ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

CS498
Applied Machine Learning
Assignment #7

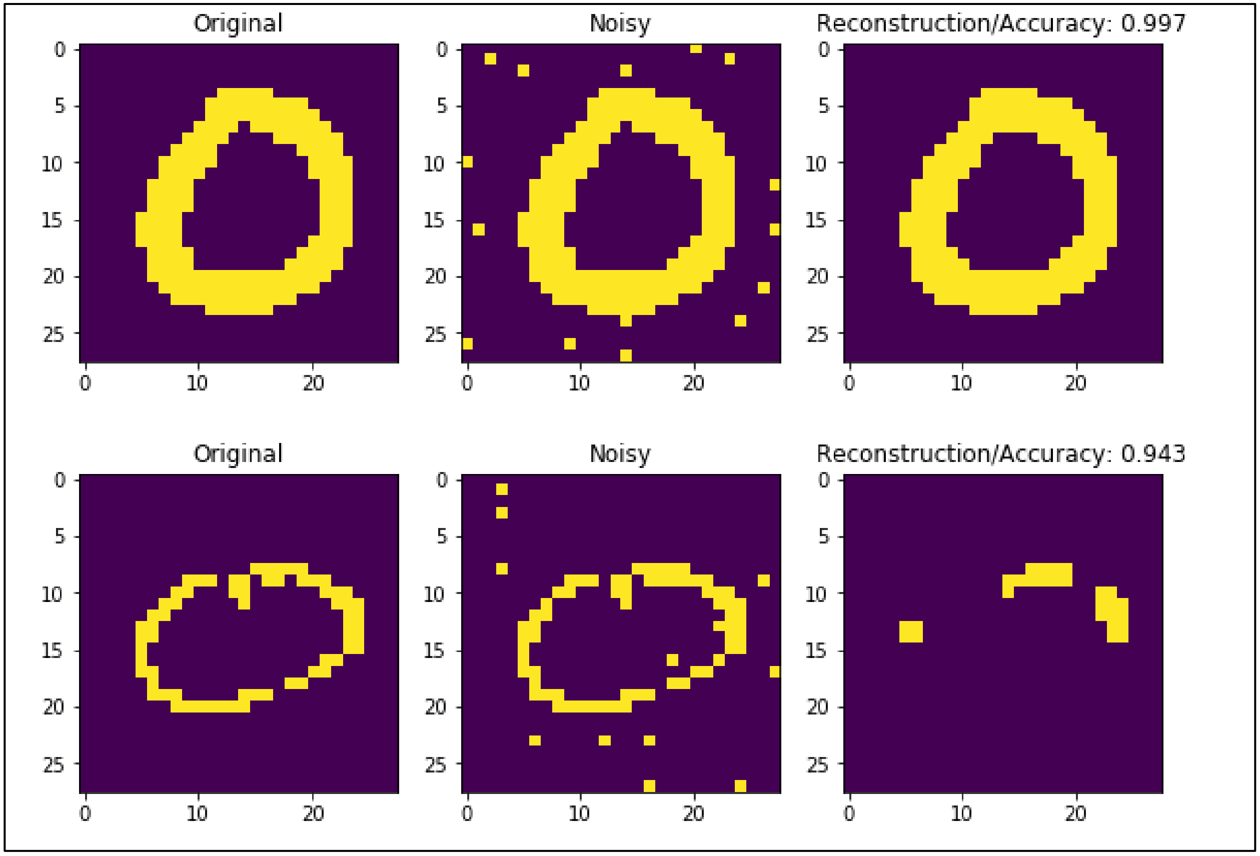
Students:
nidiaib2, Nidia Bucarelli
sunnyk2, Sunny Katiyar
wangx2, Wang Xiang

May 14, 2020
Spring 2020

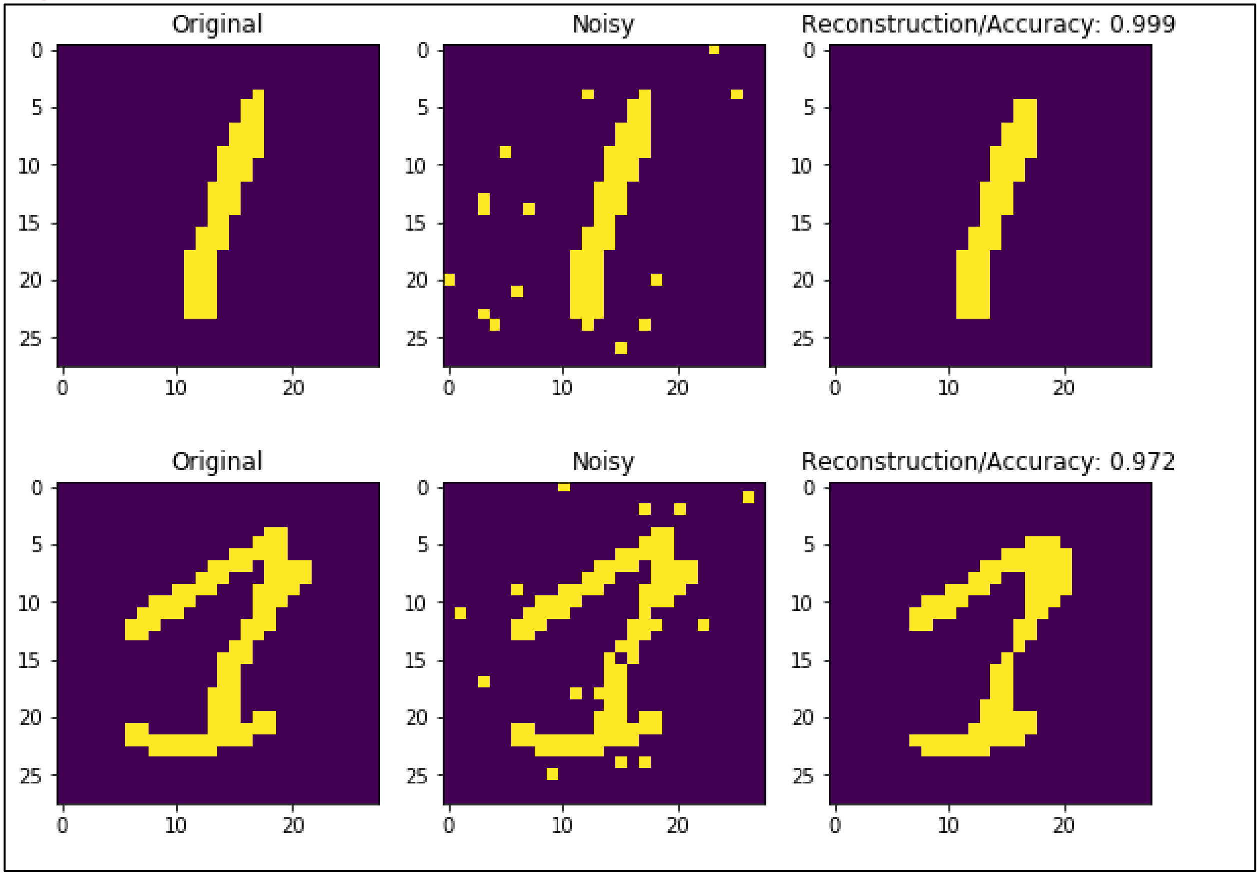
RESULTS

- Fraction of correct pixels: **0.9811**
- The best and worst reconstruction of each number (0-9):

