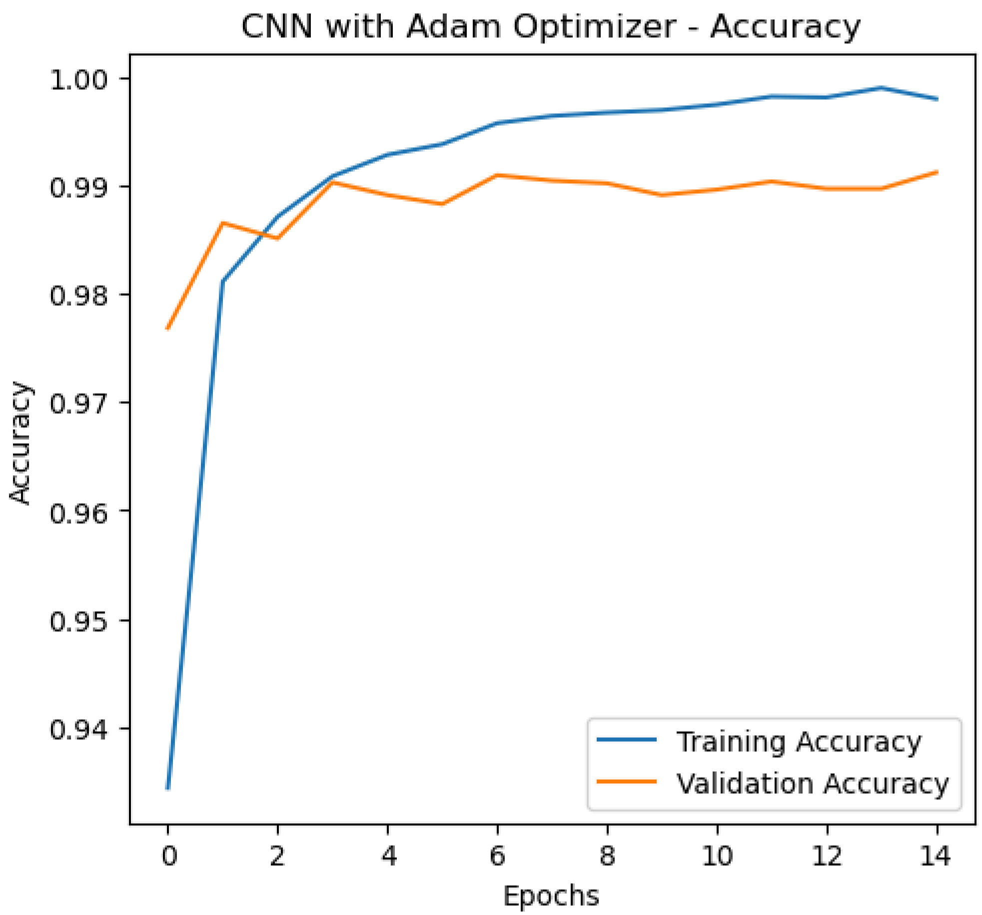
```
    epochs=15,
    batch_size=100,
    verbose=2
)
```

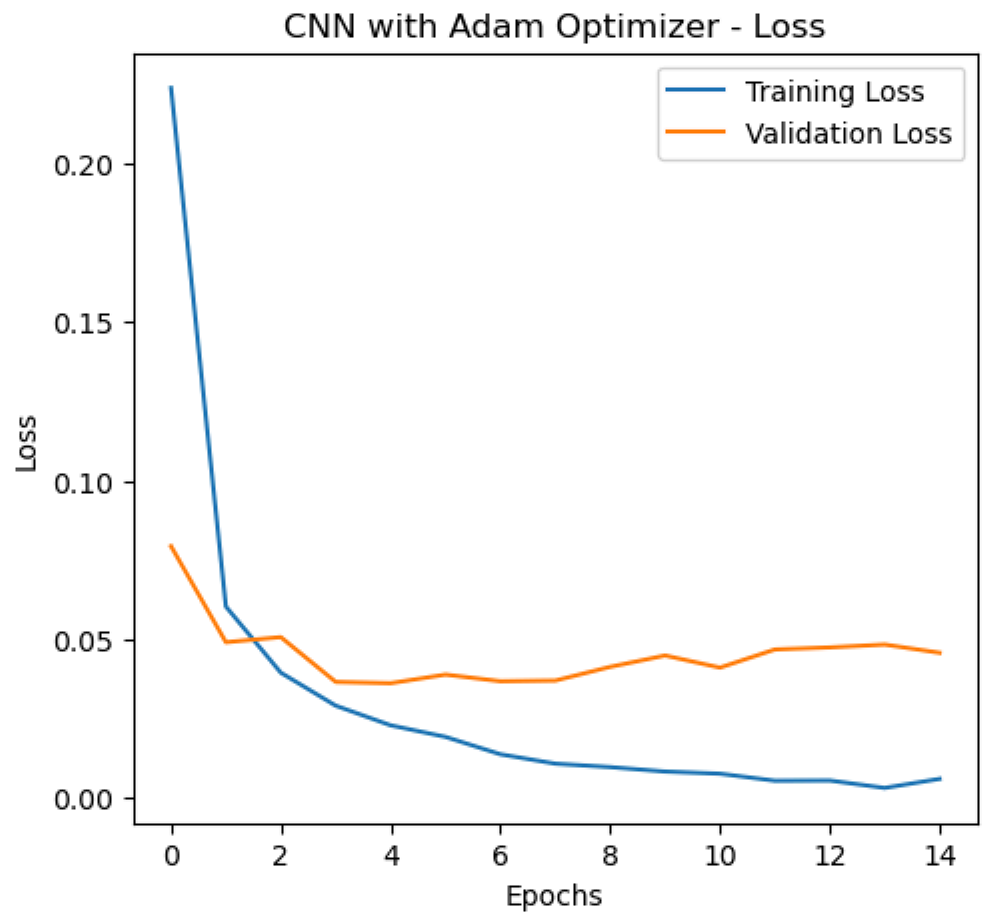
```
Epoch 1/15
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
480/480 - 6s - 12ms/step - accuracy: 0.9344 - loss: 0.2238 - val_accuracy: 0.9768 - val_loss: 0.0793
Epoch 2/15
480/480 - 6s - 12ms/step - accuracy: 0.9811 - loss: 0.0603 - val_accuracy: 0.9865 - val_loss: 0.0490
Epoch 3/15
480/480 - 6s - 13ms/step - accuracy: 0.9871 - loss: 0.0395 - val_accuracy: 0.9851 - val_loss: 0.0507
Epoch 4/15
480/480 - 6s - 12ms/step - accuracy: 0.9908 - loss: 0.0290 - val_accuracy: 0.9902 - val_loss: 0.0365
Epoch 5/15
480/480 - 6s - 12ms/step - accuracy: 0.9928 - loss: 0.0229 - val_accuracy: 0.9891 - val_loss: 0.0361
Epoch 6/15
480/480 - 6s - 13ms/step - accuracy: 0.9938 - loss: 0.0192 - val_accuracy: 0.9883 - val_loss: 0.0388
Epoch 7/15
480/480 - 6s - 13ms/step - accuracy: 0.9957 - loss: 0.0137 - val_accuracy: 0.9909 - val_loss: 0.0368
Epoch 8/15
480/480 - 6s - 12ms/step - accuracy: 0.9964 - loss: 0.0108 - val_accuracy: 0.9904 - val_loss: 0.0370
Epoch 9/15
480/480 - 6s - 13ms/step - accuracy: 0.9967 - loss: 0.0097 - val_accuracy: 0.9902 - val_loss: 0.0413
Epoch 10/15
480/480 - 6s - 13ms/step - accuracy: 0.9969 - loss: 0.0083 - val_accuracy: 0.9891 - val_loss: 0.0449
Epoch 11/15
480/480 - 7s - 14ms/step - accuracy: 0.9974 - loss: 0.0076 - val_accuracy: 0.9896 - val_loss: 0.0410
Epoch 12/15
480/480 - 6s - 13ms/step - accuracy: 0.9982 - loss: 0.0054 - val_accuracy: 0.9903 - val_loss: 0.0468
Epoch 13/15
480/480 - 7s - 14ms/step - accuracy: 0.9981 - loss: 0.0055 - val_accuracy: 0.9897 - val_loss: 0.0474
Epoch 14/15
480/480 - 7s - 14ms/step - accuracy: 0.9990 - loss: 0.0031 - val_accuracy: 0.9897 - val_loss: 0.0483
Epoch 15/15
480/480 - 6s - 13ms/step - accuracy: 0.9980 - loss: 0.0060 - val_accuracy: 0.9912 - val_loss: 0.0457
```

In [10]:

```
# Step 4B: Plot Training vs Validation Accuracy and Loss

def plot_training_curves(history, title='Model'):
```

```
    plt.figure(figsize=(12,5))

    # Accuracy Plot
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{title} - Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss Plot
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{title} - Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

# Now plot for Adam optimizer model
plot_training_curves(history_adam, title='CNN with Adam Optimizer')
```

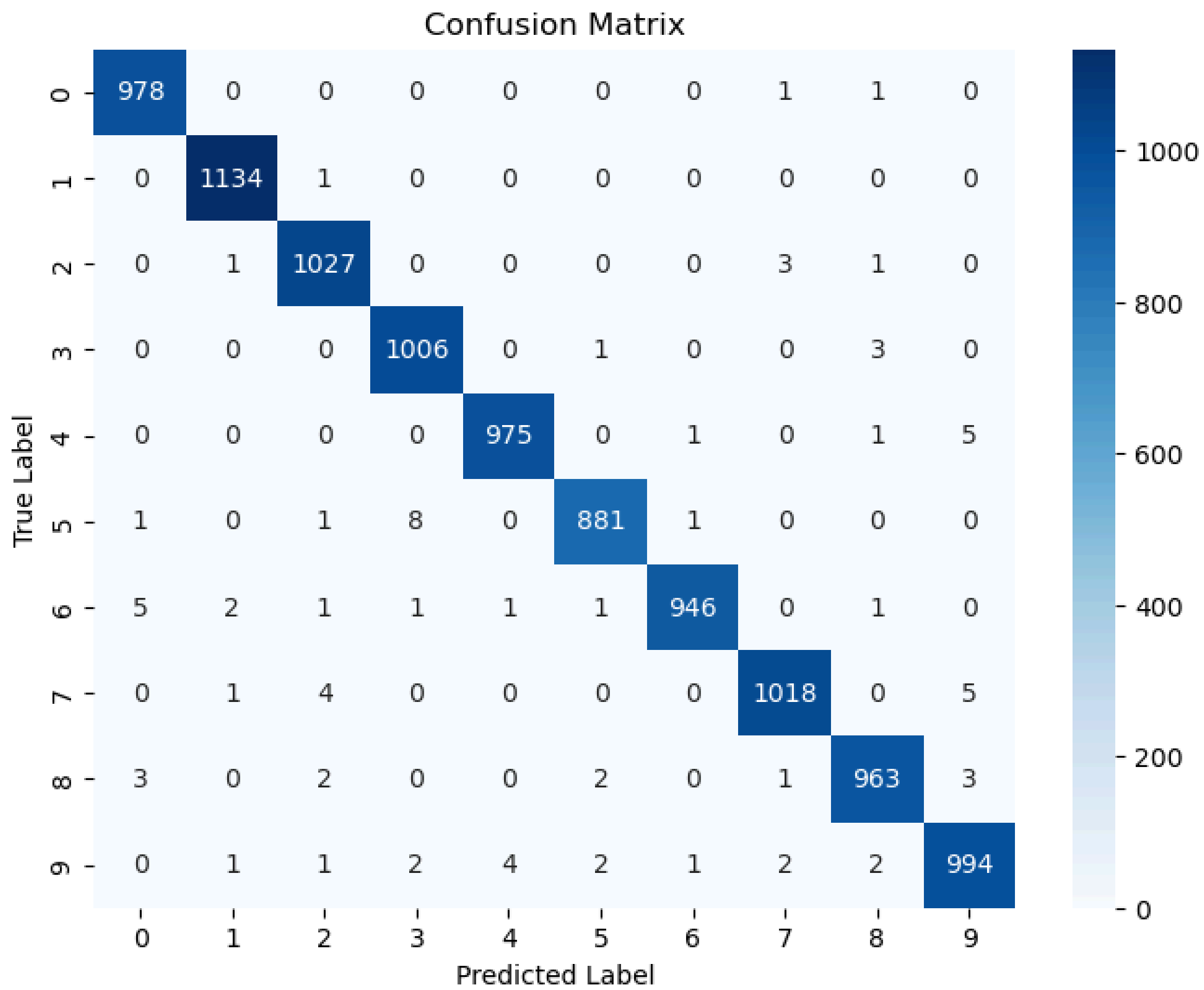




In [11]:
```
# Step 5: Evaluate the trained CNN Model

from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

def evaluate_model(model, X_test, y_test):
    # Predict probabilities
    y_pred_probs = model.predict(X_test)

    # Convert probabilities to class labels
    y_pred = np.argmax(y_pred_probs, axis=1)
    y_true = np.argmax(y_test, axis=1)
```

```
    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    # Classification Report
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Now evaluate the model you trained (Adam optimizer model)
evaluate_model(model_adam, X_test, y_test)
```

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


Confusion Matrix
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
978 0 0 0 0 0 0 1 1 0
0 1134 1 0 0 0 0 0 0 0
0 1 1027 0 0 0 0 3 1 0
0 0 0 1006 0 1 0 0 3 0
0 0 0 0 975 0 1 0 1 5
1 0 1 8 0 881 1 0 0 0
5 2 1 1 1 1 946 0 1 0
0 1 4 0 0 0 0 1018 0 5
3 0 2 0 0 2 0 1 963 3
0 1 1 2 4 2 1 2 2 994
0
200
400
600
800
1000

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       1.00      1.00      1.00      1135
           2       0.99      1.00      0.99      1032
           3       0.99      1.00      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

In [12]:

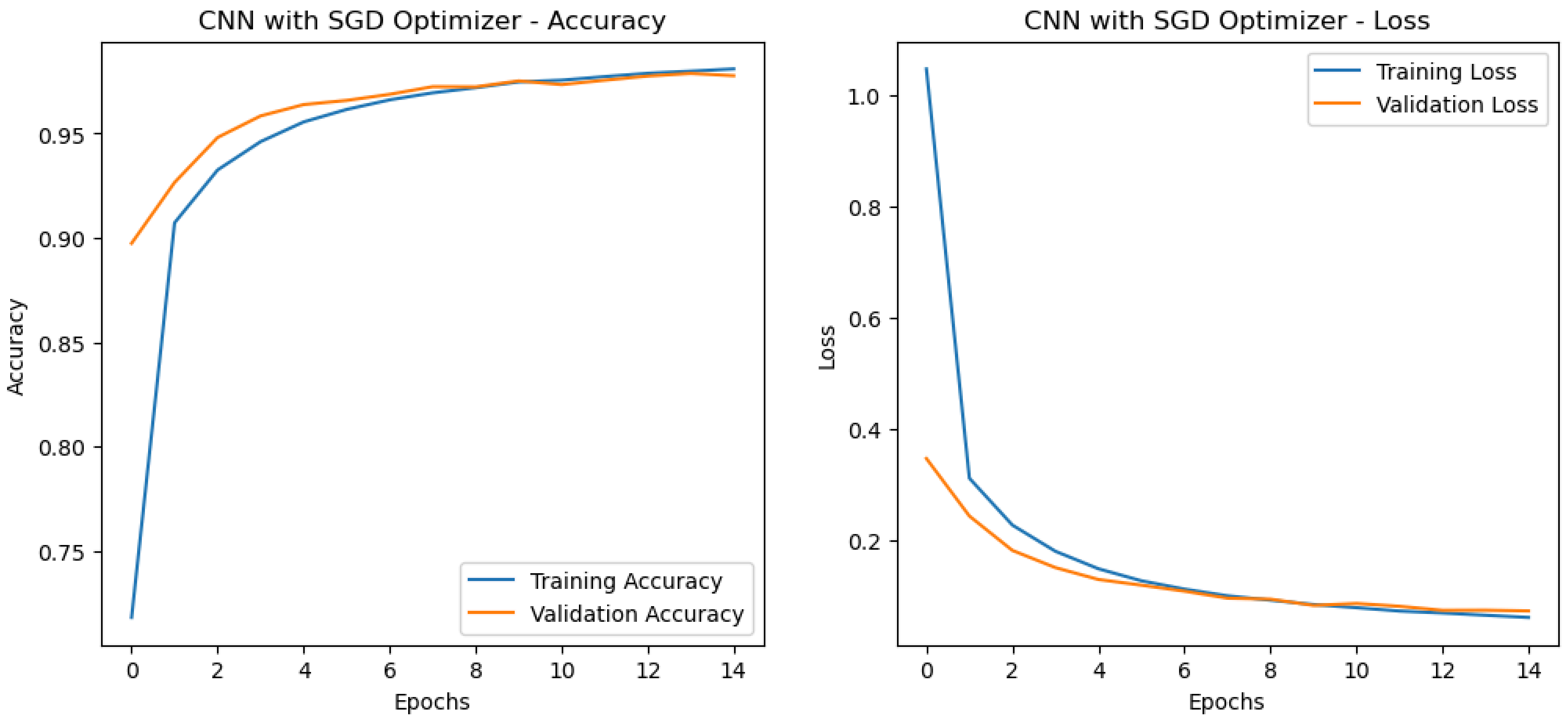
```
# Step 6A: Build and Train another CNN Model using SGD Optimizer

# Build model with SGD optimizer
model_sgd = build_cnn_model(optimizer='sgd', learning_rate=0.01)

# Train the model
history_sgd = model_sgd.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=15,
    batch_size=100,
    verbose=2
)
```

```
Epoch 1/15
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_di
m` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
480/480 - 6s - 13ms/step - accuracy: 0.7183 - loss: 1.0462 - val_accuracy: 0.8974 - val_loss: 0.3477
Epoch 2/15
480/480 - 6s - 13ms/step - accuracy: 0.9072 - loss: 0.3123 - val_accuracy: 0.9266 - val_loss: 0.2445
Epoch 3/15
480/480 - 6s - 13ms/step - accuracy: 0.9325 - loss: 0.2283 - val_accuracy: 0.9481 - val_loss: 0.1829
Epoch 4/15
480/480 - 6s - 13ms/step - accuracy: 0.9461 - loss: 0.1813 - val_accuracy: 0.9584 - val_loss: 0.1520
Epoch 5/15
480/480 - 6s - 13ms/step - accuracy: 0.9555 - loss: 0.1500 - val_accuracy: 0.9638 - val_loss: 0.1307
Epoch 6/15
480/480 - 6s - 13ms/step - accuracy: 0.9615 - loss: 0.1284 - val_accuracy: 0.9658 - val_loss: 0.1206
Epoch 7/15
480/480 - 6s - 13ms/step - accuracy: 0.9661 - loss: 0.1135 - val_accuracy: 0.9688 - val_loss: 0.1098
Epoch 8/15
480/480 - 6s - 13ms/step - accuracy: 0.9695 - loss: 0.1015 - val_accuracy: 0.9724 - val_loss: 0.0973
Epoch 9/15
480/480 - 6s - 13ms/step - accuracy: 0.9719 - loss: 0.0937 - val_accuracy: 0.9723 - val_loss: 0.0955
Epoch 10/15
480/480 - 6s - 13ms/step - accuracy: 0.9747 - loss: 0.0859 - val_accuracy: 0.9752 - val_loss: 0.0843
Epoch 11/15
480/480 - 6s - 13ms/step - accuracy: 0.9755 - loss: 0.0802 - val_accuracy: 0.9734 - val_loss: 0.0877
Epoch 12/15
480/480 - 6s - 13ms/step - accuracy: 0.9772 - loss: 0.0741 - val_accuracy: 0.9755 - val_loss: 0.0827
Epoch 13/15
480/480 - 6s - 13ms/step - accuracy: 0.9788 - loss: 0.0707 - val_accuracy: 0.9775 - val_loss: 0.0755
Epoch 14/15
480/480 - 6s - 13ms/step - accuracy: 0.9798 - loss: 0.0667 - val_accuracy: 0.9787 - val_loss: 0.0757
Epoch 15/15
480/480 - 6s - 13ms/step - accuracy: 0.9809 - loss: 0.0627 - val_accuracy: 0.9777 - val_loss: 0.0744
```

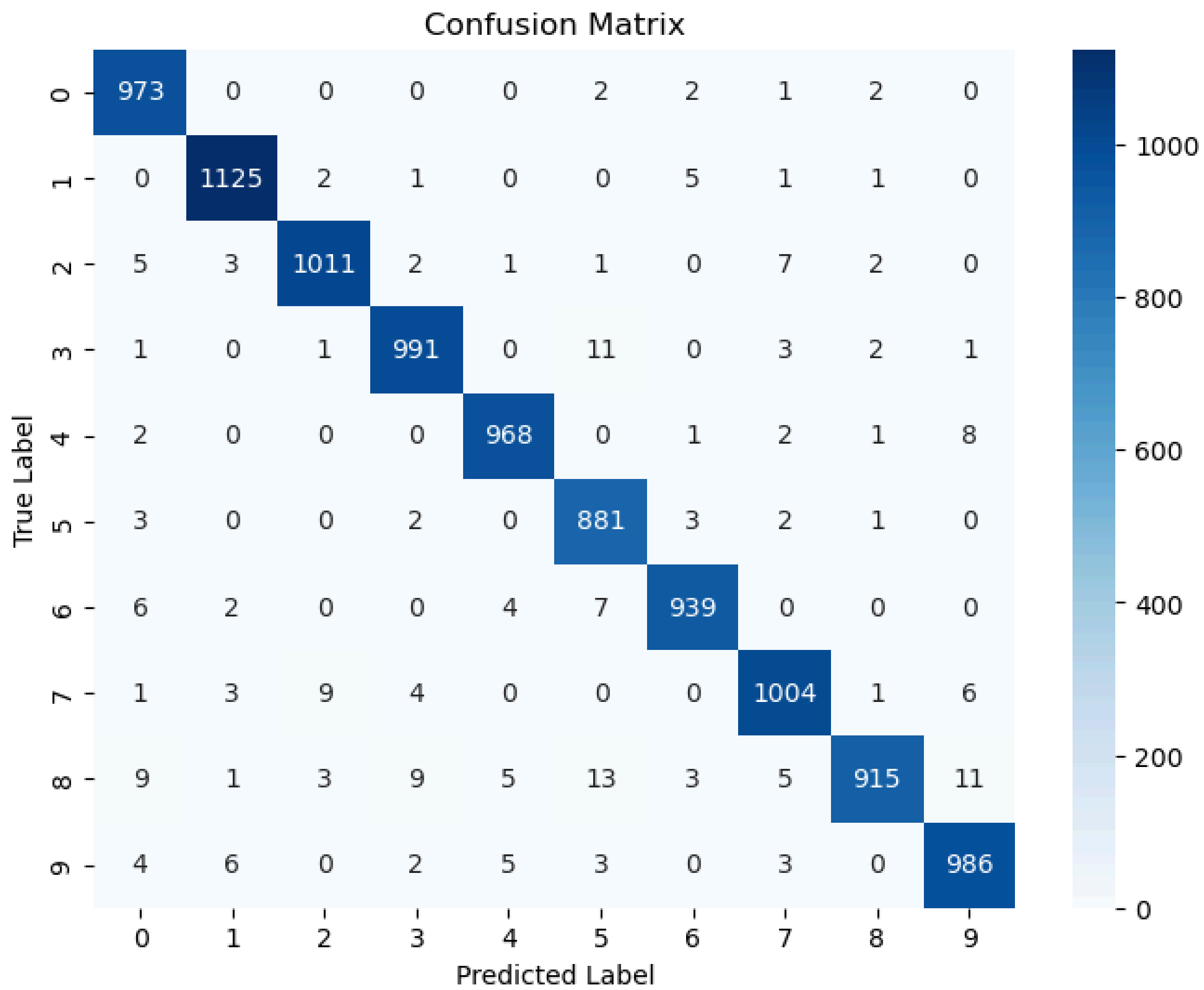
In [13]:

```python
# Step 6B: Plot Training vs Validation Accuracy and Loss for SGD Model

plot_training_curves(history_sgd, title='CNN with SGD Optimizer')
```



In [14]:
```
# Step 6C: Evaluate the SGD Optimizer Model

evaluate_model(model_sgd, X_test, y_test)
```

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


Confusion Matrix
True Label
Predicted Label
0
1
2
3
4
5
6
7
8
9
973 0 0 0 0 2 2 1 2 0
0 1125 2 1 0 0 5 1 1 0
5 3 1011 2 1 1 0 7 2 0
1 0 1 991 0 11 0 3 2 1
2 0 0 0 968 0 1 2 1 8
3 0 0 2 0 881 3 2 1 0
6 2 0 0 4 7 939 0 0 0
1 3 9 4 0 0 0 1004 1 6
9 1 3 9 5 13 3 5 915 11
4 6 0 2 5 3 0 3 0 986
1000
800
600
400
200
0

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.99      0.98      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.98      0.99      0.99       982
           5       0.96      0.99      0.97       892
           6       0.99      0.98      0.98       958
           7       0.98      0.98      0.98      1028
           8       0.99      0.94      0.96       974
           9       0.97      0.98      0.98      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

In [15]:

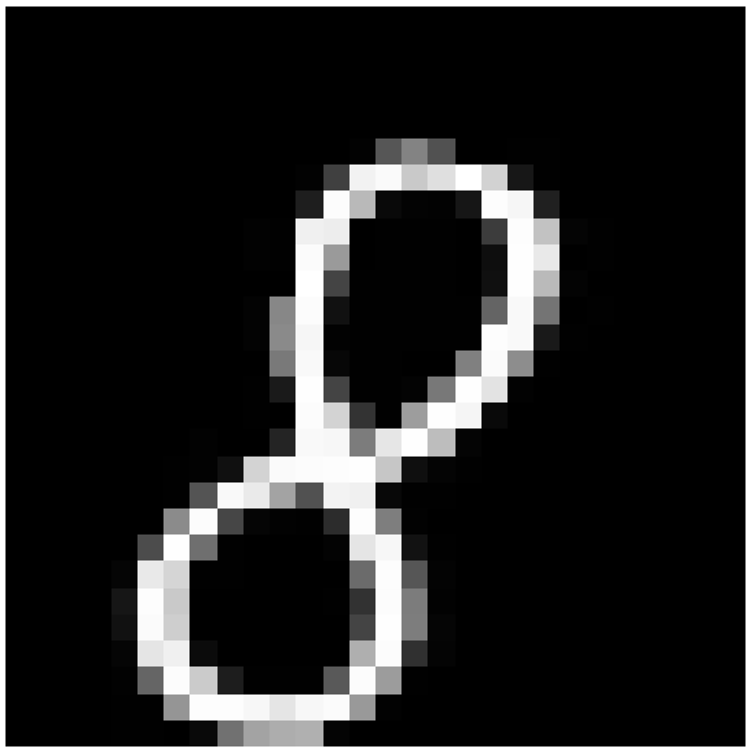
```python
# Step 7: Predict a Single Digit Image using the trained model

import cv2

def predict_single_digit(image_path, model):
    # Load image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Resize to 28x28
    img = cv2.resize(img, (28, 28))

    # Invert colors if needed
    img = 255 - img

    # Normalize pixel values
    img = img.astype('float32') / 255.0

    # Reshape to model input shape (1,28,28,1)
    img = img.reshape(1, 28, 28, 1)

    # Predict
    prediction = model.predict(img)
    predicted_digit = np.argmax(prediction)
```

```
    # Show image and prediction
    plt.imshow(img.squeeze(), cmap='gray')
    plt.title(f"Predicted Digit: {predicted_digit}")
    plt.axis('off')
    plt.show()

    return predicted_digit
```

In [16]:
```
# Give the path of your custom digit image
image_path = '/Users/manoharshasappa/Desktop/Files/3 Sem/AIT 736 NLP/Final_Project/Screenshot 2025-04-26 at 11.13.57 AM.png'

# Predict using Adam model
predicted_digit = predict_single_digit(image_path, model_adam)
print(f"Predicted Digit: {predicted_digit}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
```

Predicted Digit: 8



```
Predicted Digit: 8
```

In [ ]:

```

```

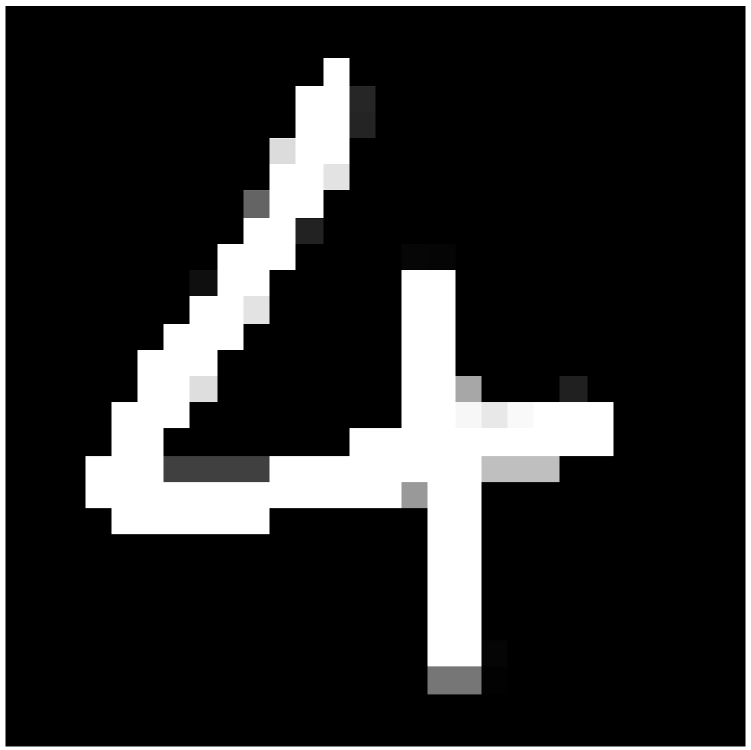
In this project, we built a CNN model to classify handwritten digits using the MNIST dataset. We trained two models using Adam and SGD optimizers for comparison. Data preprocessing steps included normalization and reshaping. The models were evaluated using accuracy/loss curves, confusion matrices, and classification reports. Hyperparameter tuning was performed by adjusting learning rate and batch size. The final system successfully predicts single handwritten digits from custom images with high accuracy

In [18]:

```python
image_path = '/Users/manoharshasappa/Desktop/Files/3 Sem/AIT 736 NLP/Final_Project/Screenshot 2025-04-26 at 12.08.46 PM.png'

# Predict using Adam model
predicted_digit = predict_single_digit(image_path, model_adam)
print(f"Predicted Digit: {predicted_digit}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```

Predicted Digit: 4

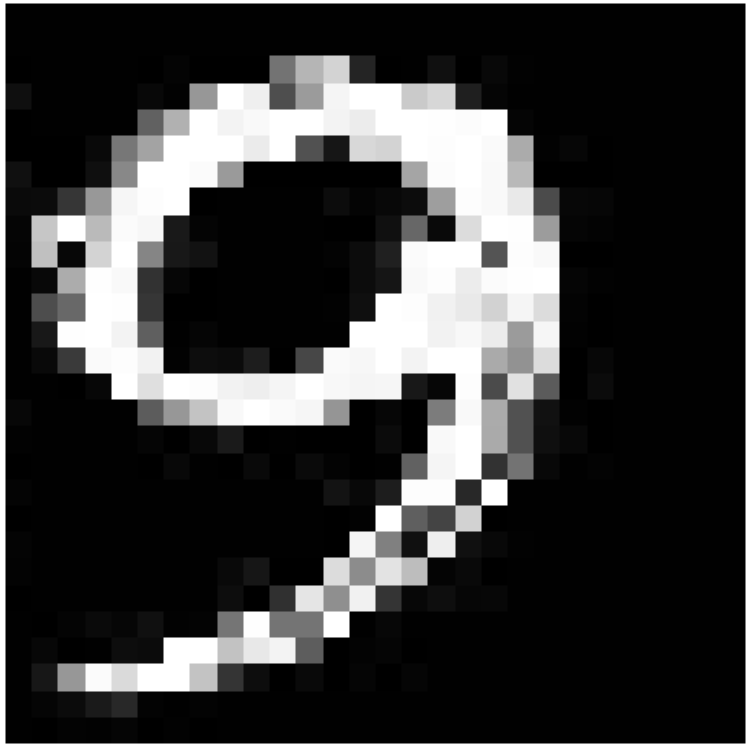


Predicted Digit: 4

In [26]:
```
image_path = '/Users/manoharshasappa/Desktop/Files/3 Sem/AIT 736 NLP/Final_Project/Screenshot 2025-04-27 at 11.10.48 PM.png'

# Predict using Adam model
predicted_digit = predict_single_digit(image_path, model_adam)
print(f"Predicted Digit: {predicted_digit}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```

Predicted Digit: 9



Predicted Digit: 9

In [ ]: