```
import numpy as np
from tensorflow.keras.datasets import mnist

# Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize to [0, 1]
x_train = x_train.astype(np.float32) / 255.0
x_test = x_test.astype(np.float32) / 255.0

# Flatten images: 28x28 -> 784
x_train = x_train.reshape(-1, 784)
x_test = x_test.reshape(-1, 784)

# One-hot encode labels
def one_hot_encode(y, num_classes=10):
    return np.eye(num_classes)[y]

y_train_oh = one_hot_encode(y_train)
y_test_oh = one_hot_encode(y_test)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
def initialize_weights():
    np.random.seed(42)  # For reproducibility
    weights = {
        'W1': np.random.randn(784, 128) * np.sqrt(1. / 784),
        'b1': np.zeros((1, 128)),
        'W2': np.random.randn(128, 64) * np.sqrt(1. / 128),
        'b2': np.zeros((1, 64)),
        'W3': np.random.randn(64, 10) * np.sqrt(1. / 64),
        'b3': np.zeros((1, 10))
    }
    return weights

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

def softmax(x):
    exp_shifted = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_shifted / np.sum(exp_shifted, axis=1, keepdims=True)

def forward_pass(x, weights):
    z1 = np.dot(x, weights['W1']) + weights['b1']
    a1 = relu(z1)
```

```
    z2 = np.dot(a1, weights['W2']) + weights['b2']
    a2 = relu(z2)

    z3 = np.dot(a2, weights['W3']) + weights['b3']
    a3 = softmax(z3)

    # Cache values for backpropagation
    cache = {
        'x': x, 'z1': z1, 'a1': a1,
        'z2': z2, 'a2': a2,
        'z3': z3, 'a3': a3
    }
    return a3, cache

def cross_entropy_loss(y_pred, y_true):
    # Add small epsilon to avoid log(0)
    eps = 1e-10
    y_pred = np.clip(y_pred, eps, 1 - eps)
    loss = -np.sum(y_true * np.log(y_pred)) / y_true.shape[0]
    return loss

def loss_derivative(y_pred, y_true):
    return (y_pred - y_true) / y_true.shape[0]

def backward_pass(weights, cache, y_true):
    x, a1, a2, a3 = cache['x'], cache['a1'], cache['a2'], cache['a3']
    z1, z2 = cache['z1'], cache['z2']

    # Output layer gradient
    dz3 = loss_derivative(a3, y_true)
    dW3 = np.dot(a2.T, dz3)
    db3 = np.sum(dz3, axis=0, keepdims=True)

    # Layer 2 gradients
    da2 = np.dot(dz3, weights['W3'].T)
    dz2 = da2 * relu_derivative(z2)
    dW2 = np.dot(a1.T, dz2)
    db2 = np.sum(dz2, axis=0, keepdims=True)

    # Layer 1 gradients
    da1 = np.dot(dz2, weights['W2'].T)
    dz1 = da1 * relu_derivative(z1)
    dW1 = np.dot(x.T, dz1)
    db1 = np.sum(dz1, axis=0, keepdims=True)

    gradients = {
        'dW1': dW1, 'db1': db1,
        'dW2': dW2, 'db2': db2,
        'dW3': dW3, 'db3': db3
```

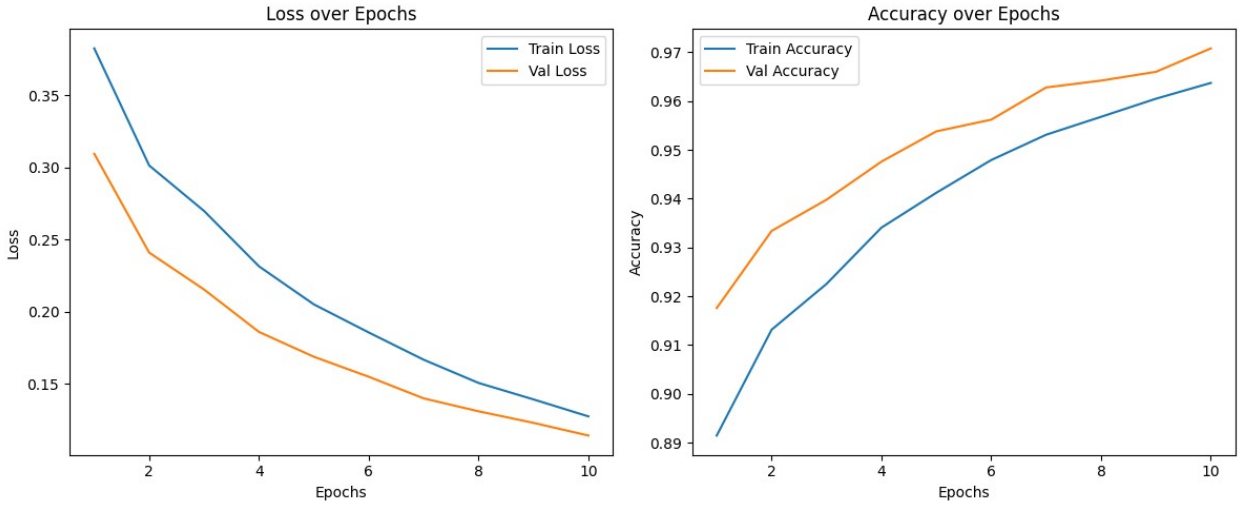
```
    }
    return gradients

def update_weights(weights, gradients, lr):
    for key in weights:
        weights[key] -= lr * gradients['d' + key]
    return weights

def train_model(x_train, y_train, x_val, y_val, epochs=10,
batch_size=32, lr=0.01):
    weights = initialize_weights()
    history = {'train_loss': [], 'val_loss': [], 'train_acc': [],
'val_acc': []}

    for epoch in range(epochs):
        # Shuffle training data
        indices = np.arange(x_train.shape[0])
        np.random.shuffle(indices)
        x_train, y_train = x_train[indices], y_train[indices]

        for i in range(0, x_train.shape[0], batch_size):
            x_batch = x_train[i:i+batch_size]
            y_batch = y_train[i:i+batch_size]

            # Forward
            y_pred, cache = forward_pass(x_batch, weights)

            # Backward
            gradients = backward_pass(weights, cache, y_batch)

            # Update
            weights = update_weights(weights, gradients, lr)

        # Epoch-end evaluation
        train_pred, _ = forward_pass(x_train, weights)
        val_pred, _ = forward_pass(x_val, weights)
        train_loss = cross_entropy_loss(train_pred, y_train)
        val_loss = cross_entropy_loss(val_pred, y_val)
        train_acc = np.mean(np.argmax(train_pred, axis=1) ==
np.argmax(y_train, axis=1))
        val_acc = np.mean(np.argmax(val_pred, axis=1) ==
np.argmax(y_val, axis=1))

        history['train_loss'].append(train_loss)
        history['val_loss'].append(val_loss)
        history['train_acc'].append(train_acc)
        history['val_acc'].append(val_acc)

        print(f"Epoch {epoch+1}/{epochs} - Train Loss:
{train_loss:.4f}, Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f},
```

```
Acc: {val_acc:.4f}")

    return weights, history

def evaluate_model(x_test, y_test, weights):
    y_pred, _ = forward_pass(x_test, weights)
    loss = cross_entropy_loss(y_pred, y_test)
    accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_test,
axis=1))
    print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
    return loss, accuracy

# Split off a small validation set
x_val, y_val = x_train[-5000:], y_train_oh[-5000:]
x_train_sub, y_train_sub = x_train[:-5000], y_train_oh[:-5000]

weights, history = train_model(x_train_sub, y_train_sub, x_val, y_val,
epochs=10, batch_size=32, lr=0.01)

evaluate_model(x_test, y_test_oh, weights)
```

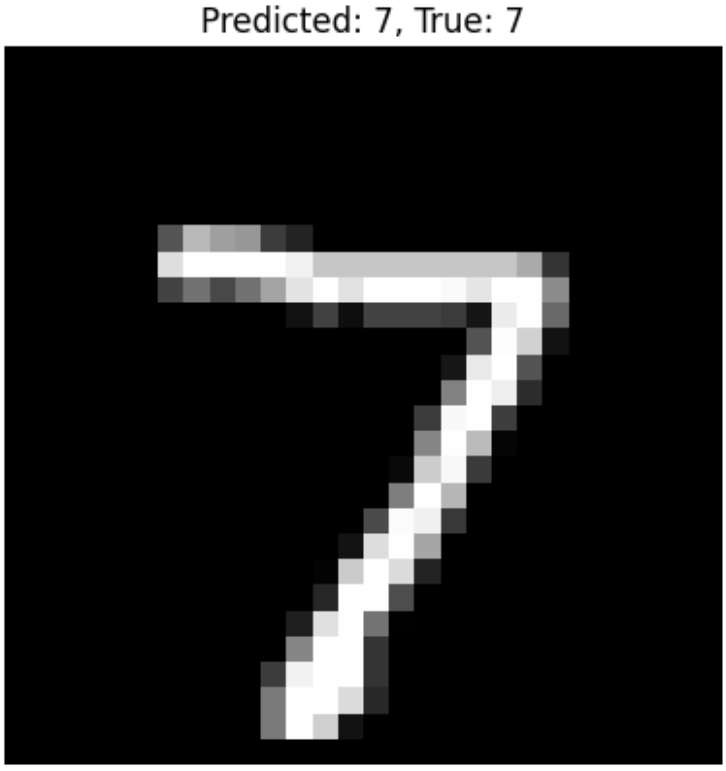
```
Epoch 1/10 - Train Loss: 0.3826, Acc: 0.8915 | Val Loss: 0.3095, Acc:
0.9176
Epoch 2/10 - Train Loss: 0.3014, Acc: 0.9132 | Val Loss: 0.2409, Acc:
0.9334
Epoch 3/10 - Train Loss: 0.2698, Acc: 0.9225 | Val Loss: 0.2153, Acc:
0.9398
Epoch 4/10 - Train Loss: 0.2314, Acc: 0.9341 | Val Loss: 0.1859, Acc:
0.9476
Epoch 5/10 - Train Loss: 0.2050, Acc: 0.9412 | Val Loss: 0.1687, Acc:
0.9538
Epoch 6/10 - Train Loss: 0.1855, Acc: 0.9479 | Val Loss: 0.1548, Acc:
0.9562
Epoch 7/10 - Train Loss: 0.1666, Acc: 0.9531 | Val Loss: 0.1398, Acc:
0.9628
Epoch 8/10 - Train Loss: 0.1505, Acc: 0.9568 | Val Loss: 0.1308, Acc:
0.9642
Epoch 9/10 - Train Loss: 0.1391, Acc: 0.9605 | Val Loss: 0.1228, Acc:
0.9660
Epoch 10/10 - Train Loss: 0.1273, Acc: 0.9637 | Val Loss: 0.1140, Acc:
0.9708
Test Loss: 0.1362, Test Accuracy: 0.9611

(np.float64(0.136242277100814), np.float64(0.9611))
```

```
import matplotlib.pyplot as plt

def plot_metrics(history):
    epochs = range(1, len(history['train_loss']) + 1)

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

```
    # Loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history['train_loss'], label='Train Loss')
    plt.plot(epochs, history['val_loss'], label='Val Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Loss over Epochs')
    plt.legend()

    # Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, history['train_acc'], label='Train Accuracy')
    plt.plot(epochs, history['val_acc'], label='Val Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Accuracy over Epochs')
    plt.legend()

    plt.tight_layout()
    plt.show()

def visualize_predictions(x_test, y_test, weights, num_samples=10):
    import matplotlib.pyplot as plt

    y_pred, _ = forward_pass(x_test, weights)
    pred_labels = np.argmax(y_pred, axis=1)
    true_labels = np.argmax(y_test, axis=1)

    for i in range(num_samples):
        plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
        plt.title(f"Predicted: {pred_labels[i]}, True:
{true_labels[i]}")
        plt.axis('off')
        plt.show()

plot_metrics(history)
```

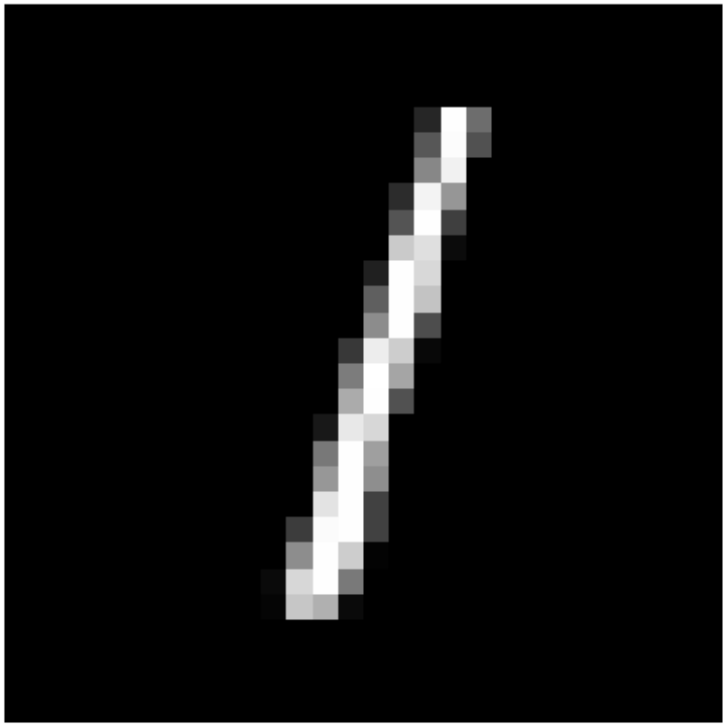



```
visualize_predictions(x_test, y_test_oh, weights)
```

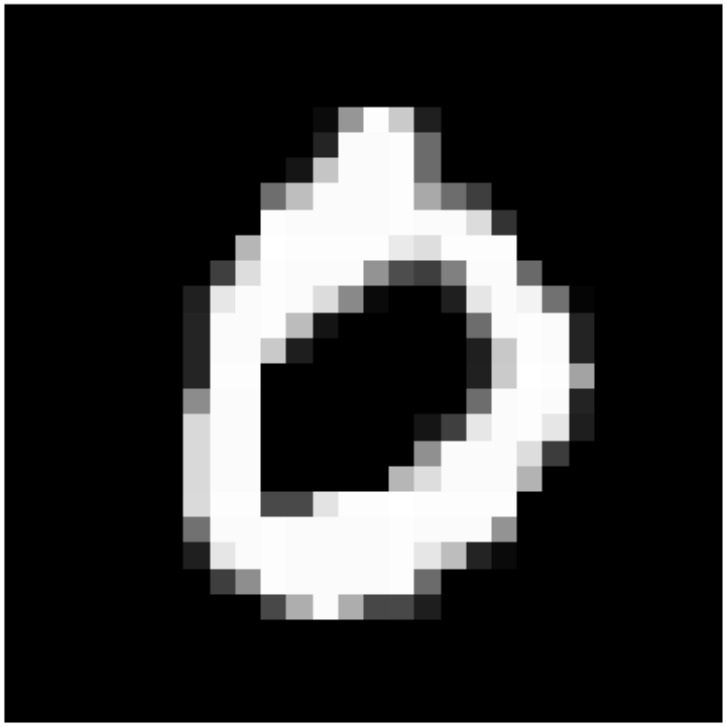


Predicted: 2, True: 2

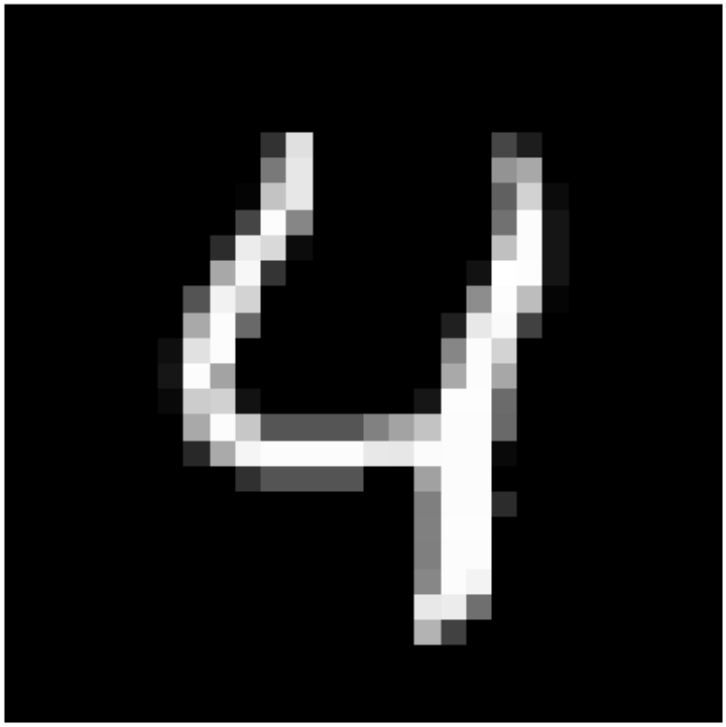
Predicted: 1, True: 1



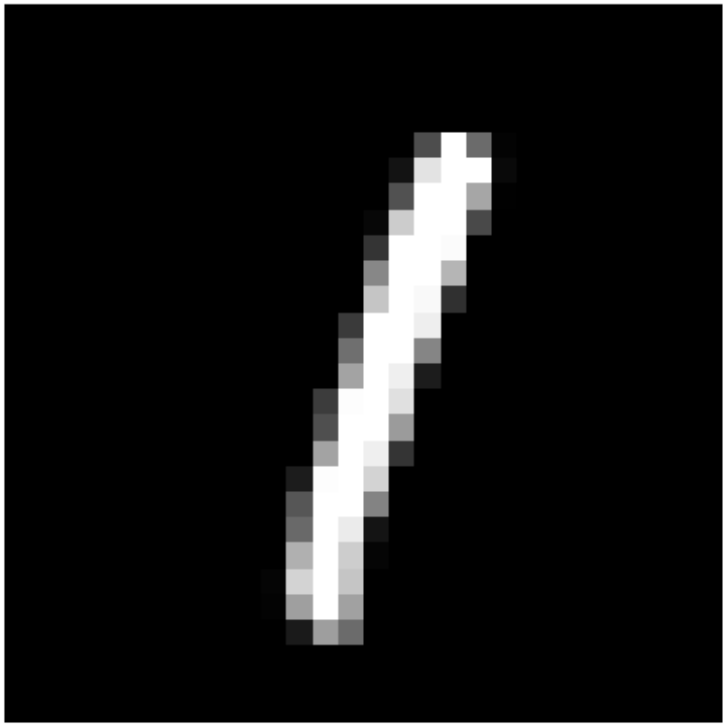
Predicted: 0, True: 0



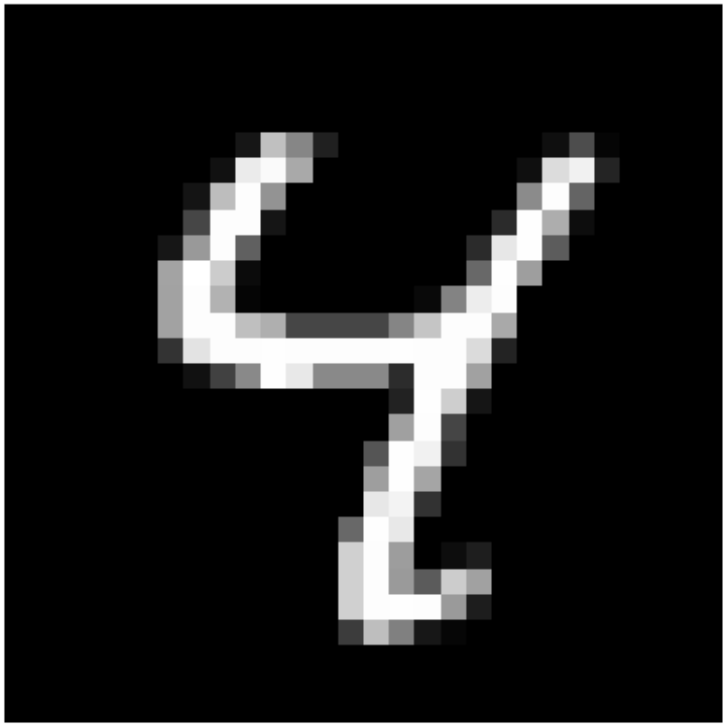
Predicted: 4, True: 4



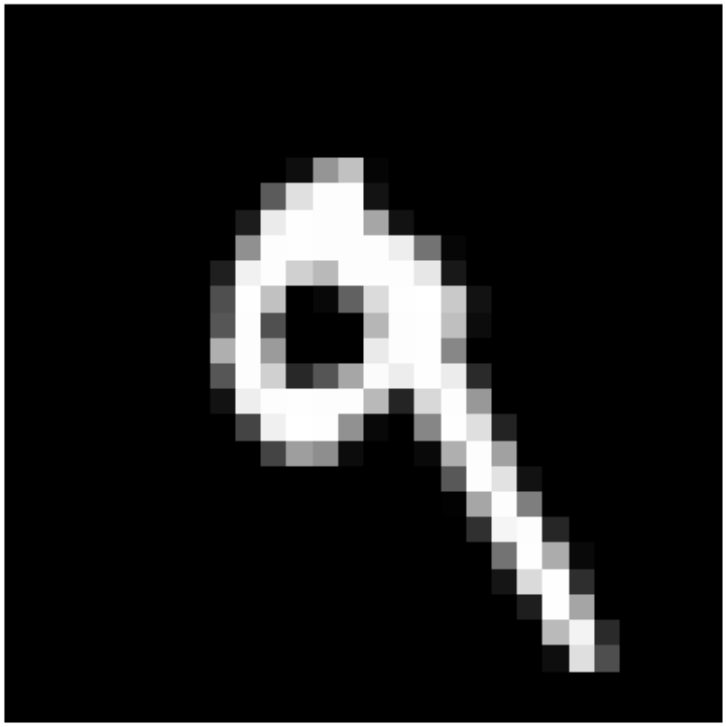
Predicted: 1, True: 1



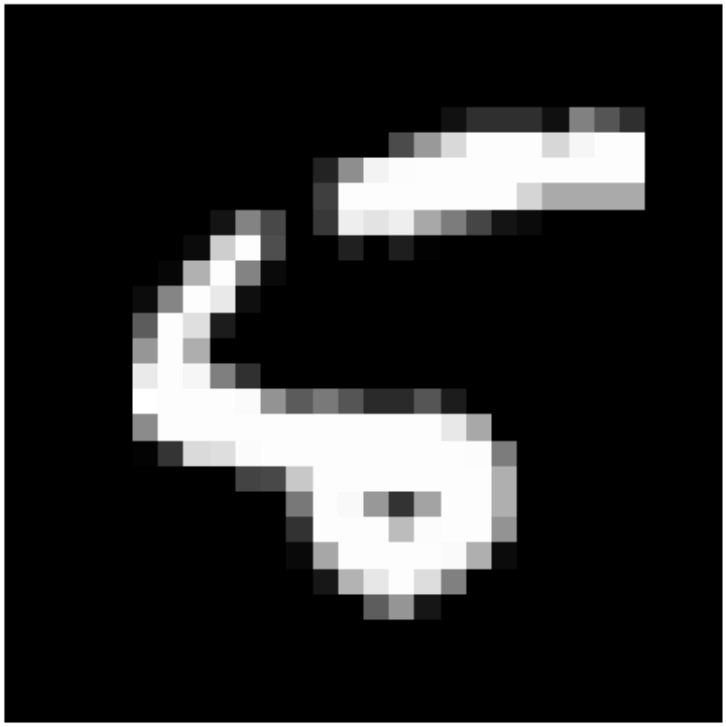
Predicted: 4, True: 4

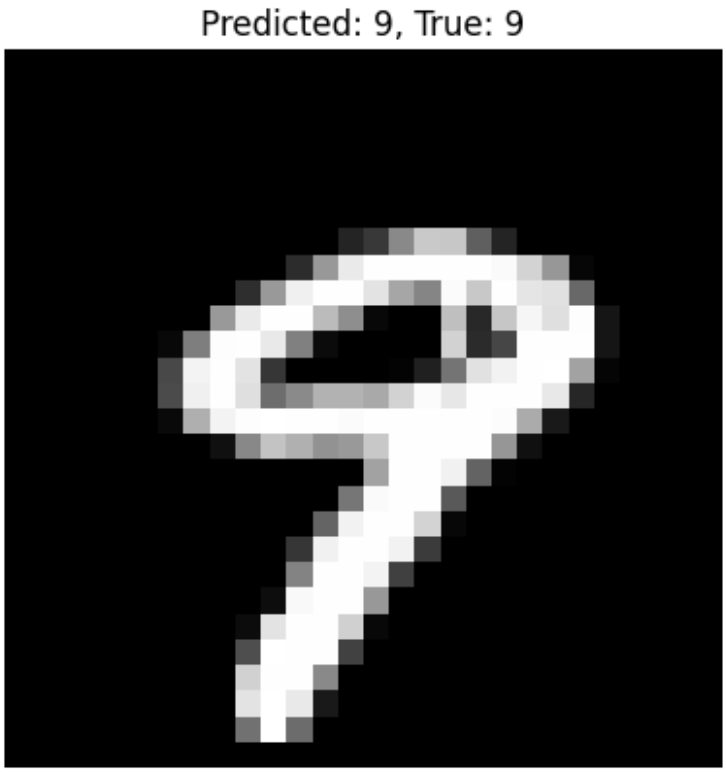


Predicted: 9, True: 9



Predicted: 6, True: 5






```
learning_rates = [0.1, 0.01, 0.001]
batch_sizes = [32, 64, 128]

results = []

for lr in learning_rates:
    for bs in batch_sizes:
        print(f"\nTraining with LR={lr}, Batch Size={bs}")
        weights = initialize_weights()
        weights, history = train_model(x_train_sub, y_train_sub,
x_val, y_val,
                                       epochs=10, batch_size=bs,
lr=lr)
        test_loss, test_acc = evaluate_model(x_test, y_test_oh,
weights)
        results.append((lr, bs, test_acc, test_loss))


Training with LR=0.1, Batch Size=32
Epoch 1/10 - Train Loss: 0.1498, Acc: 0.9543 | Val Loss: 0.1261, Acc:
0.9652
Epoch 2/10 - Train Loss: 0.0849, Acc: 0.9749 | Val Loss: 0.0904, Acc:
0.9736
Epoch 3/10 - Train Loss: 0.0691, Acc: 0.9789 | Val Loss: 0.0860, Acc:
0.9736
Epoch 4/10 - Train Loss: 0.0478, Acc: 0.9856 | Val Loss: 0.0763, Acc:
```

```
0.9780
Epoch 5/10 - Train Loss: 0.0445, Acc: 0.9858 | Val Loss: 0.0774, Acc:
0.9776
Epoch 6/10 - Train Loss: 0.0372, Acc: 0.9879 | Val Loss: 0.0788, Acc:
0.9780
Epoch 7/10 - Train Loss: 0.0261, Acc: 0.9924 | Val Loss: 0.0739, Acc:
0.9800
Epoch 8/10 - Train Loss: 0.0330, Acc: 0.9893 | Val Loss: 0.0834, Acc:
0.9786
Epoch 9/10 - Train Loss: 0.0247, Acc: 0.9920 | Val Loss: 0.0844, Acc:
0.9776
Epoch 10/10 - Train Loss: 0.0123, Acc: 0.9968 | Val Loss: 0.0737, Acc:
0.9822
Test Loss: 0.0692, Test Accuracy: 0.9805

Training with LR=0.1, Batch Size=64
Epoch 1/10 - Train Loss: 0.2251, Acc: 0.9331 | Val Loss: 0.1832, Acc:
0.9492
Epoch 2/10 - Train Loss: 0.1360, Acc: 0.9589 | Val Loss: 0.1219, Acc:
0.9640
Epoch 3/10 - Train Loss: 0.1114, Acc: 0.9668 | Val Loss: 0.1066, Acc:
0.9690
Epoch 4/10 - Train Loss: 0.0831, Acc: 0.9748 | Val Loss: 0.0922, Acc:
0.9716
Epoch 5/10 - Train Loss: 0.0751, Acc: 0.9769 | Val Loss: 0.0915, Acc:
0.9730
Epoch 6/10 - Train Loss: 0.0558, Acc: 0.9835 | Val Loss: 0.0831, Acc:
0.9774
Epoch 7/10 - Train Loss: 0.0435, Acc: 0.9876 | Val Loss: 0.0790, Acc:
0.9790
Epoch 8/10 - Train Loss: 0.0418, Acc: 0.9876 | Val Loss: 0.0801, Acc:
0.9790
Epoch 9/10 - Train Loss: 0.0333, Acc: 0.9901 | Val Loss: 0.0750, Acc:
0.9792
Epoch 10/10 - Train Loss: 0.0377, Acc: 0.9884 | Val Loss: 0.0846, Acc:
0.9786
Test Loss: 0.0911, Test Accuracy: 0.9735

Training with LR=0.1, Batch Size=128
Epoch 1/10 - Train Loss: 0.2871, Acc: 0.9156 | Val Loss: 0.2309, Acc:
0.9370
Epoch 2/10 - Train Loss: 0.2131, Acc: 0.9379 | Val Loss: 0.1744, Acc:
0.9488
Epoch 3/10 - Train Loss: 0.1898, Acc: 0.9449 | Val Loss: 0.1570, Acc:
0.9558
Epoch 4/10 - Train Loss: 0.1364, Acc: 0.9593 | Val Loss: 0.1218, Acc:
0.9666
Epoch 5/10 - Train Loss: 0.1170, Acc: 0.9645 | Val Loss: 0.1115, Acc:
0.9712
```

```
Epoch 6/10 - Train Loss: 0.1005, Acc: 0.9709 | Val Loss: 0.1032, Acc:
0.9712
Epoch 7/10 - Train Loss: 0.0808, Acc: 0.9764 | Val Loss: 0.0890, Acc:
0.9770
Epoch 8/10 - Train Loss: 0.0771, Acc: 0.9774 | Val Loss: 0.0907, Acc:
0.9754
Epoch 9/10 - Train Loss: 0.0694, Acc: 0.9794 | Val Loss: 0.0866, Acc:
0.9750
Epoch 10/10 - Train Loss: 0.0559, Acc: 0.9841 | Val Loss: 0.0785, Acc:
0.9780
Test Loss: 0.0871, Test Accuracy: 0.9715

Training with LR=0.01, Batch Size=32
Epoch 1/10 - Train Loss: 0.3826, Acc: 0.8915 | Val Loss: 0.3095, Acc:
0.9176
Epoch 2/10 - Train Loss: 0.3014, Acc: 0.9132 | Val Loss: 0.2409, Acc:
0.9334
Epoch 3/10 - Train Loss: 0.2698, Acc: 0.9225 | Val Loss: 0.2153, Acc:
0.9398
Epoch 4/10 - Train Loss: 0.2314, Acc: 0.9341 | Val Loss: 0.1859, Acc:
0.9476
Epoch 5/10 - Train Loss: 0.2050, Acc: 0.9412 | Val Loss: 0.1687, Acc:
0.9538
Epoch 6/10 - Train Loss: 0.1855, Acc: 0.9479 | Val Loss: 0.1548, Acc:
0.9562
Epoch 7/10 - Train Loss: 0.1666, Acc: 0.9531 | Val Loss: 0.1398, Acc:
0.9628
Epoch 8/10 - Train Loss: 0.1505, Acc: 0.9568 | Val Loss: 0.1308, Acc:
0.9642
Epoch 9/10 - Train Loss: 0.1391, Acc: 0.9605 | Val Loss: 0.1228, Acc:
0.9660
Epoch 10/10 - Train Loss: 0.1273, Acc: 0.9637 | Val Loss: 0.1140, Acc:
0.9708
Test Loss: 0.1362, Test Accuracy: 0.9611

Training with LR=0.01, Batch Size=64
Epoch 1/10 - Train Loss: 0.5169, Acc: 0.8623 | Val Loss: 0.4378, Acc:
0.8938
Epoch 2/10 - Train Loss: 0.3795, Acc: 0.8920 | Val Loss: 0.3061, Acc:
0.9170
Epoch 3/10 - Train Loss: 0.3371, Acc: 0.9032 | Val Loss: 0.2697, Acc:
0.9276
Epoch 4/10 - Train Loss: 0.3040, Acc: 0.9121 | Val Loss: 0.2408, Acc:
0.9342
Epoch 5/10 - Train Loss: 0.2785, Acc: 0.9193 | Val Loss: 0.2218, Acc:
0.9390
Epoch 6/10 - Train Loss: 0.2640, Acc: 0.9247 | Val Loss: 0.2115, Acc:
0.9408
Epoch 7/10 - Train Loss: 0.2443, Acc: 0.9316 | Val Loss: 0.1964, Acc:
```

```
0.9448
Epoch 8/10 - Train Loss: 0.2289, Acc: 0.9354 | Val Loss: 0.1857, Acc:
0.9464
Epoch 9/10 - Train Loss: 0.2170, Acc: 0.9381 | Val Loss: 0.1768, Acc:
0.9502
Epoch 10/10 - Train Loss: 0.2033, Acc: 0.9426 | Val Loss: 0.1660, Acc:
0.9528
Test Loss: 0.2032, Test Accuracy: 0.9419

Training with LR=0.01, Batch Size=128
Epoch 1/10 - Train Loss: 0.8692, Acc: 0.8121 | Val Loss: 0.8052, Acc:
0.8550
Epoch 2/10 - Train Loss: 0.5155, Acc: 0.8624 | Val Loss: 0.4360, Acc:
0.8952
Epoch 3/10 - Train Loss: 0.4240, Acc: 0.8819 | Val Loss: 0.3473, Acc:
0.9082
Epoch 4/10 - Train Loss: 0.3789, Acc: 0.8914 | Val Loss: 0.3047, Acc:
0.9184
Epoch 5/10 - Train Loss: 0.3500, Acc: 0.8997 | Val Loss: 0.2801, Acc:
0.9248
Epoch 6/10 - Train Loss: 0.3315, Acc: 0.9053 | Val Loss: 0.2647, Acc:
0.9294
Epoch 7/10 - Train Loss: 0.3144, Acc: 0.9104 | Val Loss: 0.2520, Acc:
0.9330
Epoch 8/10 - Train Loss: 0.3011, Acc: 0.9133 | Val Loss: 0.2404, Acc:
0.9346
Epoch 9/10 - Train Loss: 0.2911, Acc: 0.9160 | Val Loss: 0.2327, Acc:
0.9374
Epoch 10/10 - Train Loss: 0.2781, Acc: 0.9204 | Val Loss: 0.2215, Acc:
0.9386
Test Loss: 0.2685, Test Accuracy: 0.9256

Training with LR=0.001, Batch Size=32
Epoch 1/10 - Train Loss: 1.8089, Acc: 0.6567 | Val Loss: 1.7919, Acc:
0.6876
Epoch 2/10 - Train Loss: 1.0857, Acc: 0.7814 | Val Loss: 1.0336, Acc:
0.8204
Epoch 3/10 - Train Loss: 0.7348, Acc: 0.8278 | Val Loss: 0.6634, Acc:
0.8670
Epoch 4/10 - Train Loss: 0.5908, Acc: 0.8480 | Val Loss: 0.5128, Acc:
0.8812
Epoch 5/10 - Train Loss: 0.5152, Acc: 0.8619 | Val Loss: 0.4360, Acc:
0.8944
Epoch 6/10 - Train Loss: 0.4684, Acc: 0.8720 | Val Loss: 0.3900, Acc:
0.9026
Epoch 7/10 - Train Loss: 0.4361, Acc: 0.8801 | Val Loss: 0.3589, Acc:
0.9080
Epoch 8/10 - Train Loss: 0.4116, Acc: 0.8852 | Val Loss: 0.3353, Acc:
0.9138
```

```
Epoch 9/10 - Train Loss: 0.3932, Acc: 0.8895 | Val Loss: 0.3189, Acc:
0.9156
Epoch 10/10 - Train Loss: 0.3775, Acc: 0.8936 | Val Loss: 0.3047, Acc:
0.9188
Test Loss: 0.3576, Test Accuracy: 0.9001

Training with LR=0.001, Batch Size=64
Epoch 1/10 - Train Loss: 2.1179, Acc: 0.4674 | Val Loss: 2.1124, Acc:
0.4806
Epoch 2/10 - Train Loss: 1.8077, Acc: 0.6574 | Val Loss: 1.7906, Acc:
0.6926
Epoch 3/10 - Train Loss: 1.4188, Acc: 0.7367 | Val Loss: 1.3841, Acc:
0.7718
Epoch 4/10 - Train Loss: 1.0843, Acc: 0.7832 | Val Loss: 1.0325, Acc:
0.8222
Epoch 5/10 - Train Loss: 0.8670, Acc: 0.8116 | Val Loss: 0.8029, Acc:
0.8496
Epoch 6/10 - Train Loss: 0.7341, Acc: 0.8282 | Val Loss: 0.6627, Acc:
0.8676
Epoch 7/10 - Train Loss: 0.6489, Acc: 0.8399 | Val Loss: 0.5733, Acc:
0.8750
Epoch 8/10 - Train Loss: 0.5902, Acc: 0.8490 | Val Loss: 0.5123, Acc:
0.8820
Epoch 9/10 - Train Loss: 0.5477, Acc: 0.8572 | Val Loss: 0.4692, Acc:
0.8890
Epoch 10/10 - Train Loss: 0.5151, Acc: 0.8625 | Val Loss: 0.4365, Acc:
0.8940
Test Loss: 0.4902, Test Accuracy: 0.8706

Training with LR=0.001, Batch Size=128
Epoch 1/10 - Train Loss: 2.2303, Acc: 0.2396 | Val Loss: 2.2291, Acc:
0.2480
Epoch 2/10 - Train Loss: 2.1177, Acc: 0.4651 | Val Loss: 2.1123, Acc:
0.4778
Epoch 3/10 - Train Loss: 1.9775, Acc: 0.5918 | Val Loss: 1.9670, Acc:
0.6188
Epoch 4/10 - Train Loss: 1.8072, Acc: 0.6573 | Val Loss: 1.7900, Acc:
0.6938
Epoch 5/10 - Train Loss: 1.6150, Acc: 0.7062 | Val Loss: 1.5894, Acc:
0.7400
Epoch 6/10 - Train Loss: 1.4184, Acc: 0.7356 | Val Loss: 1.3838, Acc:
0.7726
Epoch 7/10 - Train Loss: 1.2372, Acc: 0.7602 | Val Loss: 1.1933, Acc:
0.7994
Epoch 8/10 - Train Loss: 1.0838, Acc: 0.7844 | Val Loss: 1.0318, Acc:
0.8230
Epoch 9/10 - Train Loss: 0.9619, Acc: 0.8009 | Val Loss: 0.9032, Acc:
0.8398
Epoch 10/10 - Train Loss: 0.8668, Acc: 0.8119 | Val Loss: 0.8028, Acc:
```

```
0.8510
Test Loss: 0.8374, Test Accuracy: 0.8229
```

```
print("\nSummary of Hyperparameter Tuning:")
for lr, bs, acc, loss in results:
    print(f"LR={lr}, Batch Size={bs} → Test Acc: {acc:.4f}, Test Loss:
{loss:.4f}")
```

```
Summary of Hyperparameter Tuning:
LR=0.1, Batch Size=32 → Test Acc: 0.9805, Test Loss: 0.0692
LR=0.1, Batch Size=64 → Test Acc: 0.9735, Test Loss: 0.0911
LR=0.1, Batch Size=128 → Test Acc: 0.9715, Test Loss: 0.0871
LR=0.01, Batch Size=32 → Test Acc: 0.9611, Test Loss: 0.1362
LR=0.01, Batch Size=64 → Test Acc: 0.9419, Test Loss: 0.2032
LR=0.01, Batch Size=128 → Test Acc: 0.9256, Test Loss: 0.2685
LR=0.001, Batch Size=32 → Test Acc: 0.9001, Test Loss: 0.3576
LR=0.001, Batch Size=64 → Test Acc: 0.8706, Test Loss: 0.4902
LR=0.001, Batch Size=128 → Test Acc: 0.8229, Test Loss: 0.8374
```