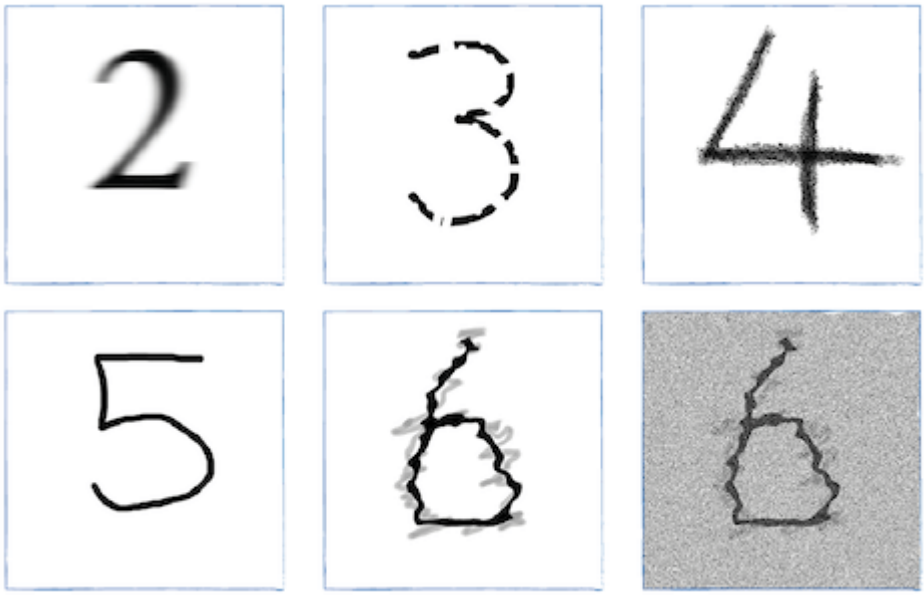
Your Own Handwriting

In this experiment, we will create our test dataset using our own handwriting. We'll also try using different styles of writing, and noisy or shaky images to see how well our neural network copes.

Throughout this guide, we have been using images of handwritten numbers from the MNIST dataset. Why not use your own handwriting? You can create images using any image editing or painting software you like. You don't have to use the expensive Photoshop, the GIMP is a free open source alternative available for Windows, Mac, and Linux. You can even use a pen on paper and photograph your writing with a smartphone or camera, or even use a proper scanner. The only requirement is that the image is square (the width is the same as the length) and you save it as PNG format. You'll often find the saving format option under File > Save As, or File > Export in your favorite image editor. Here are some images I made.



The number 5 is simply my own handwriting. The 4 is done using a chalk rather than a marker. The number 3 is my own handwriting but deliberately with bits chopped out. The 2 is a very traditional newspaper or book typeface but blurred a bit. The 6 is a deliberately wobbly shaky image, almost like a reflection in the water. The last image is the same as the previous one but with noise added to see if we can make the neural network's job even harder!

This is fun, but there is a serious point here. Scientists have been amazed at the human brain's ability to continue to function amazingly well after suffering damage. The suggestion is that neural networks distribute what they've learned across several link weights, which means if they suffer some damage, they can perform fairly well. This also means that they can perform fairly well if the input image is damaged or incomplete. That's a powerful thing to have. That's what we want to test with the chopped up 3 in the image set above. We'll need to create smaller versions of these PNG images rescaled to 28 by 28 pixels, to match what we've used from the MNIST data. You can use your image editor to do this. Python libraries again help us out with reading and decoding the data from common image file formats, including the PNG format. Have a look at the following simple code:

```
import scipy.misc
img_array = scipy.misc.imread(image_file_name, flatten=True)

img_data  = 255.0 - img_array.reshape(784)
img_data = (img_data / 255.0 * 0.99) + 0.01
```

The `scipy.misc.imread()` function is the one that helps us get data out of image files such as PNG or JPG files. We have to import the `scipy.misc` library to use it. The "flatten=True" parameter turns the image into a simple array of floating point numbers, and if the image were colored, the color values would be flattened into grey scale, which is what we need. The next line reshapes the array from a $28 * 28$ square into a long list of values, which is what we need to feed to our neural network. We've done that many times before. What is new is the subtraction of the array's values from 255.0. The reason for this is that it is conventional for 0 to mean black and 255 to mean white, but the MNIST data set has this the opposite way around, so we have to reverse the values to match what the MNIST data does. The last line does the familiar rescaling of the data values, so they range from 0.01 to 1.0. Sample code to demonstrate the reading of PNG files is always online at GitHub:

- https://github.com/makeyourownneuralnetwork/makeyourownneuralnetwork/blob/master/part3_load_own_images.ipynb

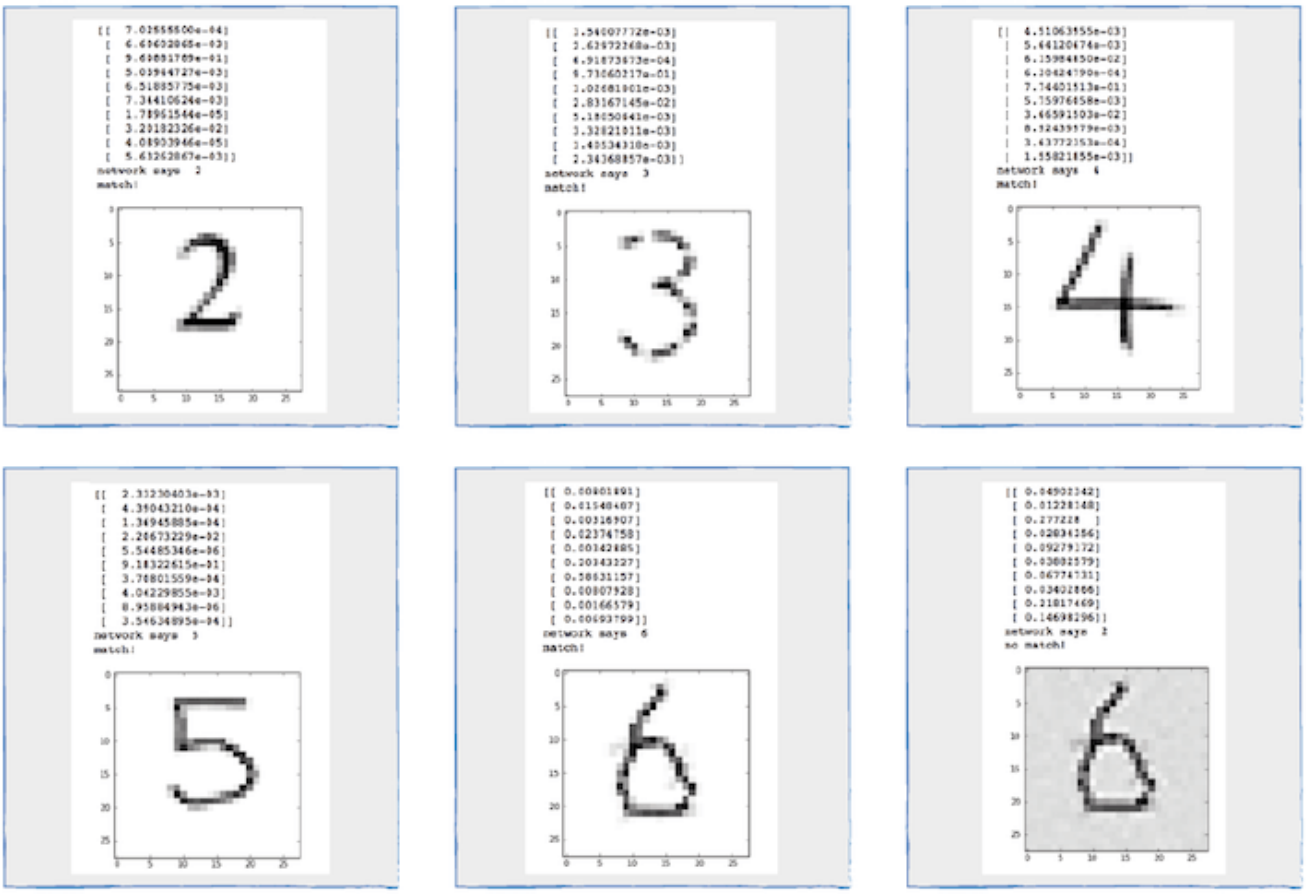
We need to create a version of our basic neural network program that trains on the MNIST training data set but instead of testing with the MNIST test set, it

on the MNIST training data set but instead of testing with the MNIST test set, it

tests against data created from our own images. The new program is online at GitHub:

- https://github.com/makeyourownneuralnetwork/makeyourownneuralnetwork/blob/master/part3_neural_network_mnist_and_own_data.ipynb

Does it work? It does! The following summarizes the results of querying with our own images.



You can see that the neural network recognized all of the images we created, including the deliberately damaged 3. Only the 6 with added noise failed. Try it with your own images, especially handwritten, to prove to yourself that the neural network really does work. And see how far you can go with damaged or deformed images. You'll be impressed with how resilient the neural network is.