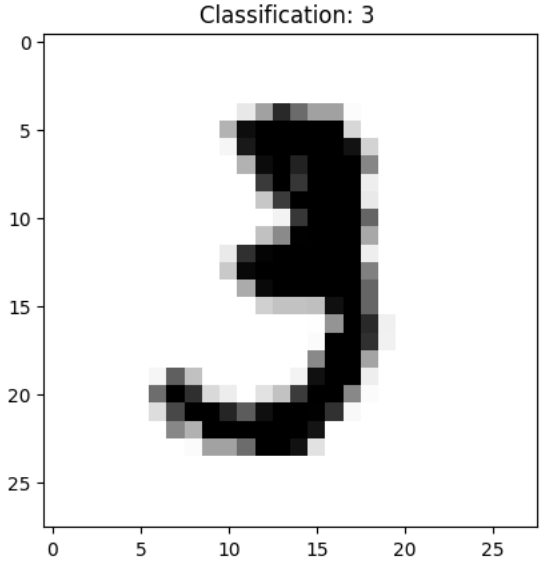
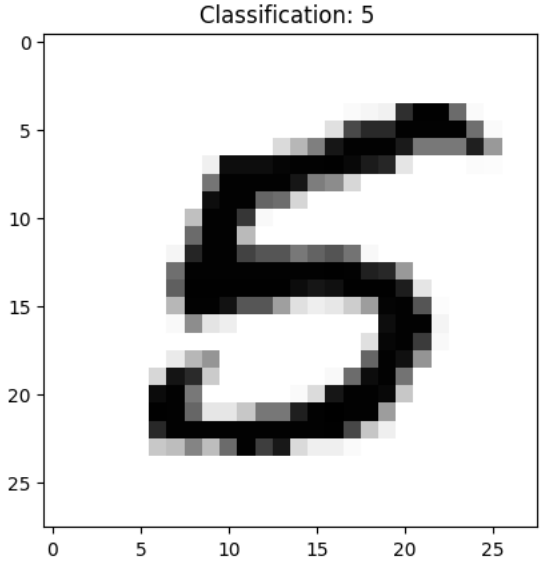
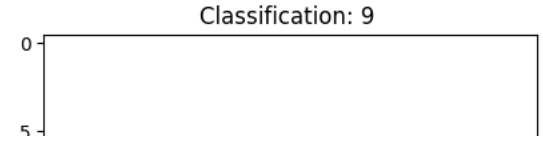
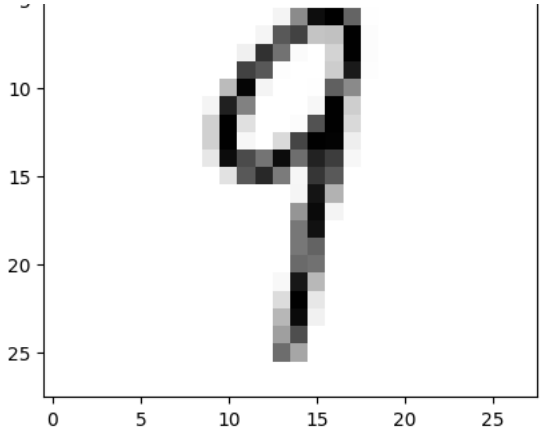
```
import numpy as np
import matplotlib.pyplot as plt


def get_mnist():
    with np.load("mnist.npz") as f:
        images, classes = f["x_train"], f["y_train"]
    images = images.astype("float32") / 255
    images = np.reshape(images, (images.shape[0], images.shape[1] * images.shape[2]))
    classes = np.eye(10)[classes]
    return images, classes


def nn_digit_classifier():
    imgs, classes = get_mnist()
    w_i_h = np.random.uniform(-0.5, 0.5, (20, 784))
    w_h_o = np.random.uniform(-0.5, 0.5, (10, 20))
    b_i_h = np.zeros((20, 1))
    b_h_o = np.zeros((10, 1))

    learn_rate = 0.01
    epochs = 3
    num_correct = 0

    # Train nn classifier to obtain input & hidden layer weights & biases
    for epoch in range(epochs):
        for i, l in zip(imgs, classes):
            i.shape += (1,)
            l.shape += (1,)
            # Forward propagation input -> hidden
            h_pre = b_i_h + np.dot(w_i_h, i)
            h = 1 / (1 + np.exp(-h_pre))
            # Forward propagation hidden -> output
            o_pre = b_h_o + np.dot(w_h_o, h)
            o = 1 / (1 + np.exp(-o_pre))

            num_correct += int(np.argmax(o) == np.argmax(l))

            # Backpropagation output -> hidden (cost function derivative)
            delta_o = o - l
            w_h_o += -learn_rate * np.dot(delta_o, np.transpose(h))
            b_h_o += -learn_rate * delta_o
            # Backpropagation hidden -> input (activation function derivative)
            delta_h = np.dot(np.transpose(w_h_o), delta_o) * (h * (1 - h))
            w_i_h += -learn_rate * np.dot(delta_h, np.transpose(i))
            b_i_h += -learn_rate * delta_h

        # Show accuracy for epoch
        print(f"Accuracy: {round((num_correct / imgs.shape[0]) * 100, 2)}%")
        num_correct = 0

    # Show results
    while True:
        idx = int(input("Enter a number (0 - 59999): "))
        img = imgs[idx]
        plt.imshow(img.reshape(28, 28), cmap="Greys")

        img.shape += (1,)
        # Forward propagation input -> hidden
        h_pre = b_i_h + np.dot(w_i_h, img.reshape(784, 1))
        h = 1 / (1 + np.exp(-h_pre))
        # Forward propagation hidden -> output
        o_pre = b_h_o + np.dot(w_h_o, h)
        o = 1 / (1 + np.exp(-o_pre))

        plt.title(f"Classification: {o.argmax()} ")
        plt.show()


nn_digit_classifier()
```

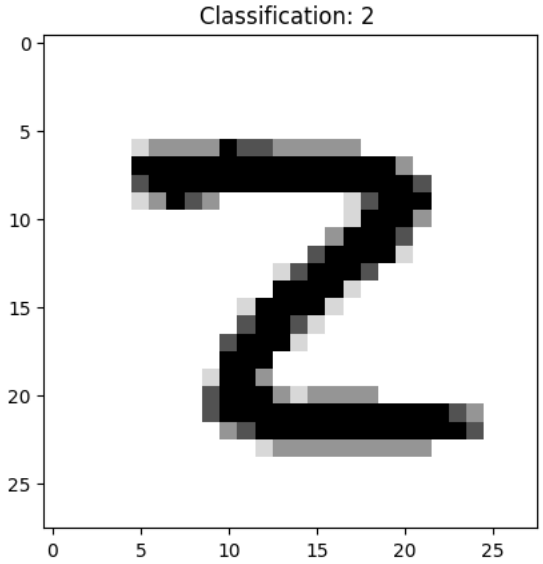
```
Accuracy: 86.28%
Accuracy: 92.45%
Accuracy: 93.5%
```








10
15
20
25
0
5
10
15
20
25


Classification: 2
0
5
10
15
20
25
0
5
10
15
20
25