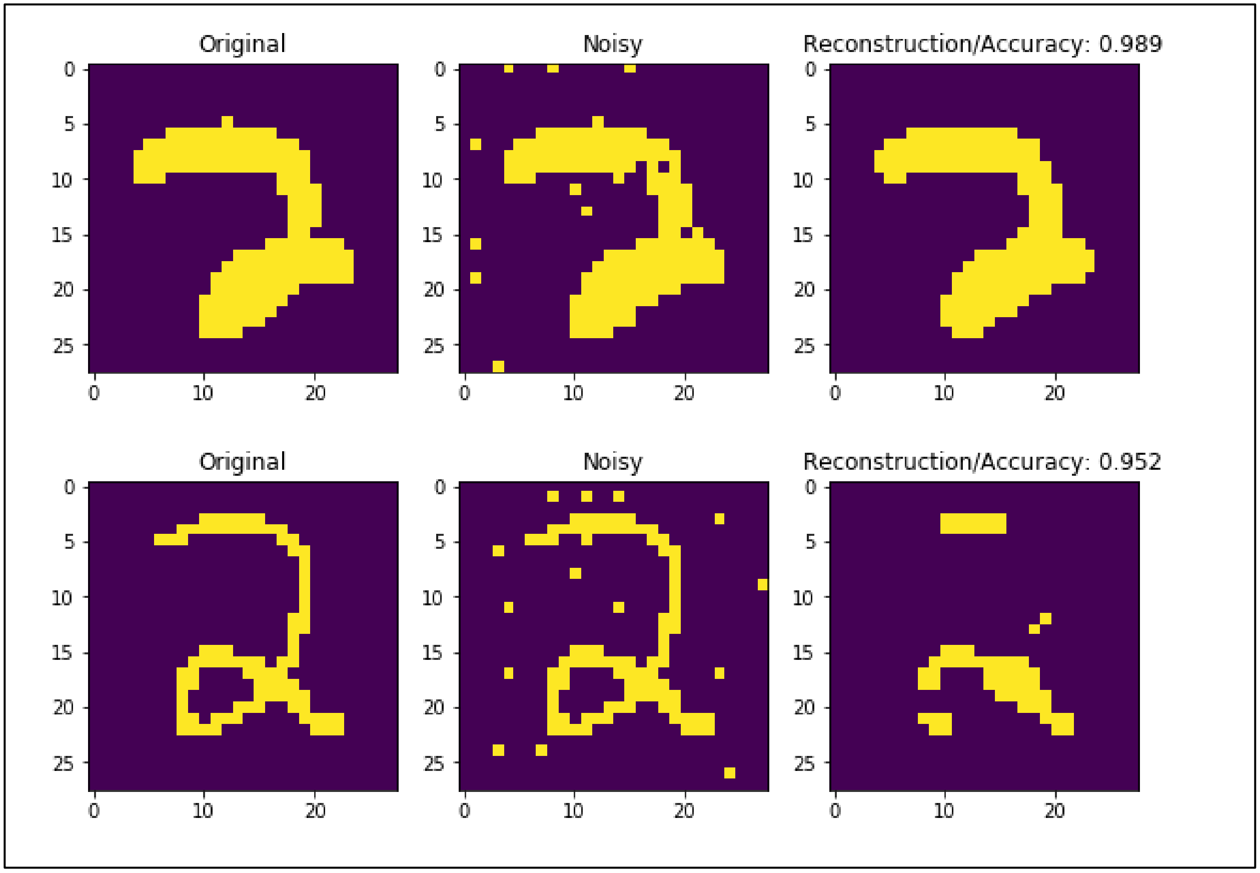
Digit 0 – Best and Worst reconstruction



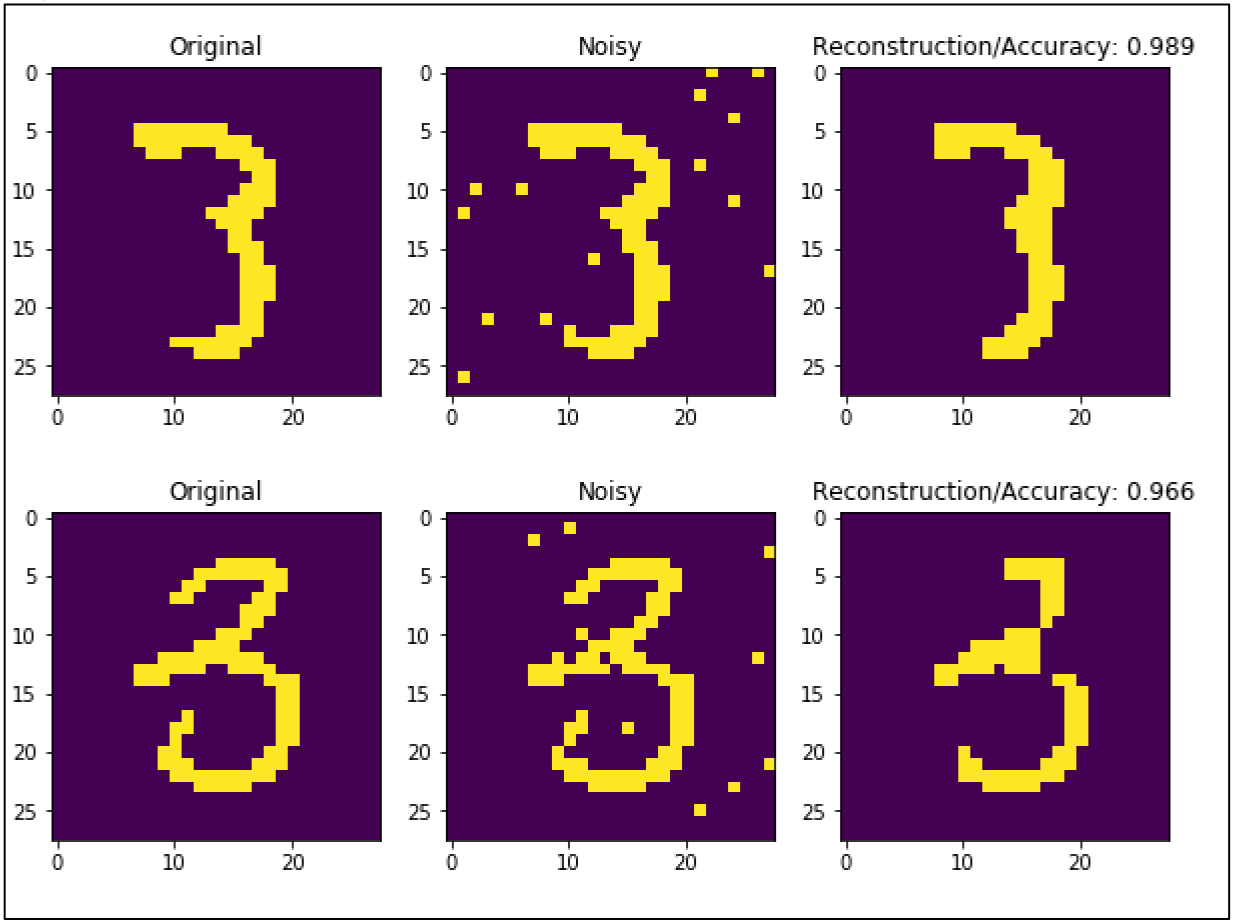
Digit 1 – Best and Worst reconstruction



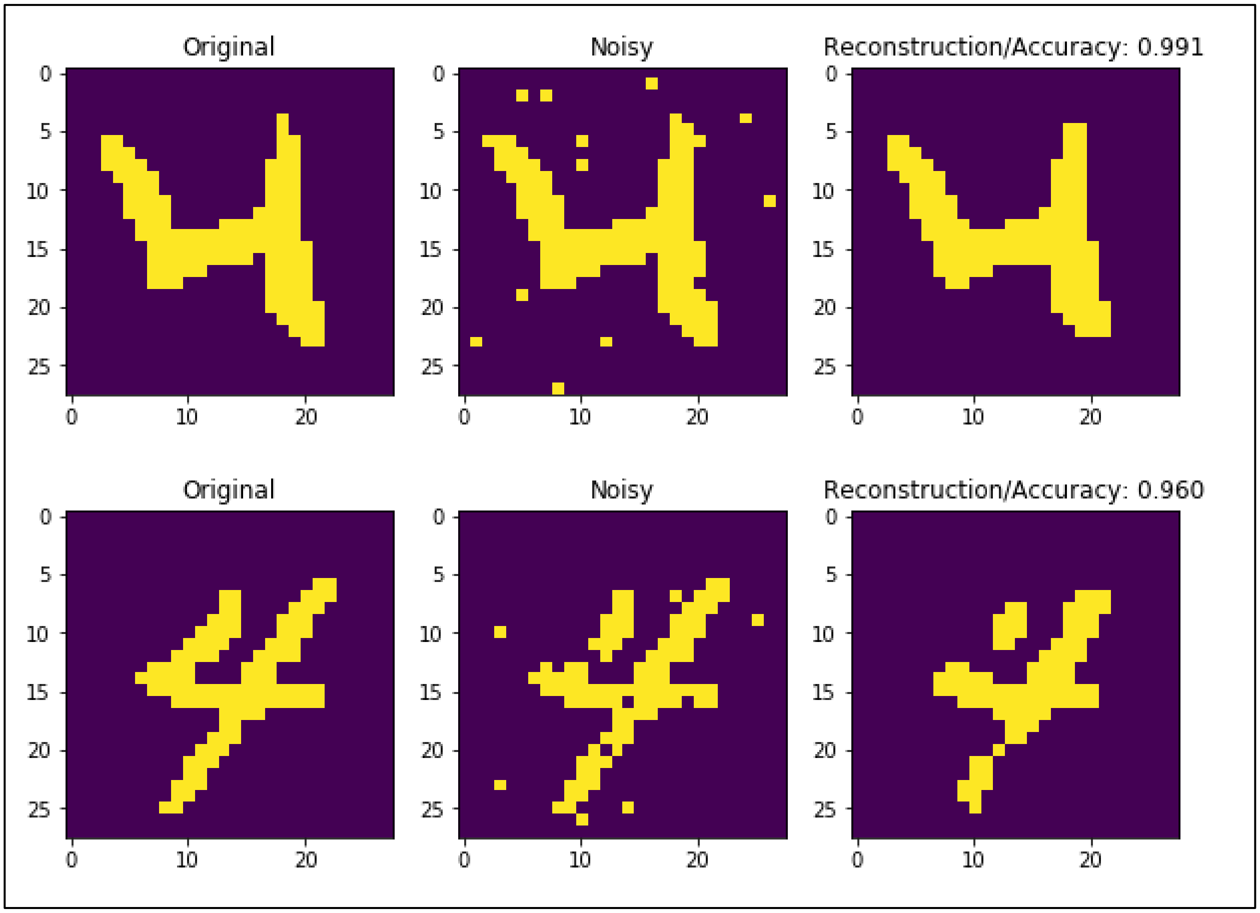
Digit 2 – Best and Worst reconstruction



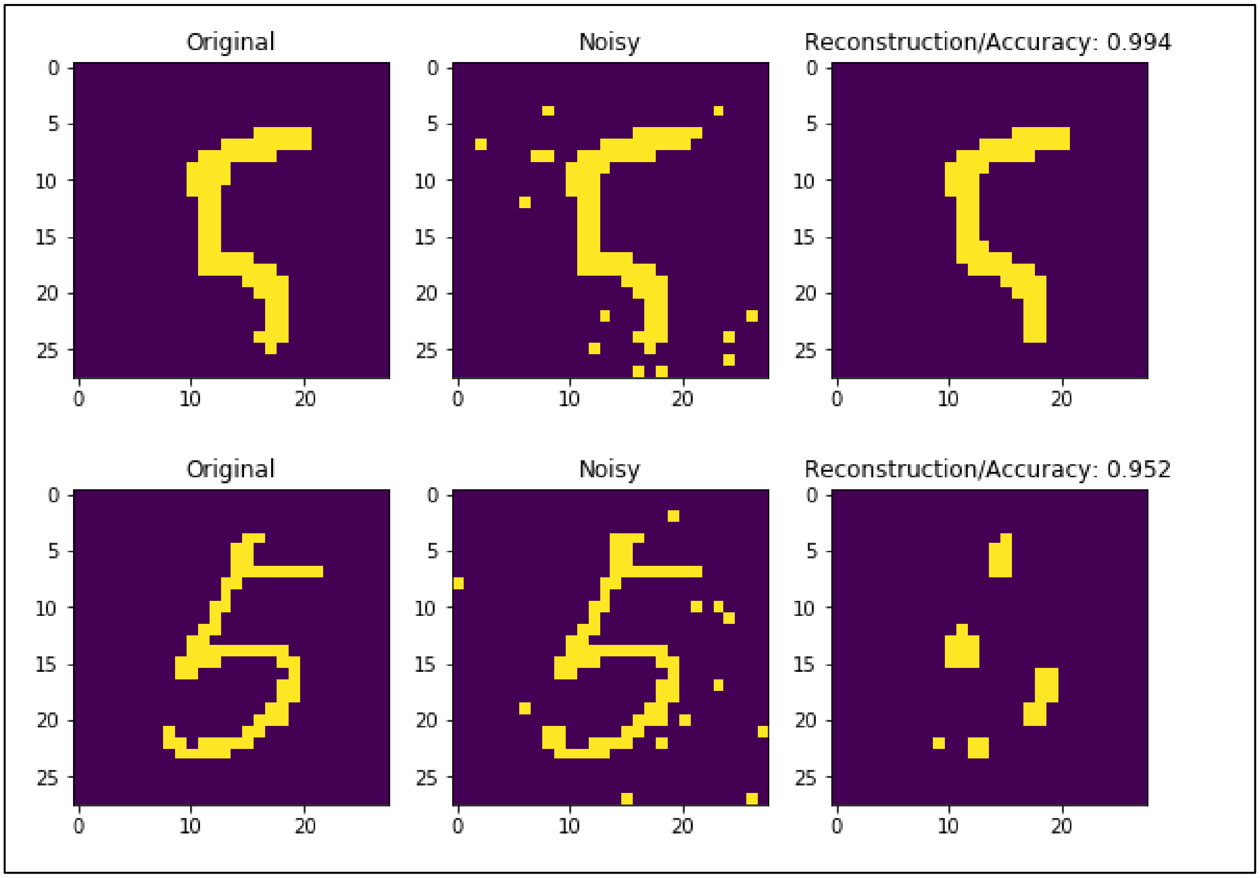
Digit 3 – Best and Worst reconstruction



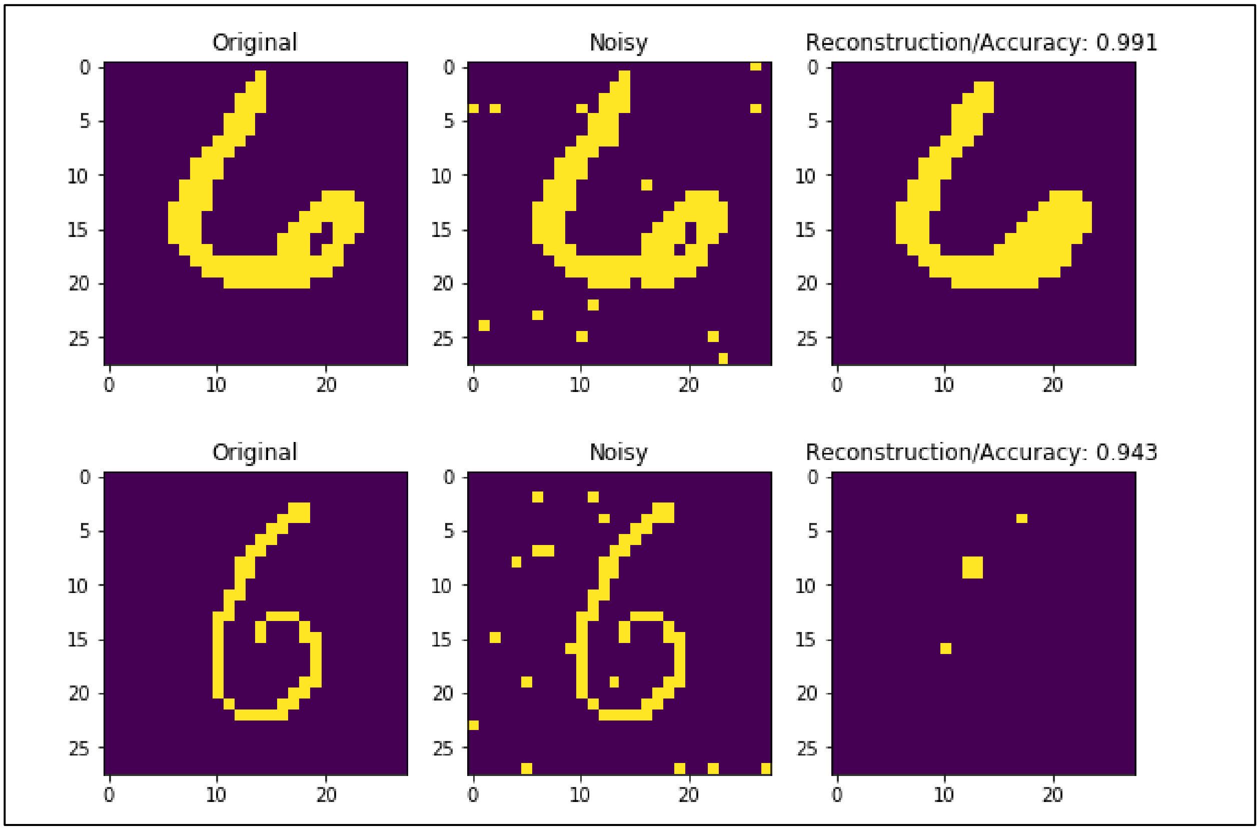
Digit 4 – Best and Worst reconstruction



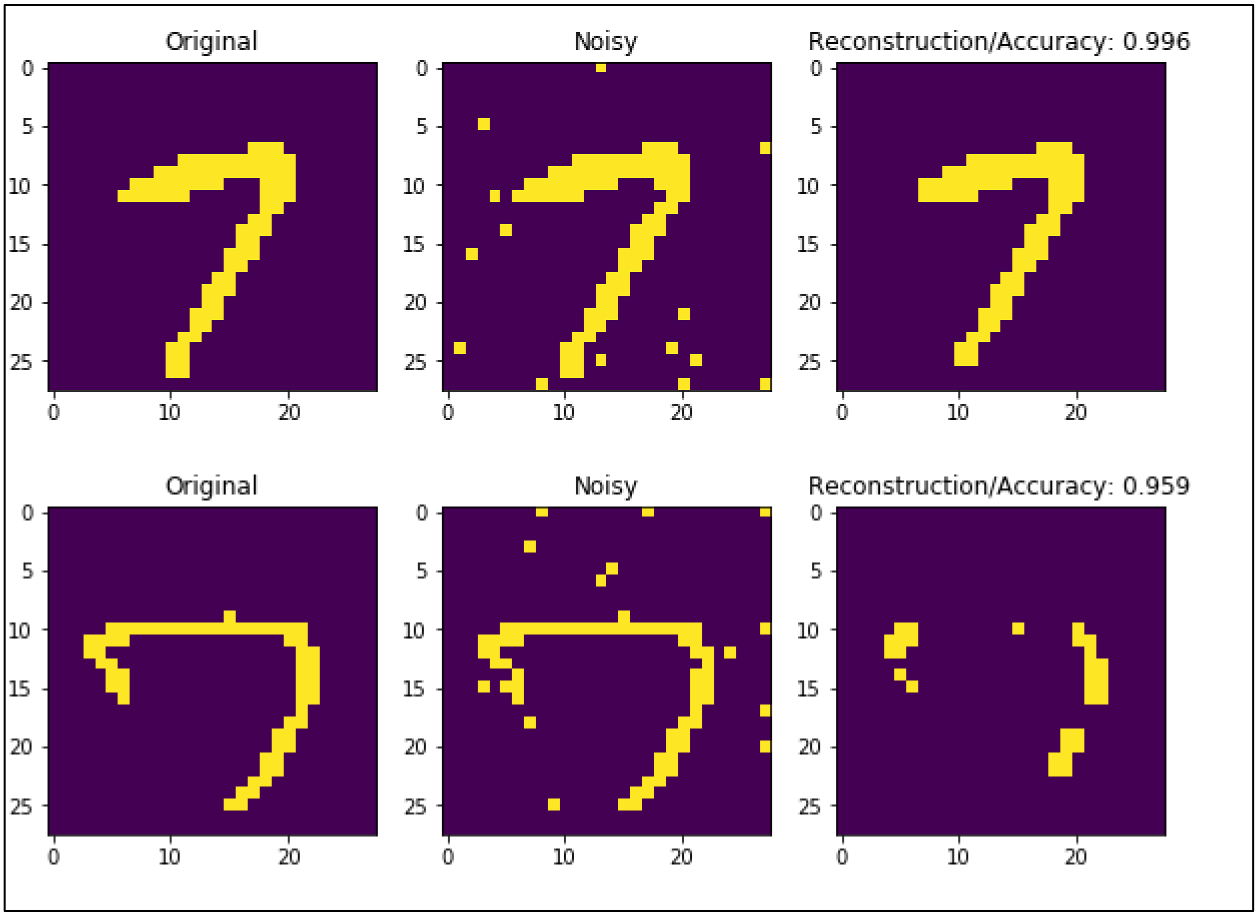
Digit 5 – Best and Worst reconstruction



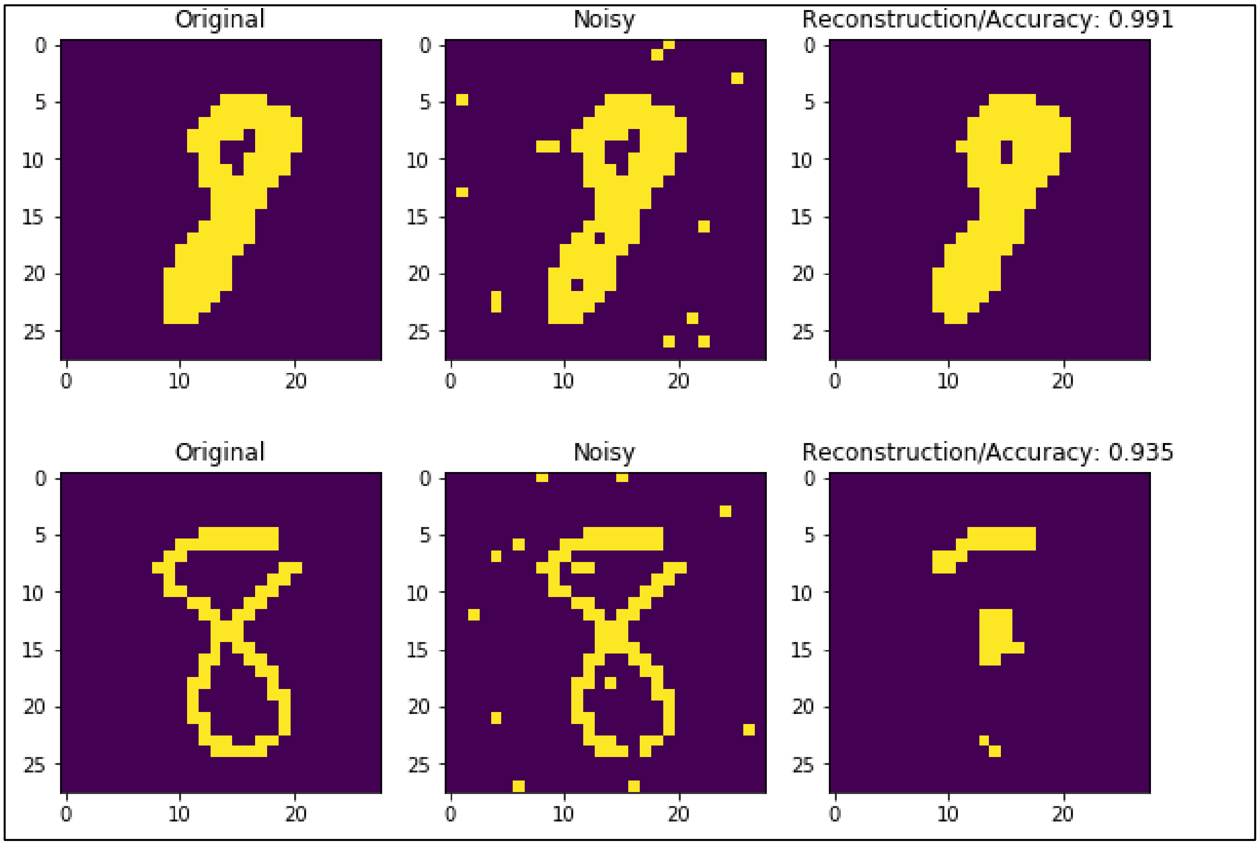
Digit 6 – Best and Worst reconstruction



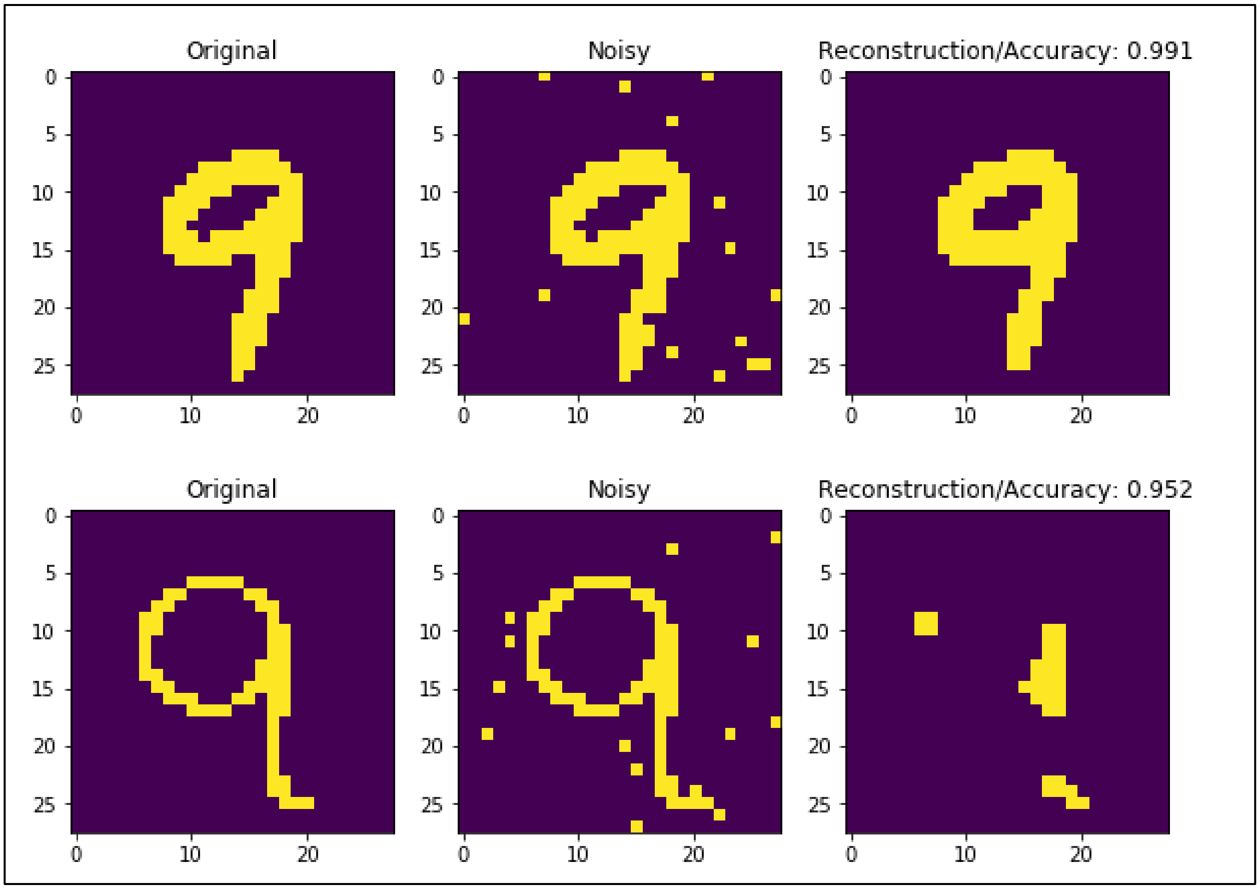
Digit 7 – Best and Worst reconstruction



Digit 8 – Best and Worst reconstruction



Digit 9 – Best and Worst reconstruction



HW7-Submission

May 14, 2020

Import necessary libraries and load the dataset

```
In [1]: import h5py
        import numpy as np
        from random import randint
        import time
        import requests
        import matplotlib.pyplot as plt
        import cv2
        from sklearn.ensemble import RandomForestClassifier
        from scipy.spatial import distance

        #Load MNIST DATA
        MNIST_DATA=h5py.File('MNISTdata.hdf5', 'r')
        x_train= np.float32(MNIST_DATA['x_train'][:])
        y_train= np.int32(np.array(MNIST_DATA['y_train'][:,0]))
        x_test= np.float32(MNIST_DATA['x_test'][:])
        y_test=np.int32(np.array(MNIST_DATA['y_test'][:,0]))
```

Extract and Binarize the first 500 images

```
In [2]: sample= x_train[:500,:]
        sample_map= sample.copy()
        for image in range (0, len(sample_map)):
            x= sample_map[image]
            x[x <= 0.5] = -1
            x[x> 0.5] = 1
            sample_map[image]= x
```

Create a noisy version

```
In [3]: samplesize= len(sample[0])
        noisy_flp= int(0.02*samplesize)
```

```
In [4]: noisy_ver= sample_map.copy()
        for image in range (0, len(noisy_ver)):
            pic = noisy_ver[image]
            permutation= np.random.permutation(samplesize)
            for i in range (0, noisy_flp):
                if pic[permutation[i]] == -1:
                    pic[permutation[i]]= 1
                else:
                    pic[permutation[i]]= -1
            noisy_ver[image]= pic
```

Denoise each image using a Boltzmann machine model and mean field inference

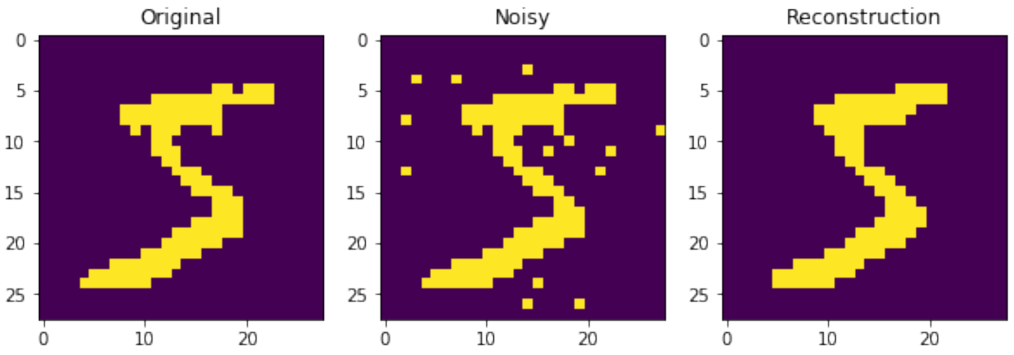
```
In [5]: def neigh_coord (x,y):
            N=(x-1,y)
            S=(x+1,y)
            E=(x, y+1)
            W=(x, y-1)
            n_pos= [N,S,E,W]
            index= []
            for i in range (0,4):
                if n_pos[i][0] <0 or n_pos[i][1] <0 or n_pos[i][0]>27 or n_pos[i][1]>27:
                    index.append(i)
            for i in range (0, len(index)):
                if i==0:
                    n_pos.pop(index[i])
                else:
                    n_pos.pop(index[i]-1)
            return (n_pos)
```

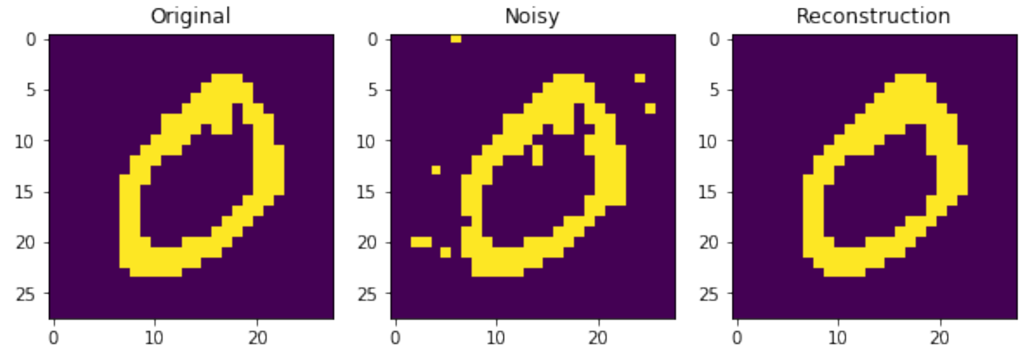
```
In [6]: theta_hj=0.2
        theta_xj=0.5
        reconstruct= []
        for sample in range (0, 500):
            pi_new=np.random.rand(28,28)
            image= noisy_ver[sample]
            image= image.reshape(28,28)
            k=0
            update=10
            while update !=0:
                k= k+1
                pi_old= pi_new.copy()
                for i in range (0,len(image)):
                    for j in range (0, len(image)):
                        n_pos= neigh_coord(i,j)
                        a1=0
                        b1=0
                        a2=0
                        b2=0
                        for pix in range (0, len(n_pos)):
                            pos= n_pos[pix]
```

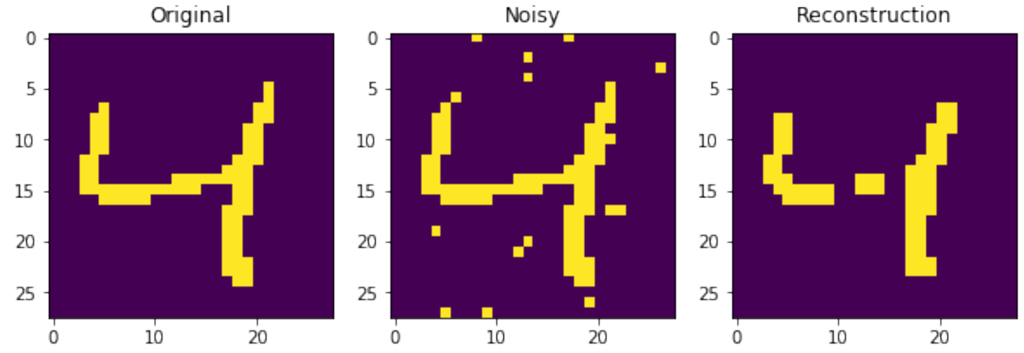
```
                   a1= (theta_hj*(2*(pi_new[pos])-1)) + a1
                   b1= (-1*theta_hj*(2*(pi_new[pos])-1)) + b1
                   a2= theta_xj*image[pos] + a2
                   b2= (-1*theta_xj*image[pos]) + b2
                   a= a1+a2
                   b= b1+b2
                   pi_new[i,j]= (np.exp(a))/(np.exp(a)+np.exp(b))
       update= distance.euclidean(pi_new.flatten(),pi_old.flatten())
       if k==30:
           break

   reconstruct.append(pi_new)
```

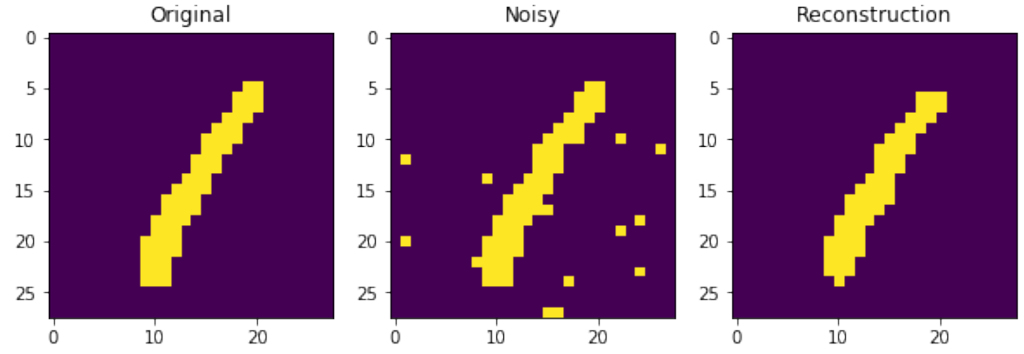
In [38]:
```
plot_go= reconstruct[:500]
plot_set= []
for i in range (0, len(plot_go)):
    img_reconst= reconstruct[i]
    img_reconst[img_reconst<=0.5]= -1
    img_reconst[img_reconst>0.5]= 1
    img_reconst.reshape(28,28)
    plot_set.append(img_reconst)
```

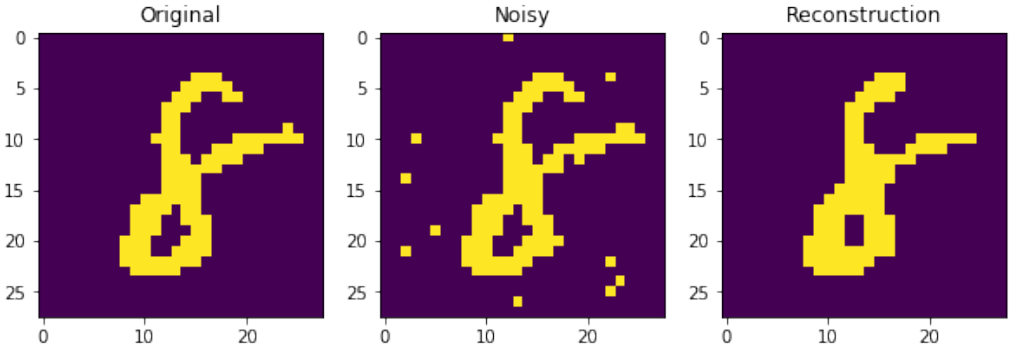
In [39]:
```
for i in range (0, len(plot_set)):
    fig= plt.figure(figsize=(10,10))
    a = fig.add_subplot(1, 3, 3)
    a.set_title('Reconstruction')
    imgplot = plt.imshow(plot_set[i])

    a = fig.add_subplot(1, 3, 2)
    a.set_title('Noisy')
    imgplot = plt.imshow(noisy_ver[i].reshape(28,28))
    a = fig.add_subplot(1, 3, 1)
    a.set_title('Original')
    imgplot = plt.imshow(sample_map[i].reshape((28,28)))

    plt.show()
```




Original
Noisy
Reconstruction
0
5
10
15
20
25
0
10
20


Original
Noisy
Reconstruction
0
5
10
15
20
25
0
10
20


Original
Noisy
Reconstruction
0
5
10
15
20
25
0
10
20



0.0.1 FRACTION OF ACCURATE RECONSTRUCTION

```
In [43]: correct_pixels = 0
         for i in range (0, 500):
             for pixel in range (0,784):
                 if sample_map[i][pixel] == (plot_set[i].flatten())[pixel]:
                     correct_pixels = 1 + correct_pixels

         fraction= correct_pixels/(500*28*28)
         print ('Fraction of correct pixels: ', fraction)
```

```
Fraction of correct pixels:  0.9810714285714286
```

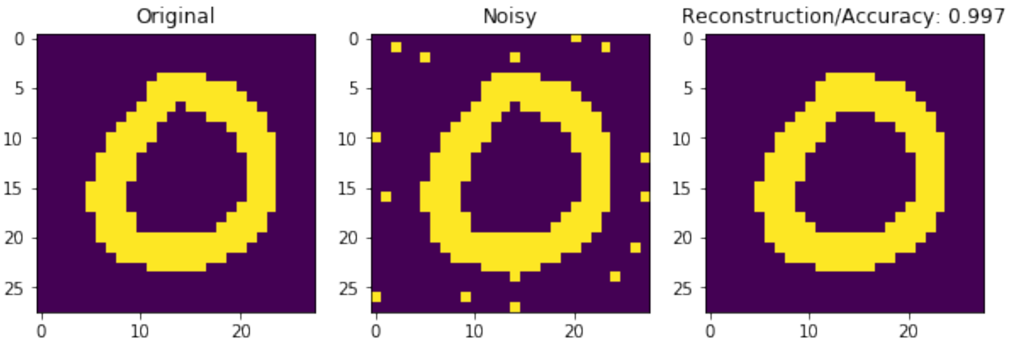
```
In [45]: sampley= y_train[:500]
```

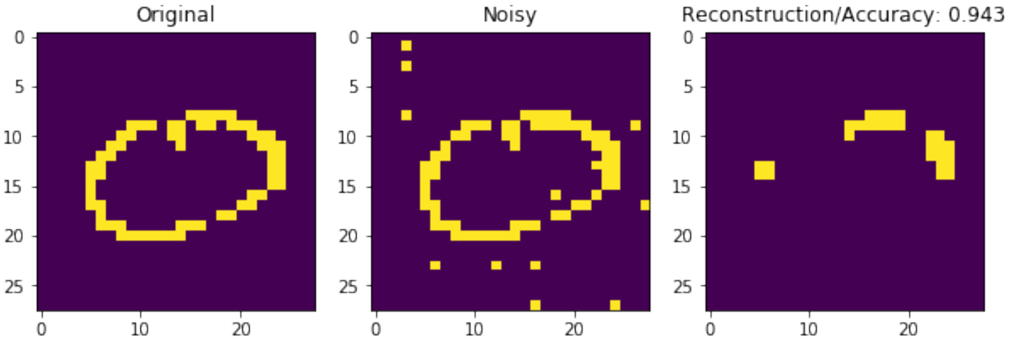
```
In [50]: dictionary= {}
         for i in range (0,10):
             index= []
             for j in range (0,500):
                 if sampley[j] == i:
                     index.append(j)
             dictionary[i]= index
```

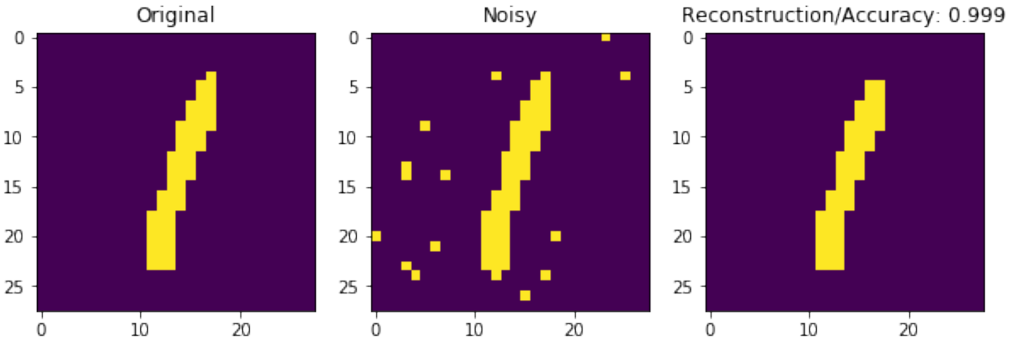
0.0.2 PLOT BEST/WORST RECONSTRUCTION FOR EACH DIGIT

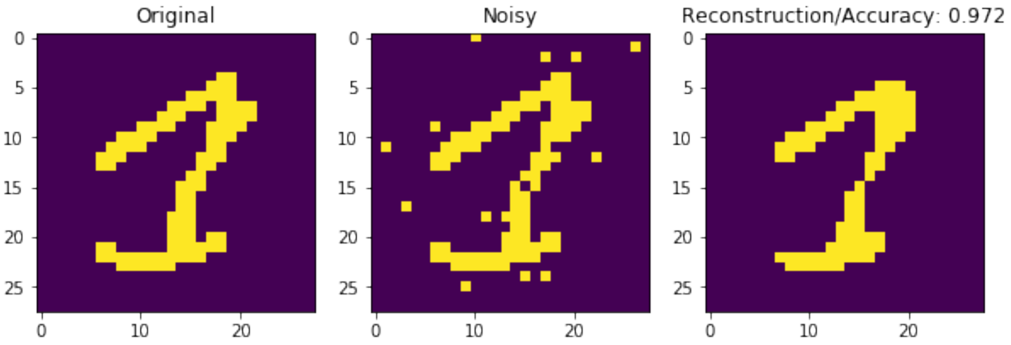
```
In [91]: for digit in range (0, 10):
             number= dictionary[digit]
             summary = []
             for i in range (0,len(number)):
                 index= number[i]
                 correct_pixels= 0
                 for pixel in range (0,784):
```

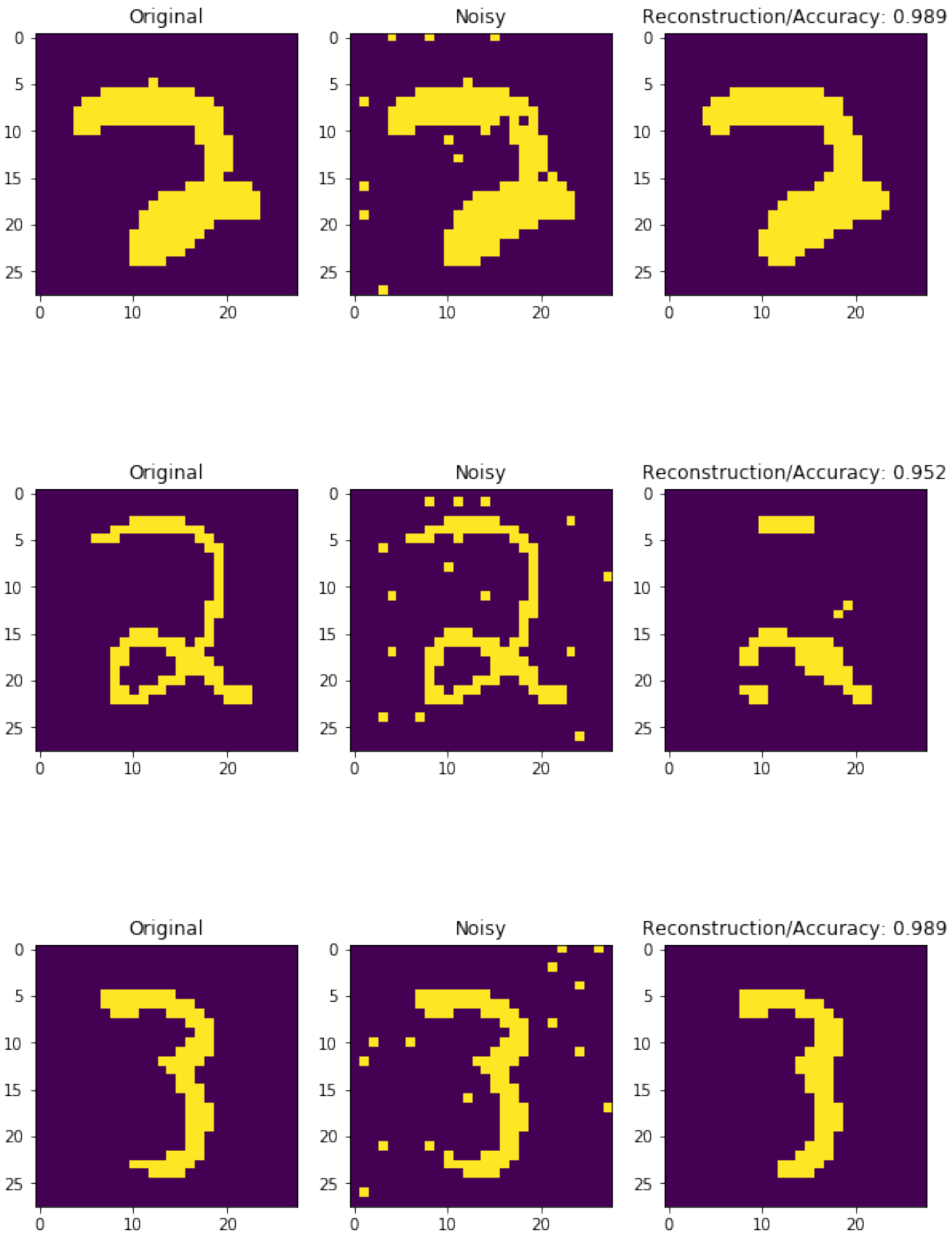
```
        if sample_map[index][pixel] == (plot_set[index].flatten())[pixel]:
            correct_pixels = 1 + correct_pixels
    summary.append(correct_pixels)

best_accuracy= number[np.argmax(summary)]
worst_accuracy= number[np.argmin(summary)]
PLOT= [best_accuracy, worst_accuracy]

value = [(summary[np.argmax(summary)]/784), (summary[np.argmin(summary)]/784)]
for i in range (0, len(PLOT)):
    fig= plt.figure(figsize=(10,10))
#     fig = plt.figure()
    a = fig.add_subplot(1, 3, 3)
    a.set_title('Reconstruction' + '/Accuracy: ' + "{:.3f}".format(value[i]))
    imgplot = plt.imshow(plot_set[PLOT[i]])

    a = fig.add_subplot(1, 3, 2)
    a.set_title('Noisy')
    imgplot = plt.imshow(noisy_ver[PLOT[i]].reshape(28,28))
    a = fig.add_subplot(1, 3, 1)
    a.set_title('Original')
    imgplot = plt.imshow(sample_map[PLOT[i]].reshape((28,28)))

    plt.show()
```

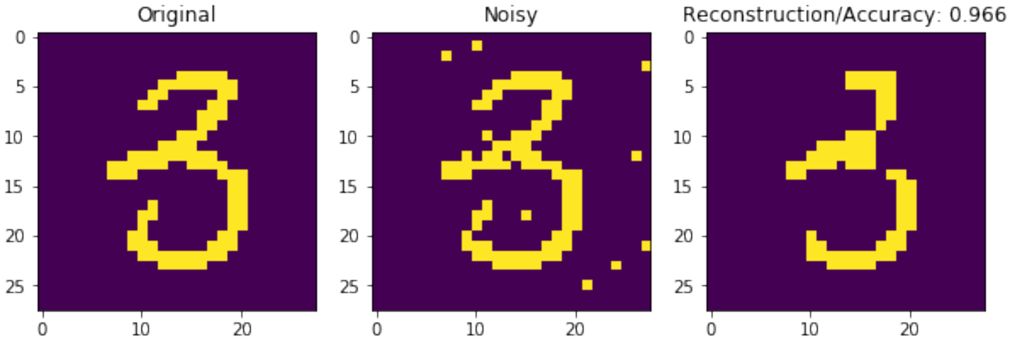


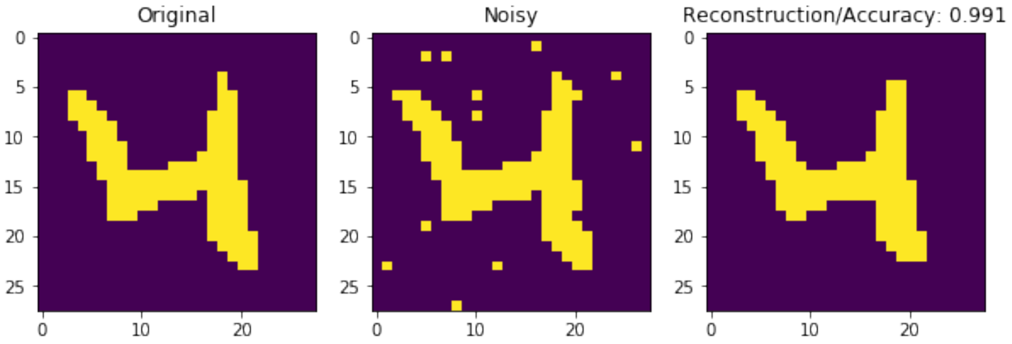

Original
Noisy
Reconstruction/Accuracy: 0.943
0
5
10
15
20
25
0
10
20

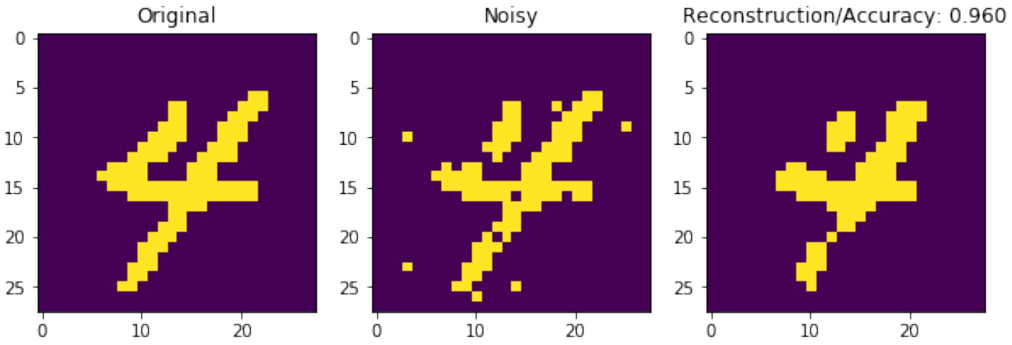

Original
Noisy
Reconstruction/Accuracy: 0.999
0
5
10
15
20
25
0
10
20

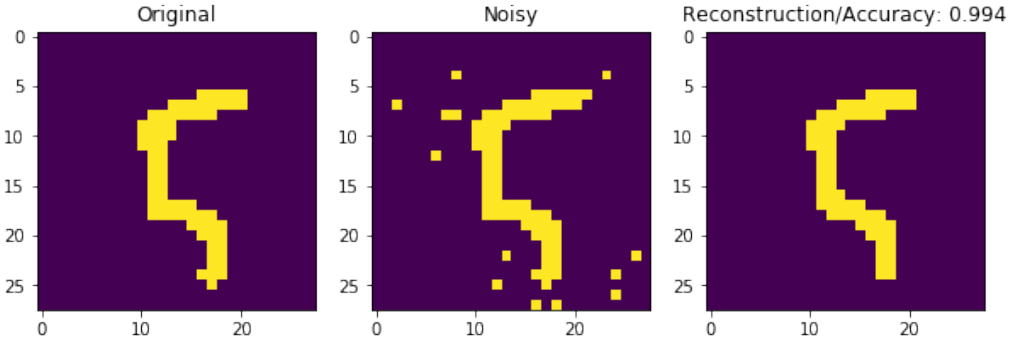

Original
Noisy
Reconstruction/Accuracy: 0.972
0
5
10
15
20
25
0
10
20

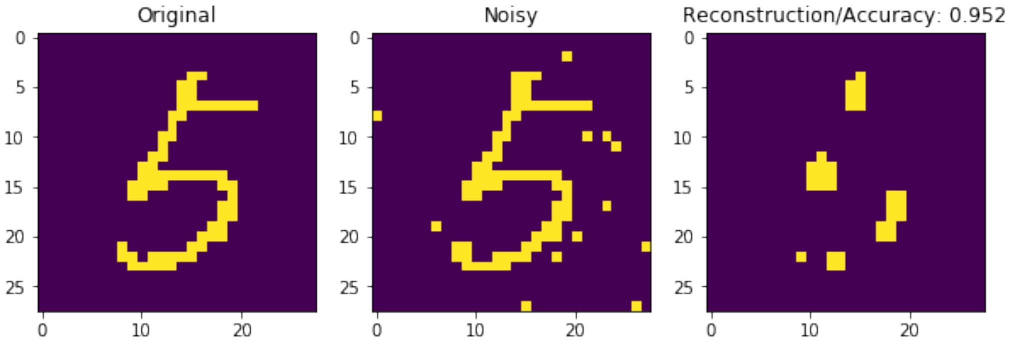

Original
Noisy
Reconstruction/Accuracy: 0.989
0
5
10
15
20
25
0
10
20
Original
Noisy
Reconstruction/Accuracy: 0.952
0
5
10
15
20
25
0
10
20
Original
Noisy
Reconstruction/Accuracy: 0.989
0
5
10
15
20
25
0
10
20

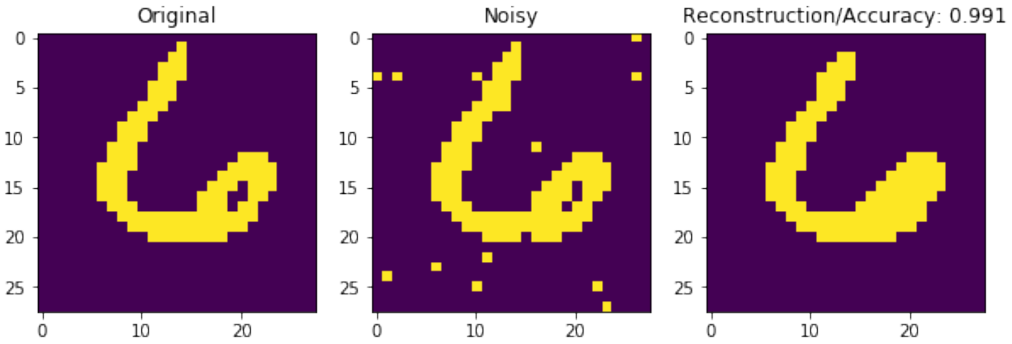

Original
Noisy
Reconstruction/Accuracy: 0.966
0
5
10
15
20
25
0
10
20

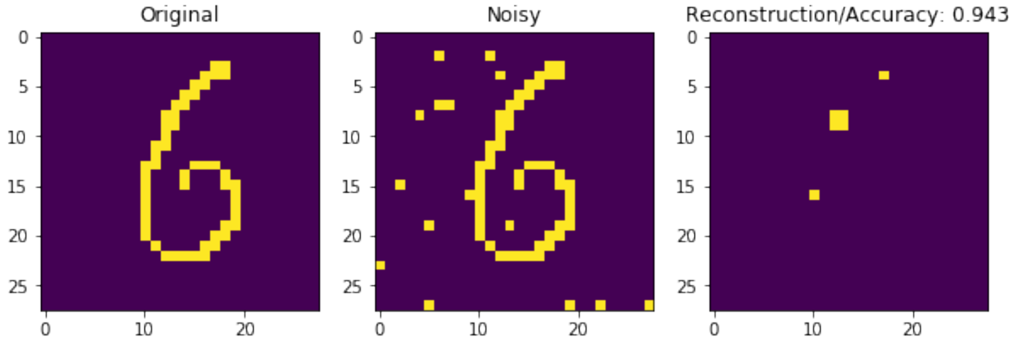

Original
Noisy
Reconstruction/Accuracy: 0.991
0
5
10
15
20
25
0
10
20

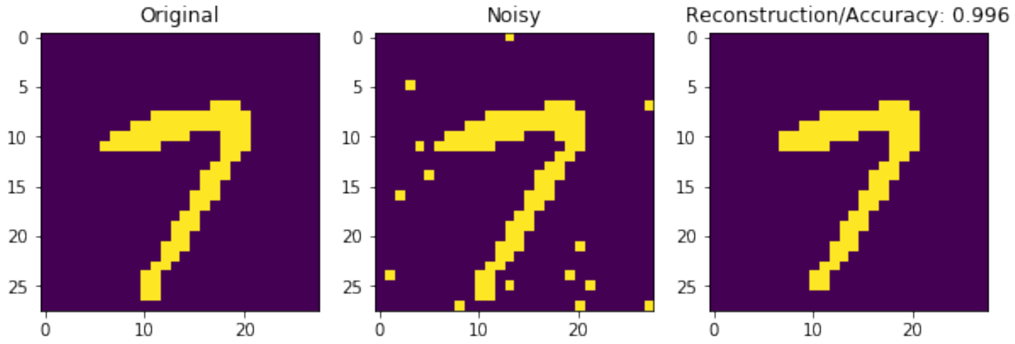

Original
Noisy
Reconstruction/Accuracy: 0.960
0
5
10
15
20
25
0
10
20

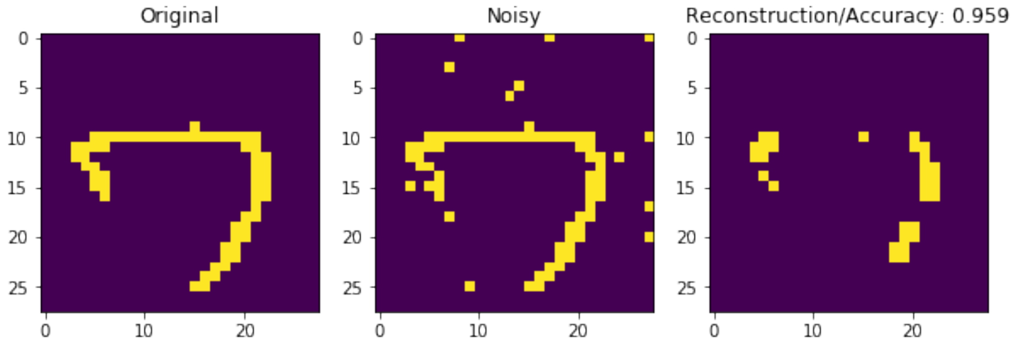

Original
Noisy
Reconstruction/Accuracy: 0.994
0
5
10
15
20
25
0
10
20

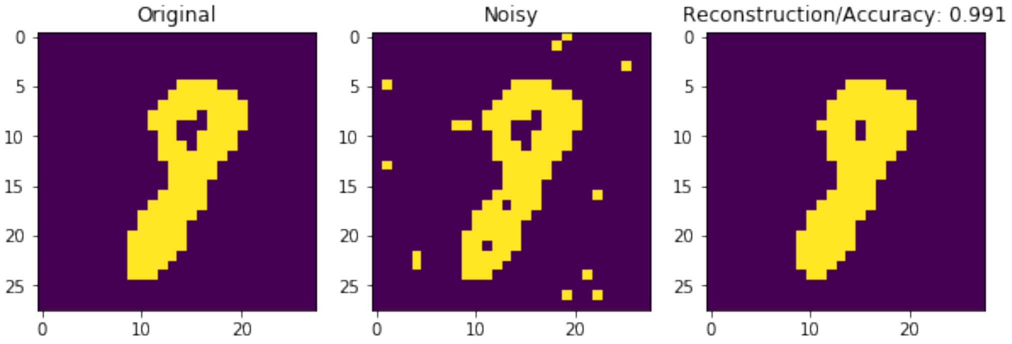

Original
Noisy
Reconstruction/Accuracy: 0.952
0
5
10
15
20
25
0
10
20

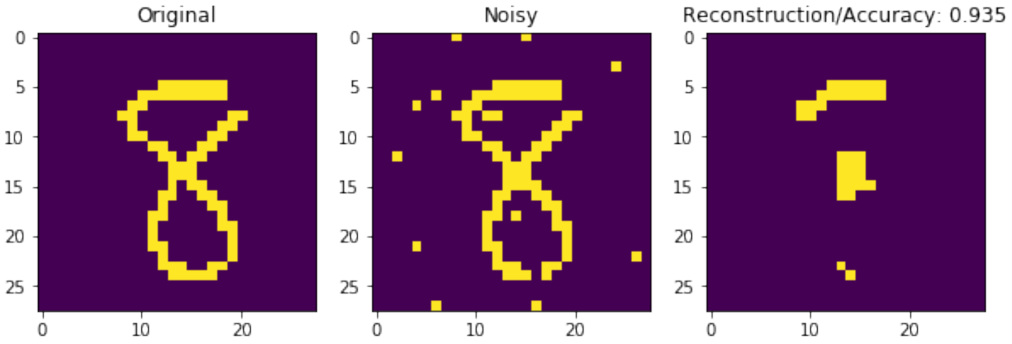

Original
Noisy
Reconstruction/Accuracy: 0.991
0
5
10
15
20
25
0
10
20

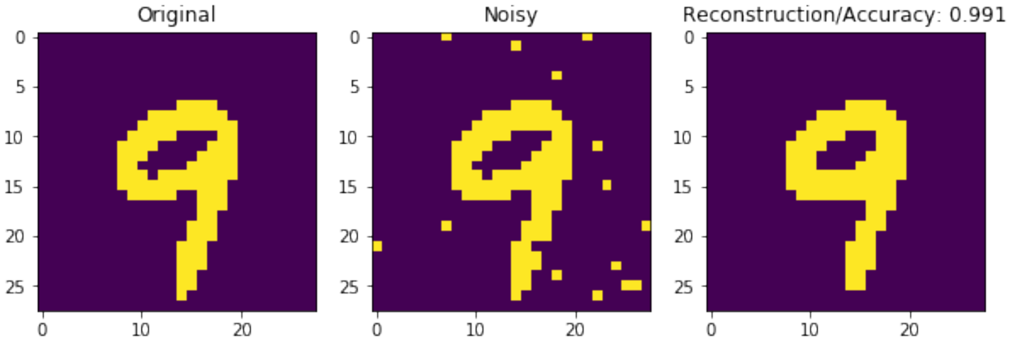

Original
Noisy
Reconstruction/Accuracy: 0.943
0
5
10
15
20
25
0
10
20

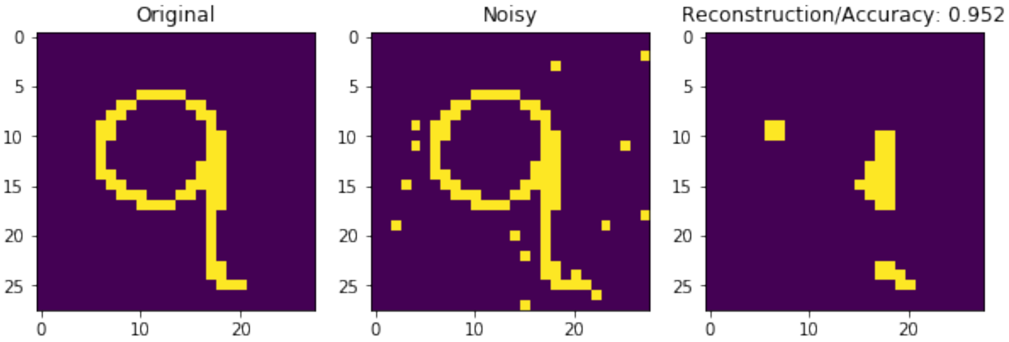

Original
Noisy
Reconstruction/Accuracy: 0.996
0
5
10
15
20
25
0
10
20


Original
Noisy
Reconstruction/Accuracy: 0.959
0
5
10
15
20
25
0
10
20


Original
Noisy
Reconstruction/Accuracy: 0.991
0
5
10
15
20
25
0
10
20


Original
Noisy
Reconstruction/Accuracy: 0.935
0
5
10
15
20
25
0
10
20


Original
Noisy
Reconstruction/Accuracy: 0.991
0
5
10
15
20
25
0
10
20


Original
Noisy
Reconstruction/Accuracy: 0.952
0
5
10
15
20
25
0
10
20