1. Average accuracy on the first 500 images: 99.48571%

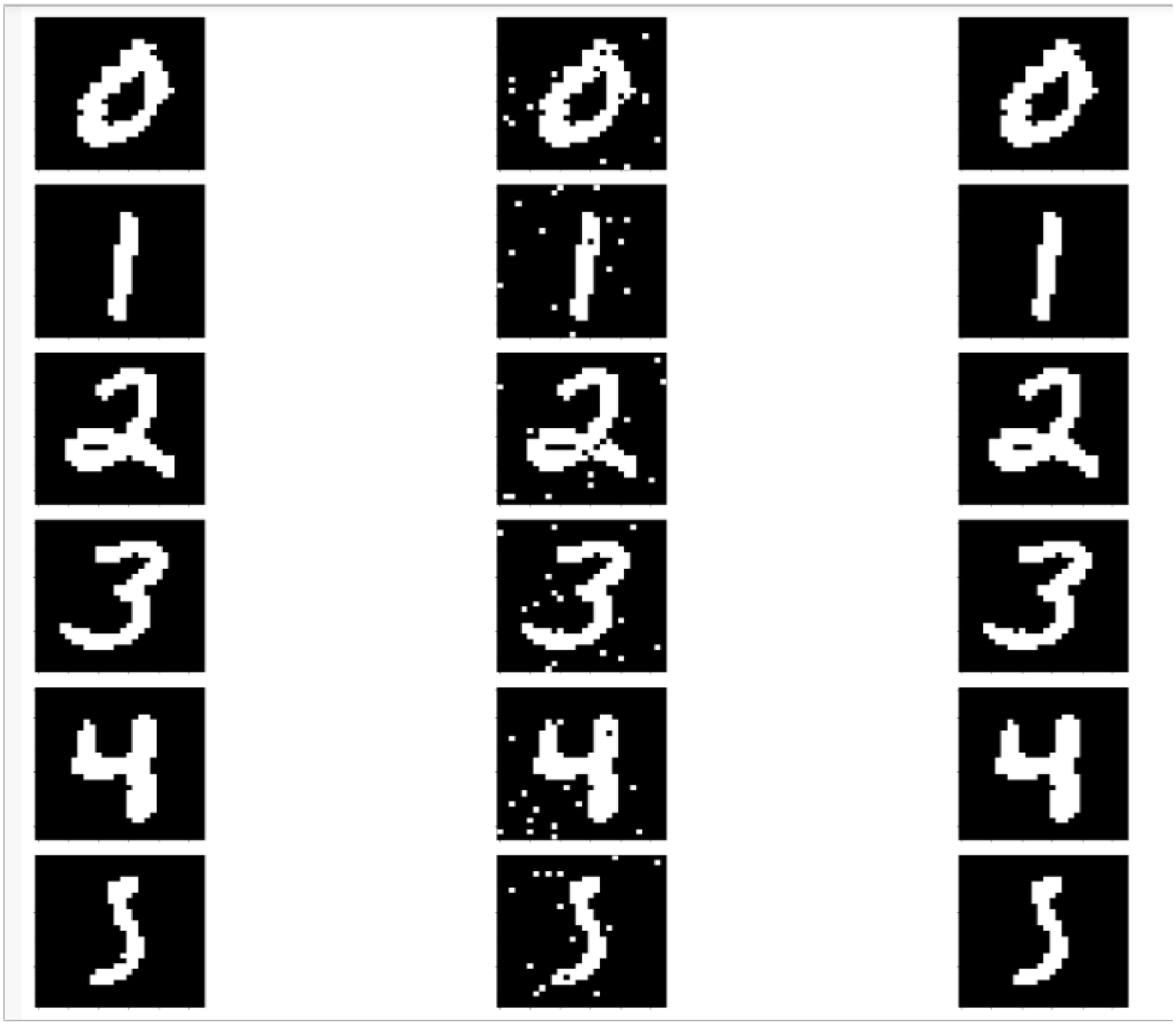
```
#Report the fraction of all pixels that are correct in the 500 images.
print("The accuracy between true images and denoised image :",(np.sum(pic==denoised_images)/(784*500))*100,"%")
```

```
The accuracy between true images and denoised image : 99.4857142857143 %
```

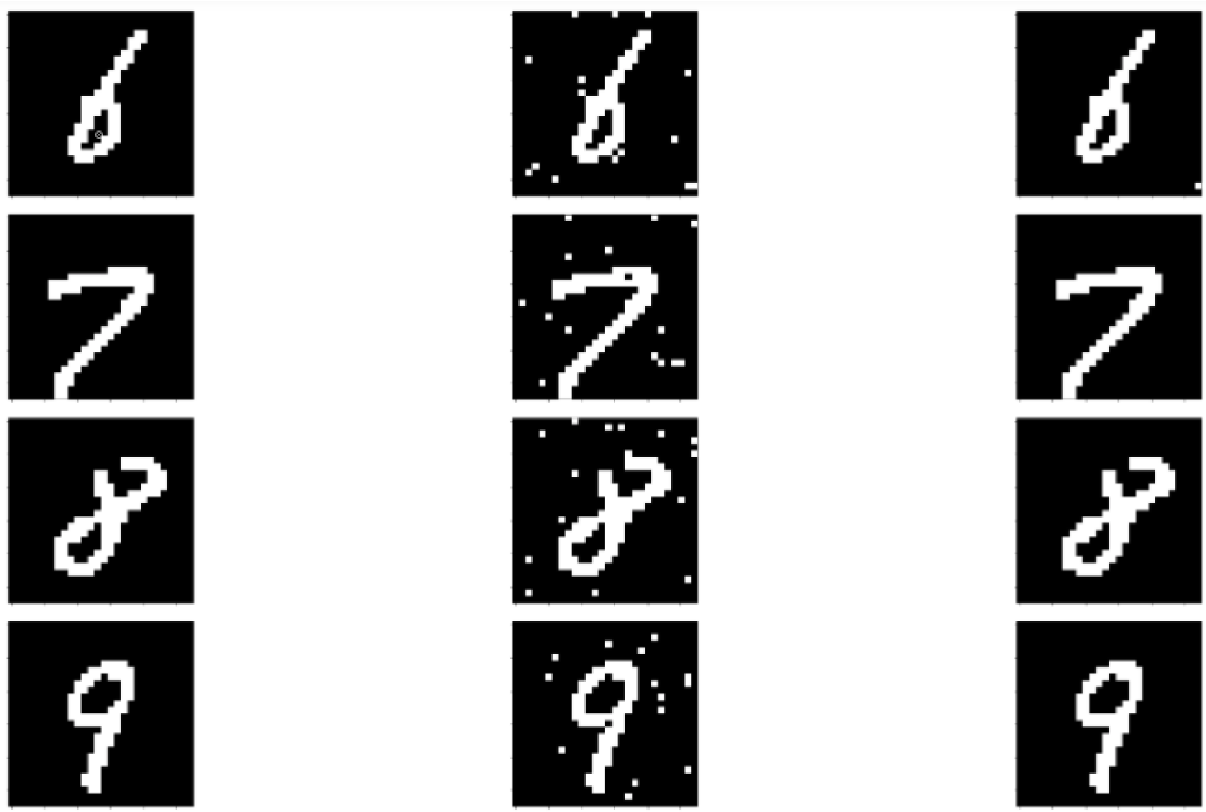
2. One set of sample images for each digit -- For each digit you should put up a row of a (sample image, noised version, denoised version via MFI). This means should have a total of 30 images (10 rows, 3 columns).

The Image is reconstructed with theta_ij(Hi_Hj)=0.2 and theta_ij(Hi_Xj)=0.2
I took the best reconstructed images.

True Image	Noisy Image	Reconstructed Image

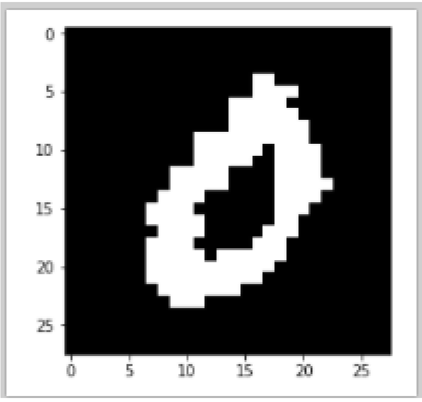


True Image **Noisy Image** **Reconstructed Image**

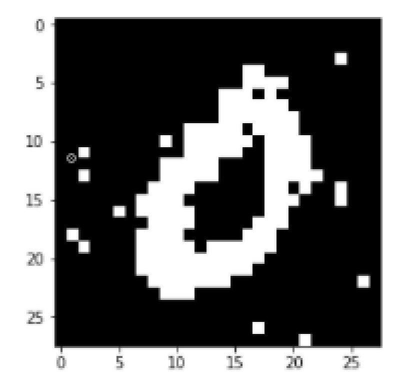


3. Best reconstruction (original, noisy, denoised):
The Image is reconstructed with theta_ij(Hi_Hj)=0.2 and theta_ij(Hi_Xj)=0.2

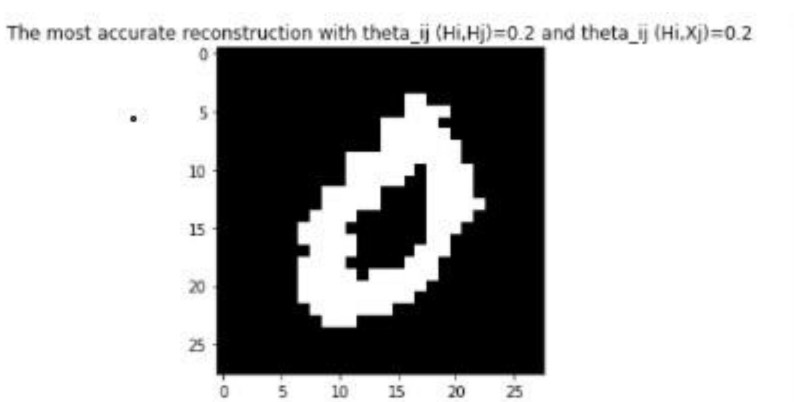
Original iMAGE



Noisy



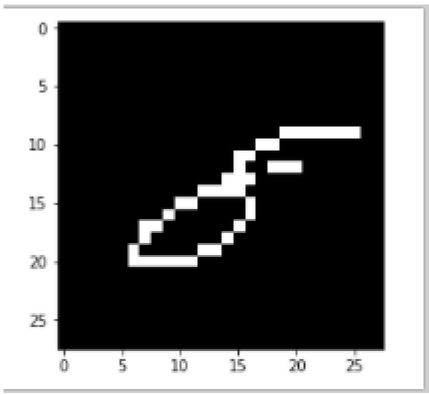
Reconstructed



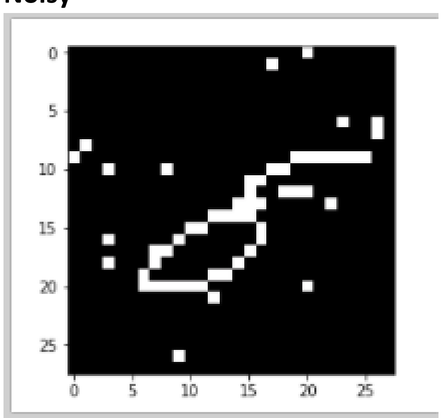
4 . Worst reconstruction (original, noisy, denoised)

The Image is reconstructed with theta_ij(Hi_Hj)=0.2 and theta_ij(Hi_Xj)=0.2

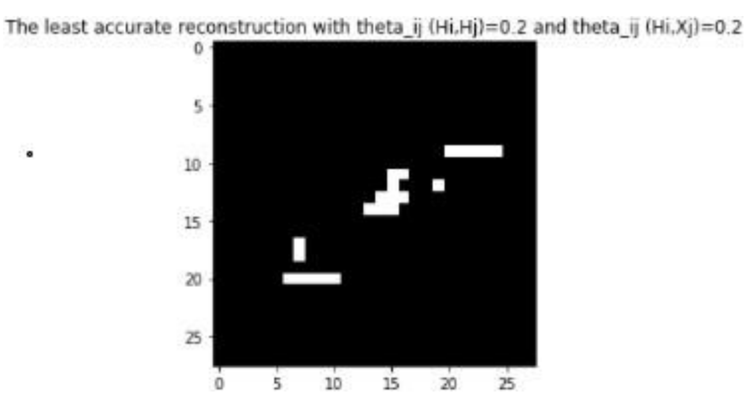
Ortginal



Noisy



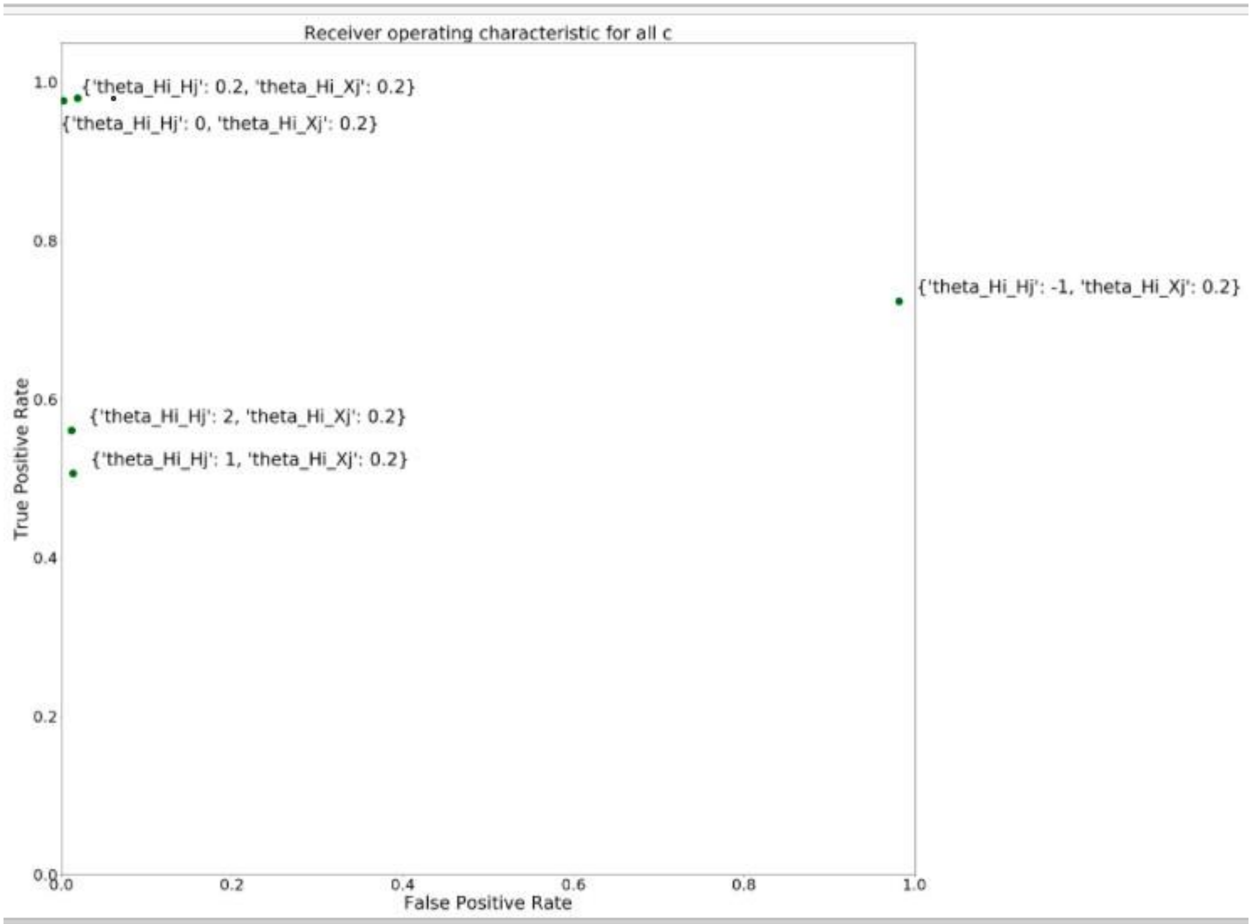
Reconstructed



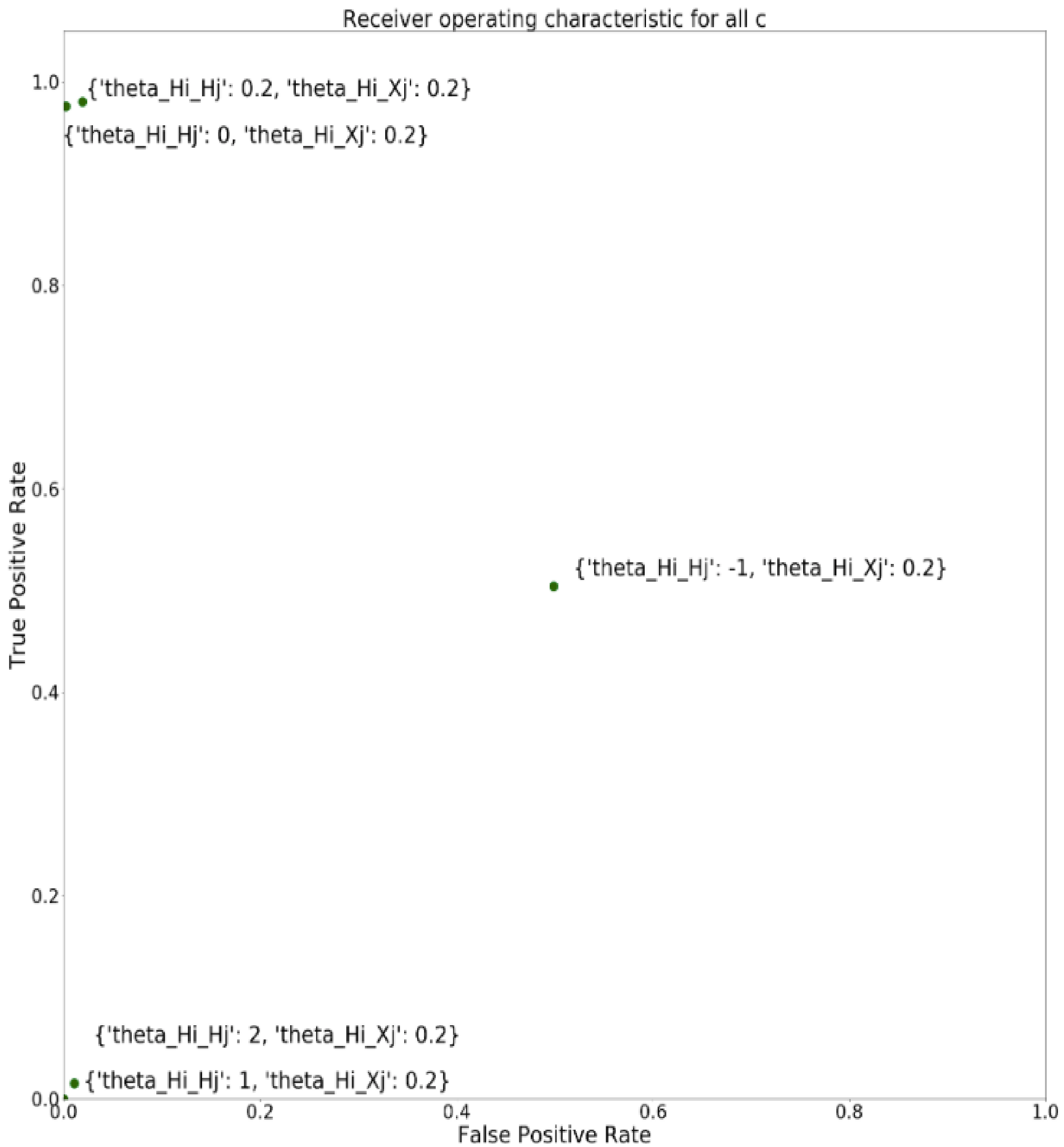
5. ROC curve:

Two Instances;

1. When I don't use updated value of pi (While updating pi for every pixel in a particular iteration)



2. When I use the updated value of pi ((While updating pi for every pixel in a particular iteration))



6. Code snippet:

Importing Libraries

```
In [1]: import os
        import struct
        import numpy as np
        import pandas as pd
        %matplotlib inline
        import matplotlib.pyplot as plt
        #import tensorflow as tf
        import numpy as np
        from sklearn.metrics import confusion_matrix
        import matplotlib.gridspec as gridspec
```

Binarize the Data set

```
def binarize_dataset(X):

  for i in range(0,X.shape[0]):
     for j in range(0,X.shape[1]):
              if(X[i,j]>127):
                   X[i,j]=1
              else:
                   X[i,j]=-1
  return X
```

Adding random Noise to the data set

```
In [11]: #Create Noisy Data (Randomly)

         X_train_noisy=np.copy(Train[:,1:785])#.values)
         y_train_noisy=np.copy(Train[:,0])#.values)

In [12]: for i in range(0,X_train.shape[0]):
             bits_to_flip = np.random.choice(784,15,replace=False)
             for j in range(len(bits_to_flip)):
                X_train_noisy[i,bits_to_flip[j]] = (X_train_noisy[i,bits_to_flip[j]])*-1
```

Boltzamn Denoising Calculation (When I don’t use the updated value of pi)

```
#Denoising the image
def boltzman(pic,theta_H,theta_X):

  pi_output=[[] for k in range(500)]

  min_error=0.1

  for i in range(len(pic)):
      pi=np.ones((28,28))*0.5
      pi_prev=np.ones((28,28))*0.5

      for l in range(30):
              for j in range(pic[i].shape[0]):
                  for k in range(pic[i].shape[0]):
                       z=0
                       if(j>0):

                           z=z+((theta_H)*((2*pi_prev[j-1,k])-1))

                       if(j<27):

                           z=z+((theta_H)*((2*pi_prev[j+1,k])-1))

                       if(k>0):

                           z=z+((theta_H)*((2*pi_prev[j,k-1])-1))

                       if(k<27):

                           z=z+((theta_H)*((2*pi_prev[j,k+1])-1))

                       z=z+(pic[i,j,k]*(theta_X))

                       pi[j,k]=(np.exp(z))/((np.exp(z)+np.exp(-z)))


              pi_prev=np.copy(pi)
      pi_output[i]=pi



  return pi_output
```

Boltzamn Denoising Calculation (When I use the updated value of pi)

```
#Denoising the image
def boltzman(pic,theta_H,theta_X):

  pi_output=[[] for k in range(500)]

  min_error=0.1

  for i in range(len(pic)):
      pi=np.ones((28,28))*0.5
      pi_prev=np.ones((28,28))*0.5

      for l in range(50):
              for j in range(pic[i].shape[0]):
                  for k in range(pic[i].shape[0]):
                        z=0
                        if(j>0):

                            z=z+((theta_H)*((2*pi[j-1,k])-1))

                        if(j<27):

                            z=z+((theta_H)*((2*pi[j+1,k])-1))

                        if(k>0):

                            z=z+((theta_H)*((2*pi[j,k-1])-1))

                        if(k<27):

                            z=z+((theta_H)*((2*pi[j,k+1])-1))

                        z=z+(pic[i,j,k]*(theta_X))

                        pi[j,k]=(np.exp(z))/((np.exp(z)+np.exp(-z)))


              pi_prev=np.copy(pi)
      pi_output[i]=pi



  return pi_output
```

Boltzman Output

```
#output->Boltzman
def result(pi_updated):

  for i in range(500):
     for j in range(28):
        for k in range(28):
              if(pi_updated[i,j,k]>0.5):
                  pi[i,j,k]=1

              else:
                  pi_updated[i,j,k]=-1
  return pi
```

Creating the ROC Curves

```
#ROC Curve

# theta_ij (Hi,Hj)(Xi,Xj)[(-1,0.2),(0,0.2),(0.2,0.2),(1,0.2),(2,0.2)]

R_P=[]
R_F=[]

for theta in [(-1,0.2),(0,0.2),(0.2,0.2),(1,0.2),(2,0.2)]:
  tp=0
  fp=0
  labels=[]
  TPR=np.sum(pic==1)
  FPR=np.sum(pic==-1)
  pi=np.array(boltzman(pic_noisy,theta[0],theta[1]))
  denoised_images=np.array(result(pi))
  for i in range(500):

    for j in range(28):
      for k in range(28):
        if(denoised_images[i,j,k]==1 and pic[i,j,k]==1):
          tp=tp+1
        elif(denoised_images[i,j,k]==1 and pic[i,j,k]==-1):
          fp=fp+1


  R_P.append(tp/TPR)
  print("True Positive",tp)
  R_F.append(fp/FPR)
  print("False Positive",fp)
```

```
plt.figure()
lw = 2
plt.rcParams['figure.figsize'] = [30,30]
types = [{"theta_Hi_Hj":-1,"theta_Hi_Xj":0.2},{"theta_Hi_Hj":0,"theta_Hi_Xj":0.2},{"theta_Hi_Hj":0.2,"theta_Hi_Xj
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='-')
plt.scatter(R_F,R_P, color='darkgreen',s=200)
plt.rc('xtick', labelsize=30)
plt.rc('ytick', labelsize=30)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate',fontsize=35)
plt.ylabel('True Positive Rate',fontsize=35)
plt.title('Receiver operating characteristic for all c',fontsize=35)

for i in range(len(R_P)):
     if i==1:
        plt.text(R_F[i]-0.02, R_P[i]-0.04, types[i], fontsize=35)
        continue
     plt.text(R_F[i]+0.02, R_P[i]+0.01, types[i], fontsize=35)




plt.show()
```