In [14]:

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras.datasets import mnist  # Import MNIST dataset

# Hyperparameters (adjust as needed)
learning_rate = 0.001  # Learning rate for the optimizer
batch_size = 32       # Number of images processed in each batch
epochs = 10           # Number of training epochs

# Load MNIST dataset (replace with your data if not using MNIST)
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess data (normalize pixel values to 0-1 range)
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data to include channel dimension (if grayscale)
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

# One-hot encode target labels (optional, depending on task)
# y_train = tf.keras.utils.to_categorical(y_train, 10)  # For 10 categories
# y_test = tf.keras.utils.to_categorical(y_test, 10)

# Define the CNN model
model = models.Sequential([
    # Convolutional Layer 1
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28
    layers.MaxPooling2D((2, 2)),

    # Convolutional Layer 2 (Corrected input shape)
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(14, 14

    # Max Pooling Layer 2
    layers.MaxPooling2D((2, 2)),

    # Flatten Layer
    layers.Flatten(),

    # Fully-Connected Layer 1
    layers.Dense(units=3136, activation='relu'),

    # Fully-Connected Layer 2
    layers.Dense(units=128, activation='relu'),

    # Fully-Connected Layer 3
    layers.Dense(units=128, activation='relu'),

    # Output Layer
    layers.Dense(units=10, activation='softmax')  # 10 output units for 10 categories (M
])

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

# Split training data for validation (optional, adjust validation split)
x_val = x_train[:10000]  # Split first 10000 images for validation
y_val = y_train[:10000]
remaining_train_images = x_train[10000:]
remaining_train_labels = y_train[10000:]

# Train the model with validation
model_history = model.fit(remaining_train_images, remaining_train_labels,
                          batch_size=batch_size, epochs=epochs,
                          validation_data=(x_val, y_val))

# Print Training and Validation Accuracy/Loss per Epoch
print("Training Accuracy:")
for i in range(epochs):
    print(f"Epoch {i+1}: {model_history.history['accuracy'][i]:.4f}")
print("\nTraining Loss:")
for i in range(epochs):
    print(f"Epoch {i+1}: {model_history.history['loss'][i]:.4f}")

print("\nValidation Accuracy:")
for i in range(epochs):
    print(f"Epoch {i+1}: {model_history.history['val_accuracy'][i]:.4f}")
print("\nValidation Loss:")
for i in range(epochs):
    print(f"Epoch {i+1}: {model_history.history['val_loss'][i]:.4f}")

# Evaluate the model on test data
# Evaluate the model on test data
test_loss, test_acc = model.evaluate(x_test, y_test)

# Print Test Accuracy and Loss
print("\nTest Accuracy:", test_acc)
print("Test Loss:", test_loss)

# Access Model Parameters (e.g., Weights and Biases)
# Example: Accessing weights of the first convolutional layer
first_conv_weights = model.layers[0].get_weights()[0]
print("\nFirst Convolutional Layer Weights (example):")
print(first_conv_weights.shape)  # Print the shape of the weight tensor

# Save the Model (Optional)
model.save('my_cnn_model.h5')
```

```
Epoch 1/10
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [14], line 64
     61 remaining_train_labels = y_train[10000:]
     63 # Train the model with validation
---> 64 model_history = model.fit(remaining_train_images, remaining_train_labels,
     65                           batch_size=batch_size, epochs=epochs,
     66                           validation_data=(x_val, y_val))
     68 # Print Training and Validation Accuracy/Loss per Epoch
     69 print("Training Accuracy:")

File ~\AppData\Roaming\Python\Python311\site-packages\keras\utils\traceback_utils.py:70,
 in filter_traceback.<locals>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File ~\AppData\Local\Temp\__autograph_generated_file1x87myu4.py:15, in outer_factory.<lo
cals>.inner_factory.<locals>.tf__train_function(iterator)
     13 try:
     14     do_return = True
---> 15     retval_ = ag__.converted_call(ag__.ld(step_function), (ag__.ld(self), ag__.l
d(iterator)), None, fscope)
     16 except:
     17     do_return = False
```

```
ValueError: in user code:

    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\en
gine\training.py", line 1284, in train_function  *
        return step_function(self, iterator)
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\en
gine\training.py", line 1268, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\en
gine\training.py", line 1249, in run_step  **
        outputs = model.train_step(data)
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\en
gine\training.py", line 1051, in train_step
        loss = self.compute_loss(x, y, y_pred, sample_weight)
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\en
gine\training.py", line 1109, in compute_loss
        return self.compiled_loss(
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\en
gine\compile_utils.py", line 265, in __call__
        loss_value = loss_obj(y_t, y_p, sample_weight=sw)
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\lo
sses.py", line 142, in __call__
        losses = call_fn(y_true, y_pred)
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\lo
sses.py", line 268, in call  **
        return ag_fn(y_true, y_pred, **self._fn_kwargs)
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\lo
sses.py", line 1984, in categorical_crossentropy
        return backend.categorical_crossentropy(
    File "C:\Users\Prayag Chawla\AppData\Roaming\Python\Python311\site-packages\keras\ba
ckend.py", line 5559, in categorical_crossentropy
        target.shape.assert_is_compatible_with(output.shape)

    ValueError: Shapes (None, 1) and (None, 10) are incompatible
```

In [15]:

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist  # Import MNIST dataset


learning_rate = 0.001
batch_size = 32
epochs = 10


(x_train, y_train), (x_test, y_test) = mnist.load_data()


x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

# One-hot encode target labels (optional, depending on task)
y_train = tf.keras.utils.to_categorical(y_train, 10)  # For 10 categories
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Define the CNN model
model = models.Sequential([

    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'),


    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),


    layers.Dense(units=3136, activation='relu'),


    layers.Dense(units=128, activation='relu'),

    layers.Dense(units=10, activation='softmax')  # 10 output units for 10 categories (M
])


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


val_split = 0.1
val_samples = int(val_split * x_train.shape[0])
x_val, y_val = x_train[:val_samples], y_train[:val_samples]
x_train, y_train = x_train[val_samples:], y_train[val_samples:]


model_history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, valida

print("Training Accuracy:")
for i in range(epochs):
    print(f"Epoch {i+1}: {model_history.history['accuracy'][i]:.4f}")
print("\nTraining Loss:")
for i in range(epochs):
    print(f"Epoch {i+1}: {model_history.history['loss'][i]:.4f}")

print("\nValidation Accuracy:")
for i in range(epochs):
    print(f"Epoch {i+1}: {model_history.history['val_accuracy'][i]:.4f}")
print("\nValidation Loss:")
for i in range(epochs):
    print(f"Epoch {i+1}: {model_history.history['val_loss'][i]:.4f}")

test_loss, test_acc = model.evaluate(x_test, y_test)

print("\nTest Accuracy:", test_acc)
print("Test Loss:", test_loss)


first_conv_weights = model.layers[0].get_weights()[0]
print("\nFirst Convolutional Layer Weights (example):")
print(first_conv_weights.shape)  #


model.save('my_cnn_model.h5')
```

```
Epoch 1/10
1688/1688 [==============================] - 74s 43ms/step - loss: 0.1216 - accuracy: 0.
9615 - val_loss: 0.0591 - val_accuracy: 0.9837
Epoch 2/10
1688/1688 [==============================] - 72s 43ms/step - loss: 0.0461 - accuracy: 0.
9857 - val_loss: 0.0525 - val_accuracy: 0.9835
Epoch 3/10
1688/1688 [==============================] - 85s 50ms/step - loss: 0.0315 - accuracy: 0.
9901 - val_loss: 0.0515 - val_accuracy: 0.9843
Epoch 4/10
1688/1688 [==============================] - 74s 44ms/step - loss: 0.0243 - accuracy: 0.
```

```
9926 - val_loss: 0.0383 - val_accuracy: 0.9883
Epoch 5/10
1688/1688 [==============================] - 85s 51ms/step - loss: 0.0198 - accuracy: 0.
9940 - val_loss: 0.0364 - val_accuracy: 0.9892
Epoch 6/10
1688/1688 [==============================] - 93s 55ms/step - loss: 0.0143 - accuracy: 0.
9958 - val_loss: 0.0521 - val_accuracy: 0.9868
Epoch 7/10
1688/1688 [==============================] - 94s 56ms/step - loss: 0.0127 - accuracy: 0.
9960 - val_loss: 0.0369 - val_accuracy: 0.9905
Epoch 8/10
1688/1688 [==============================] - 100s 59ms/step - loss: 0.0123 - accuracy:
0.9962 - val_loss: 0.0474 - val_accuracy: 0.9893
Epoch 9/10
1688/1688 [==============================] - 94s 55ms/step - loss: 0.0096 - accuracy: 0.
9972 - val_loss: 0.0396 - val_accuracy: 0.9905
Epoch 10/10
1688/1688 [==============================] - 86s 51ms/step - loss: 0.0092 - accuracy: 0.
9972 - val_loss: 0.0448 - val_accuracy: 0.9910
Training Accuracy:
Epoch 1: 0.9615
Epoch 2: 0.9857
Epoch 3: 0.9901
Epoch 4: 0.9926
Epoch 5: 0.9940
Epoch 6: 0.9958
Epoch 7: 0.9960
Epoch 8: 0.9962
Epoch 9: 0.9972
Epoch 10: 0.9972

Training Loss:
Epoch 1: 0.1216
Epoch 2: 0.0461
Epoch 3: 0.0315
Epoch 4: 0.0243
Epoch 5: 0.0198
Epoch 6: 0.0143
Epoch 7: 0.0127
Epoch 8: 0.0123
Epoch 9: 0.0096
Epoch 10: 0.0092

Validation Accuracy:
Epoch 1: 0.9837
Epoch 2: 0.9835
Epoch 3: 0.9843
Epoch 4: 0.9883
Epoch 5: 0.9892
Epoch 6: 0.9868
Epoch 7: 0.9905
Epoch 8: 0.9893
Epoch 9: 0.9905
Epoch 10: 0.9910

Validation Loss:
Epoch 1: 0.0591
Epoch 2: 0.0525
Epoch 3: 0.0515
Epoch 4: 0.0383
Epoch 5: 0.0364
Epoch 6: 0.0521
Epoch 7: 0.0369
Epoch 8: 0.0474
Epoch 9: 0.0396
Epoch 10: 0.0448
```

```
313/313 [==============================] - 5s 16ms/step - loss: 0.0405 - accuracy: 0.989
7

Test Accuracy: 0.9897000193595886
Test Loss: 0.040509123355150022

First Convolutional Layer Weights (example):
(3, 3, 1, 32)
```