In [6]:
```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, Lambda
import numpy as np

# a. Data preparation

# Sample paragraph for demonstration
paragraph = "Continuous Bag of Words (CBOW) is a neural network model for n

# Tokenize the paragraph into words
tokens = tf.keras.preprocessing.text.text_to_word_sequence(paragraph)

# Create a word-to-index mapping
word_index = {word: idx for idx, word in enumerate(set(tokens))}

# Convert words to indices
indexed_tokens = [word_index[word] for word in tokens]

# b. Generate training data

# Define the context window size
context_window = 2

# Generate training data in the CBOW format
def generate_cbow_data(indexed_tokens, context_window):
    training_data = []
    for i in range(context_window, len(indexed_tokens) - context_window):
        context = [indexed_tokens[i - j] for j in range(context_window)] +
        [indexed_tokens[i + j] for j in range(1, context_window + 1)]
        target = indexed_tokens[i]
        training_data.append((context, target))
    return training_data

cbow_training_data = generate_cbow_data(indexed_tokens, context_window)

# Convert training data to NumPy arrays
X_train = np.array([np.array(context) for context, _ in cbow_training_data]
y_train = np.array([target for _, target in cbow_training_data])

# c. Train model

# Set the embedding dimension
embedding_dim = 50

# Build the CBOW model
model = Sequential([
    Embedding(input_dim=len(word_index), output_dim=embedding_dim,
              input_length=context_window * 2),
    Lambda(lambda x: tf.reduce_mean(x, axis=1)),  # Average over the contex
    Dense(len(word_index), activation='softmax')  # Output layer for predic
])

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

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

```python
# d. Output

# Get the word embeddings from the trained model
word_embeddings = model.layers[0].get_weights()[0]

# Example: Get the embedding for a specific word
# Example: Get the embedding for a specific word
word_to_lookup = 'CBOW'

if word_to_lookup in word_index:
    word_index_lookup = word_index[word_to_lookup]
    embedding_lookup = word_embeddings[word_index_lookup]
    print(f"Embedding for '{word_to_lookup}': {embedding_lookup}")
else:
    print(f"Word '{word_to_lookup}' not found in the dataset.")
```

```
Epoch 1/10
1/1 [==============================] - 1s 960ms/step - loss: 2.6384 - accu
racy: 0.1000
Epoch 2/10
1/1 [==============================] - 0s 8ms/step - loss: 2.6339 - accura
cy: 0.1000
Epoch 3/10
1/1 [==============================] - 0s 9ms/step - loss: 2.6294 - accura
cy: 0.1000
Epoch 4/10
1/1 [==============================] - 0s 9ms/step - loss: 2.6249 - accura
cy: 0.2000
Epoch 5/10
1/1 [==============================] - 0s 10ms/step - loss: 2.6204 - accur
acy: 0.3000
Epoch 6/10
1/1 [==============================] - 0s 9ms/step - loss: 2.6158 - accura
cy: 0.3000
Epoch 7/10
1/1 [==============================] - 0s 11ms/step - loss: 2.6113 - accur
acy: 0.3000
Epoch 8/10
1/1 [==============================] - 0s 12ms/step - loss: 2.6068 - accur
acy: 0.3000
Epoch 9/10
1/1 [==============================] - 0s 13ms/step - loss: 2.6022 - accur
acy: 0.3000
Epoch 10/10
1/1 [==============================] - 0s 13ms/step - loss: 2.5976 - accur
acy: 0.3000
Word 'CBOW' not found in the dataset.
```

In [9]:
```
# a. Import required libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import numpy as np

# b. Upload / access the dataset

# Load the MNIST dataset
(train_images, _), (test_images, _) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Add a channel dimension to the images (MNIST is grayscale, so channel=1)
train_images = train_images[..., tf.newaxis]
test_images = test_images[..., tf.newaxis]

# c. Encoder converts it into a latent representation

# Create an autoencoder model
encoder = models.Sequential()
encoder.add(layers.InputLayer(input_shape=(28, 28, 1)))
encoder.add(layers.Conv2D(16, (3, 3), activation='relu', padding='same'))
encoder.add(layers.MaxPooling2D((2, 2), padding='same'))
encoder.add(layers.Conv2D(8, (3, 3), activation='relu', padding='same'))
encoder.add(layers.MaxPooling2D((2, 2), padding='same'))
encoder.add(layers.Conv2D(4, (3, 3), activation='relu', padding='same'))
encoder.add(layers.MaxPooling2D((2, 2), padding='same'))

# Print the encoder summary
encoder.summary()

# d. Decoder networks convert it back to the original input

# Create a decoder model
decoder = models.Sequential()
decoder.add(layers.InputLayer(input_shape=(4, 4, 4)))  # Latent representat
decoder.add(layers.Conv2D(4, (3, 3), activation='relu', padding='same'))
decoder.add(layers.UpSampling2D((2, 2)))
decoder.add(layers.Conv2D(8, (3, 3), activation='relu', padding='same'))
decoder.add(layers.UpSampling2D((2, 2)))
decoder.add(layers.Conv2D(16, (3, 3), activation='relu'))
decoder.add(layers.UpSampling2D((2, 2)))
decoder.add(layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same'))

# Print the decoder summary
decoder.summary()

# e. Compile the models with Optimizer, Loss, and Evaluation Metrics

# Create the autoencoder model
autoencoder = models.Sequential([encoder, decoder])

# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Train the autoencoder on normal data
autoencoder.fit(train_images, train_images, epochs=10, batch_size=128,
                shuffle=True, validation_data=(test_images, test_images))
```

```
# Use the trained autoencoder to reconstruct test images
reconstructed_images = autoencoder.predict(test_images)

# Calculate the Mean Squared Error (MSE) between original and reconstructed
mse = np.mean(np.square(test_images - reconstructed_images))
print(f'Mean Squared Error on Test Data: {mse:.4f}')
```

```
WARNING:tensorflow:From C:\Users\JANHVI MAWAL\AppData\Roaming\Python\Pytho
n39\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name
tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2  (None, 14, 14, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 8)         1160

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 8)           0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 4)           292

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 4)           0
 g2D)

=================================================================
Total params: 1612 (6.30 KB)
Trainable params: 1612 (6.30 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 4, 4, 4)           148

 up_sampling2d (UpSampling2  (None, 8, 8, 4)           0
 D)

 conv2d_4 (Conv2D)           (None, 8, 8, 8)           296

 up_sampling2d_1 (UpSamplin  (None, 16, 16, 8)         0
 g2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 16)        1168

 up_sampling2d_2 (UpSamplin  (None, 28, 28, 16)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 28, 28, 1)         145

=================================================================
Total params: 1757 (6.86 KB)
Trainable params: 1757 (6.86 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 22s 36ms/step - loss: 0.1163 -
val_loss: 0.1140
Epoch 2/10
469/469 [==============================] - 16s 35ms/step - loss: 0.1120 -
val_loss: 0.1140
Epoch 3/10
```

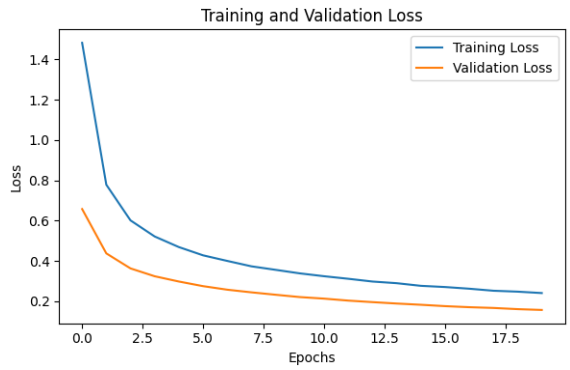
```
469/469 [==============================] - 17s 36ms/step - loss: 0.1120 -
val_loss: 0.1140
Epoch 4/10
469/469 [==============================] - 17s 36ms/step - loss: 0.1120 -
val_loss: 0.1140
Epoch 5/10
469/469 [==============================] - 16s 34ms/step - loss: 0.1120 -
val_loss: 0.1140
Epoch 6/10
469/469 [==============================] - 17s 36ms/step - loss: 0.1120 -
val_loss: 0.1140
Epoch 7/10
469/469 [==============================] - 16s 35ms/step - loss: 0.1120 -
val_loss: 0.1140
Epoch 8/10
469/469 [==============================] - 17s 35ms/step - loss: 0.1120 -
val_loss: 0.1140
Epoch 9/10
469/469 [==============================] - 16s 35ms/step - loss: 0.1120 -
val_loss: 0.1140
Epoch 10/10
469/469 [==============================] - 16s 34ms/step - loss: 0.1120 -
val_loss: 0.1140
313/313 [==============================] - 3s 7ms/step
Mean Squared Error on Test Data: 0.1140
```

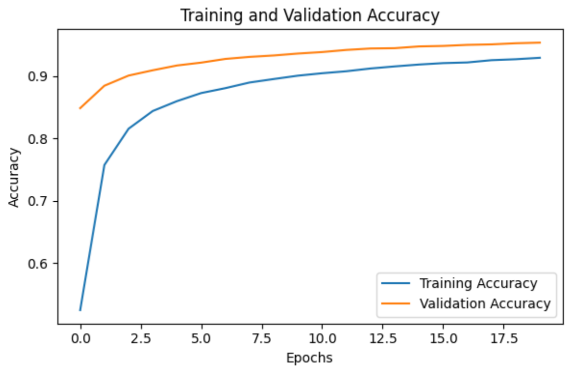
In [10]:
```
# a. Import the necessary packages
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# b. Load the training and testing data (MNIST)
(train_images, train_labels), (test_images, test_labels) = mnist.load_data(

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Flatten the images (convert 28x28 images to a 1D array of 784)
train_images = train_images.reshape((60000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))

# One-hot encode the labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# c. Define the network architecture using Keras
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))  # 10 classes for digits

# Print the model summary
model.summary()

# d. Train the model using SGD
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=20, batch_size=128,
                    validation_data=(test_images, test_labels))

# e. Evaluate the network
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_acc*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

# f. Plot the training loss and accuracy
plt.figure(figsize=(12, 4))

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

# Plot Training Accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_6 (Dense)             (None, 512)               401920    
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_7 (Dense)             (None, 256)               131328    
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
                                                                 
 dense_8 (Dense)             (None, 10)                2570      
                                                                 
=================================================================
Total params: 535818 (2.04 MB)
Trainable params: 535818 (2.04 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 7s 13ms/step - loss: 1.4821 - a
ccuracy: 0.5242 - val_loss: 0.6573 - val_accuracy: 0.8485
Epoch 2/20
469/469 [==============================] - 6s 13ms/step - loss: 0.7779 - a
ccuracy: 0.7572 - val_loss: 0.4372 - val_accuracy: 0.8845
Epoch 3/20
469/469 [==============================] - 6s 13ms/step - loss: 0.6009 - a
ccuracy: 0.8155 - val_loss: 0.3627 - val_accuracy: 0.9008
Epoch 4/20
469/469 [==============================] - 6s 13ms/step - loss: 0.5211 - a
ccuracy: 0.8439 - val_loss: 0.3236 - val_accuracy: 0.9093
Epoch 5/20
469/469 [==============================] - 6s 14ms/step - loss: 0.4684 - a
ccuracy: 0.8597 - val_loss: 0.2975 - val_accuracy: 0.9170
Epoch 6/20
469/469 [==============================] - 6s 12ms/step - loss: 0.4275 - a
ccuracy: 0.8727 - val_loss: 0.2746 - val_accuracy: 0.9216
Epoch 7/20
469/469 [==============================] - 6s 12ms/step - loss: 0.3999 - a
ccuracy: 0.8806 - val_loss: 0.2571 - val_accuracy: 0.9275
Epoch 8/20
469/469 [==============================] - 6s 12ms/step - loss: 0.3730 - a
ccuracy: 0.8897 - val_loss: 0.2439 - val_accuracy: 0.9308
Epoch 9/20
469/469 [==============================] - 6s 12ms/step - loss: 0.3555 - a
ccuracy: 0.8953 - val_loss: 0.2320 - val_accuracy: 0.9331
Epoch 10/20
469/469 [==============================] - 6s 13ms/step - loss: 0.3380 - a
ccuracy: 0.9007 - val_loss: 0.2204 - val_accuracy: 0.9362
Epoch 11/20
469/469 [==============================] - 6s 13ms/step - loss: 0.3238 - a
ccuracy: 0.9045 - val_loss: 0.2127 - val_accuracy: 0.9386
Epoch 12/20
469/469 [==============================] - 6s 13ms/step - loss: 0.3112 - a
ccuracy: 0.9077 - val_loss: 0.2029 - val_accuracy: 0.9420
Epoch 13/20
469/469 [==============================] - 6s 13ms/step - loss: 0.2975 - a
ccuracy: 0.9122 - val_loss: 0.1953 - val_accuracy: 0.9443
Epoch 14/20
469/469 [==============================] - 6s 12ms/step - loss: 0.2893 - a
ccuracy: 0.9155 - val_loss: 0.1884 - val_accuracy: 0.9448
```

```
Epoch 15/20
469/469 [==============================] - 6s 12ms/step - loss: 0.2763 - a
ccuracy: 0.9185 - val_loss: 0.1822 - val_accuracy: 0.9476
Epoch 16/20
469/469 [==============================] - 6s 12ms/step - loss: 0.2703 - a
ccuracy: 0.9208 - val_loss: 0.1754 - val_accuracy: 0.9484
Epoch 17/20
469/469 [==============================] - 6s 13ms/step - loss: 0.2620 - a
ccuracy: 0.9219 - val_loss: 0.1702 - val_accuracy: 0.9501
Epoch 18/20
469/469 [==============================] - 6s 13ms/step - loss: 0.2519 - a
ccuracy: 0.9255 - val_loss: 0.1664 - val_accuracy: 0.9509
Epoch 19/20
469/469 [==============================] - 6s 13ms/step - loss: 0.2473 - a
ccuracy: 0.9270 - val_loss: 0.1605 - val_accuracy: 0.9527
Epoch 20/20
469/469 [==============================] - 6s 13ms/step - loss: 0.2404 - a
ccuracy: 0.9293 - val_loss: 0.1565 - val_accuracy: 0.9537
313/313 [==============================] - 1s 4ms/step - loss: 0.1565 - ac
curacy: 0.9537
Test Accuracy: 95.37%
Test Loss: 0.1565
```

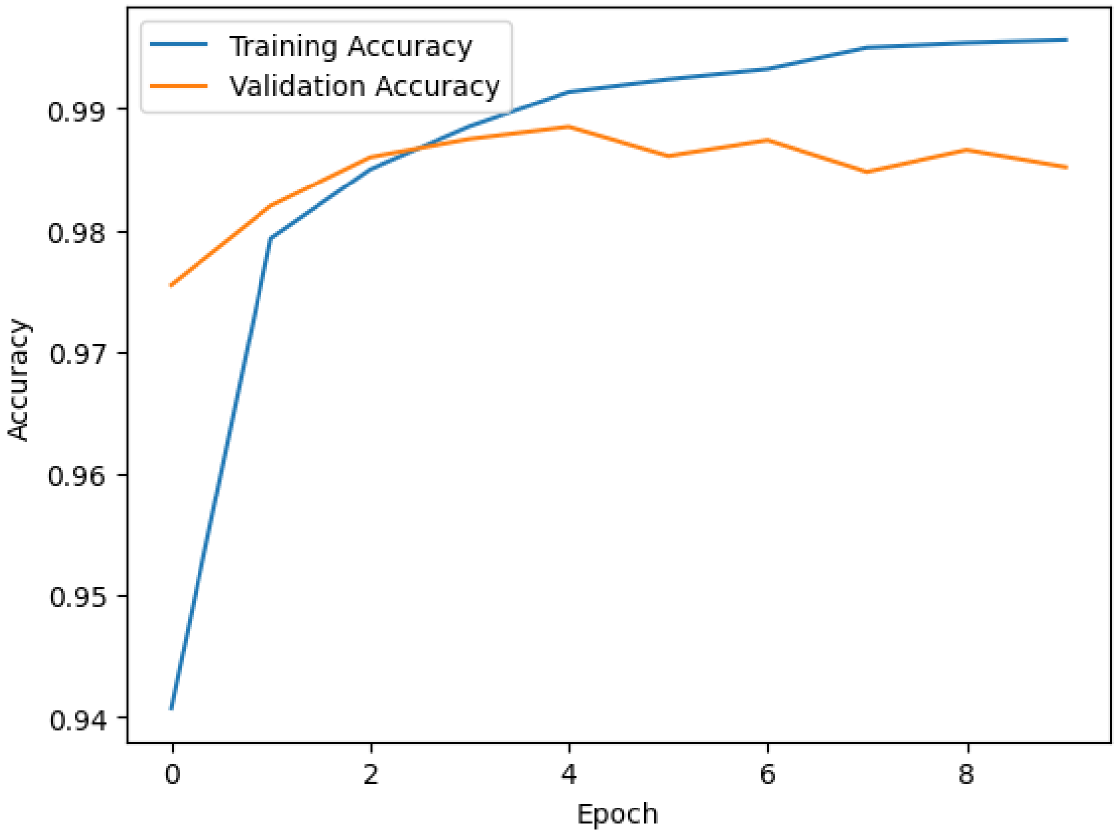




In [12]:
```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# a. Loading and preprocessing the image data

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data(

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Add a channel dimension to the images (MNIST is grayscale, so channel=1)
train_images = train_images[..., tf.newaxis]
test_images = test_images[..., tf.newaxis]

# One-hot encode the labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# b. Defining the model’s architecture

# Create a simple convolutional neural network (CNN) model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))  # 10 classes for digits

# Print the model summary
model.summary()

# c. Training the model

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
epochs = 10
history = model.fit(
    train_images,
    train_labels,
    epochs=epochs,
    validation_data=(test_images, test_labels)
)

# d. Estimating the model’s performance

# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_acc*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_7 (Conv2D)           (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 13, 13, 32)        0         
 g2D)                                                            
                                                                 
 conv2d_8 (Conv2D)           (None, 11, 11, 64)        18496     
                                                                 
 max_pooling2d_4 (MaxPoolin  (None, 5, 5, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_9 (Conv2D)           (None, 3, 3, 128)         73856     
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 1, 1, 128)         0         
 g2D)                                                            
                                                                 
 flatten_1 (Flatten)         (None, 128)               0         
                                                                 
 dense_9 (Dense)             (None, 128)               16512     
                                                                 
 dense_10 (Dense)            (None, 10)                1290      
                                                                 
=================================================================
Total params: 110474 (431.54 KB)
Trainable params: 110474 (431.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
1875/1875 [==============================] - 29s 14ms/step - loss: 0.1915
- accuracy: 0.9407 - val_loss: 0.0821 - val_accuracy: 0.9755
Epoch 2/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.0663
- accuracy: 0.9793 - val_loss: 0.0613 - val_accuracy: 0.9820
Epoch 3/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0474
- accuracy: 0.9850 - val_loss: 0.0460 - val_accuracy: 0.9860
Epoch 4/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0363
- accuracy: 0.9885 - val_loss: 0.0455 - val_accuracy: 0.9875
Epoch 5/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0289
- accuracy: 0.9914 - val_loss: 0.0416 - val_accuracy: 0.9885
Epoch 6/10
1875/1875 [==============================] - 27s 14ms/step - loss: 0.0235
- accuracy: 0.9924 - val_loss: 0.0500 - val_accuracy: 0.9861
Epoch 7/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.0199
- accuracy: 0.9933 - val_loss: 0.0474 - val_accuracy: 0.9874
Epoch 8/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.0146
- accuracy: 0.9950 - val_loss: 0.0643 - val_accuracy: 0.9848
Epoch 9/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.0149
- accuracy: 0.9954 - val_loss: 0.0596 - val_accuracy: 0.9866
Epoch 10/10
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0128
- accuracy: 0.9956 - val_loss: 0.0729 - val_accuracy: 0.9852
```




```
313/313 [==============================] - 2s 6ms/step - loss: 0.0729 - ac
curacy: 0.9852
Test Accuracy: 98.52%
Test Loss: 0.0729
```

In [ ]: