In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
```

In [10]:
```
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# One-hot encode labels
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

# Verify the shapes
print(f'Training data shape: {x_train.shape}, Training labels shape: {y_train.shape}')
print(f'Test data shape: {x_test.shape}, Test labels shape: {y_test.shape}')
```

```
Training data shape: (60000, 28, 28), Training labels shape: (60000, 10)
Test data shape: (10000, 28, 28), Test labels shape: (10000, 10)
```

In [11]:
```
# Define the model architecture
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Input layer
    Dense(128, activation='relu'),   # Hidden layer
    Dense(10, activation='softmax')   # Output layer
])

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

# Print the model summary
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_2 (Dense)	(None, 128)	100,480
dense_3 (Dense)	(None, 10)	1,290

Total params: 101,770 (397.54 KB)

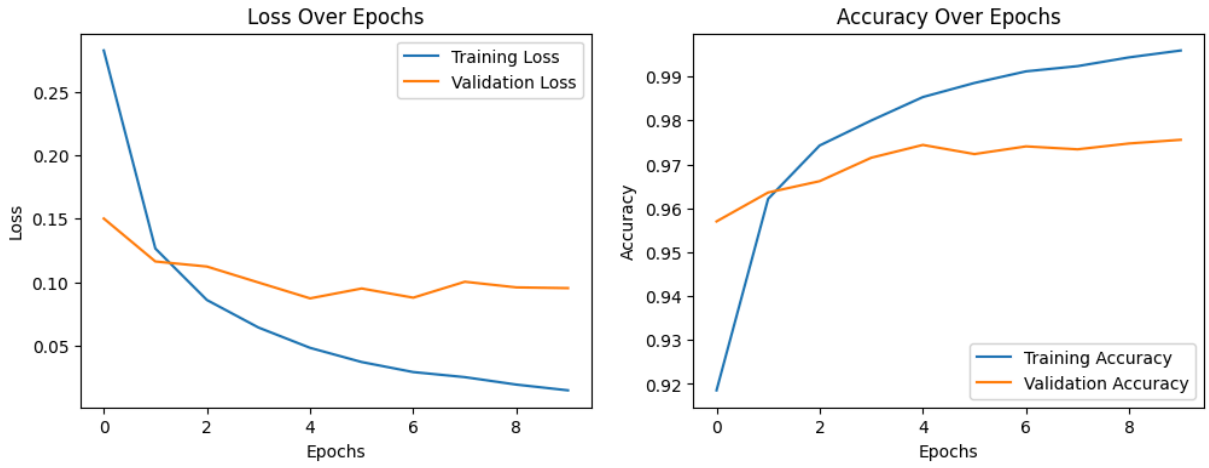
Trainable params: 101,770 (397.54 KB)

Non-trainable params: 0 (0.00 B)

In [12]:
```
# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
history = model.fit(x_train, y_train,
                    validation_split=0.2,
                    epochs=50,
                    batch_size=32,
                    callbacks=[early_stopping])
```

```
Epoch 1/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8664 - loss: 0.4685 - val_accuracy: 0.9570 - val
_loss: 0.1501
Epoch 2/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9606 - loss: 0.1363 - val_accuracy: 0.9636 - val
_loss: 0.1163
Epoch 3/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9747 - loss: 0.0828 - val_accuracy: 0.9662 - val
_loss: 0.1123
Epoch 4/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9805 - loss: 0.0643 - val_accuracy: 0.9715 - val
_loss: 0.0997
Epoch 5/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9863 - loss: 0.0461 - val_accuracy: 0.9744 - val
_loss: 0.0873
Epoch 6/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9898 - loss: 0.0345 - val_accuracy: 0.9723 - val
_loss: 0.0950
Epoch 7/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9915 - loss: 0.0295 - val_accuracy: 0.9741 - val
_loss: 0.0878
Epoch 8/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9930 - loss: 0.0235 - val_accuracy: 0.9734 - val
_loss: 0.1003
Epoch 9/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9956 - loss: 0.0168 - val_accuracy: 0.9747 - val
_loss: 0.0959
Epoch 10/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9963 - loss: 0.0136 - val_accuracy: 0.9756 - val
_loss: 0.0953
```

In [13]:
```python
# Plot training and validation loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot training and validation 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('Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [14]:
```python
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 850us/step - accuracy: 0.9733 - loss: 0.0898
Test Loss: 0.0784, Test Accuracy: 0.9764
```

In [ ]:

In [ ]: