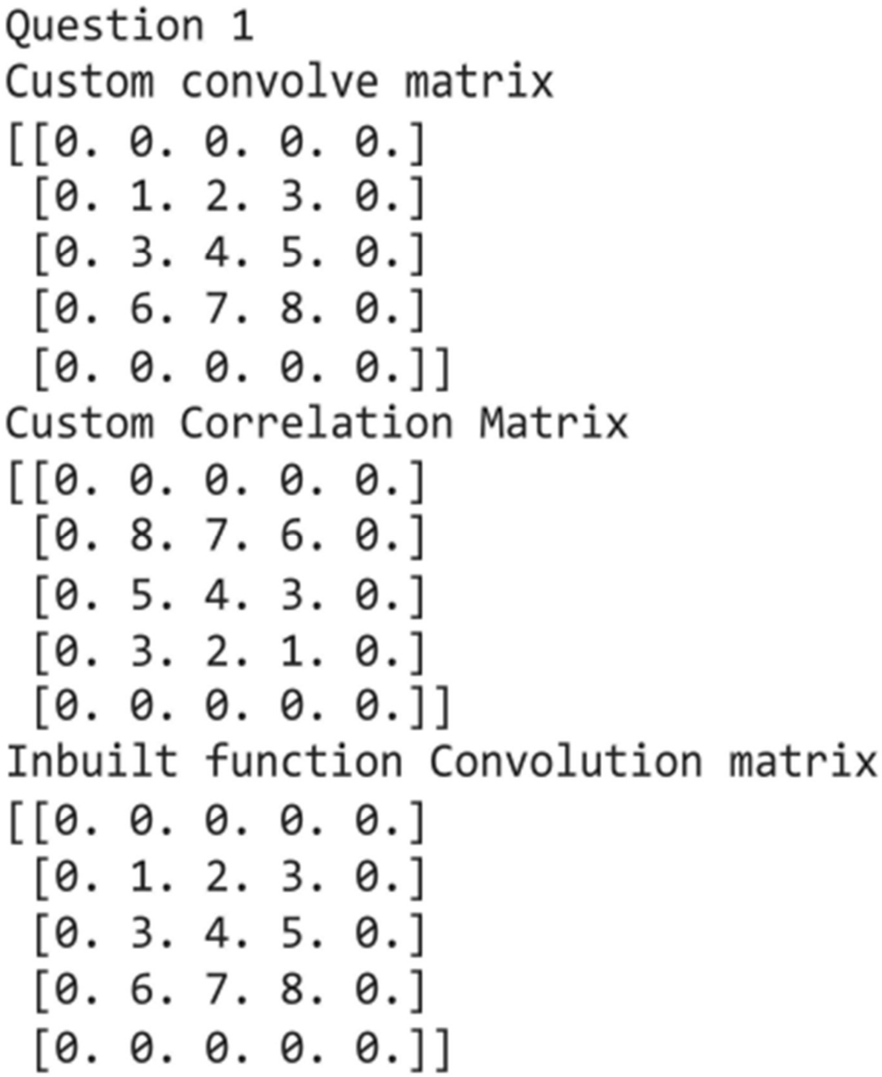
EEE 591 Machine Learning with deployment to FPGA

HOMEWORK 6
ASU ID: 1225713099
Praveen Paidi

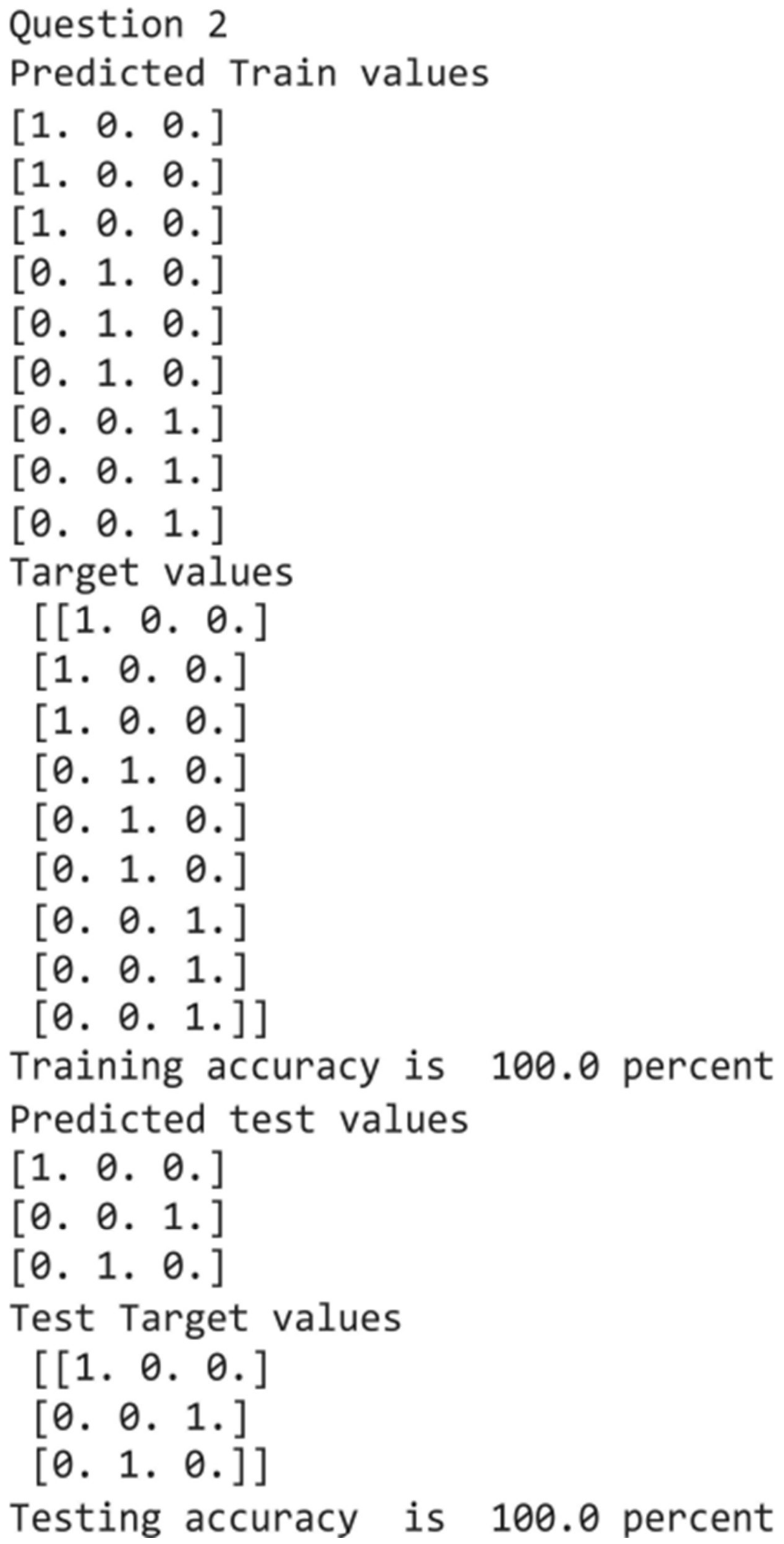
p

1.



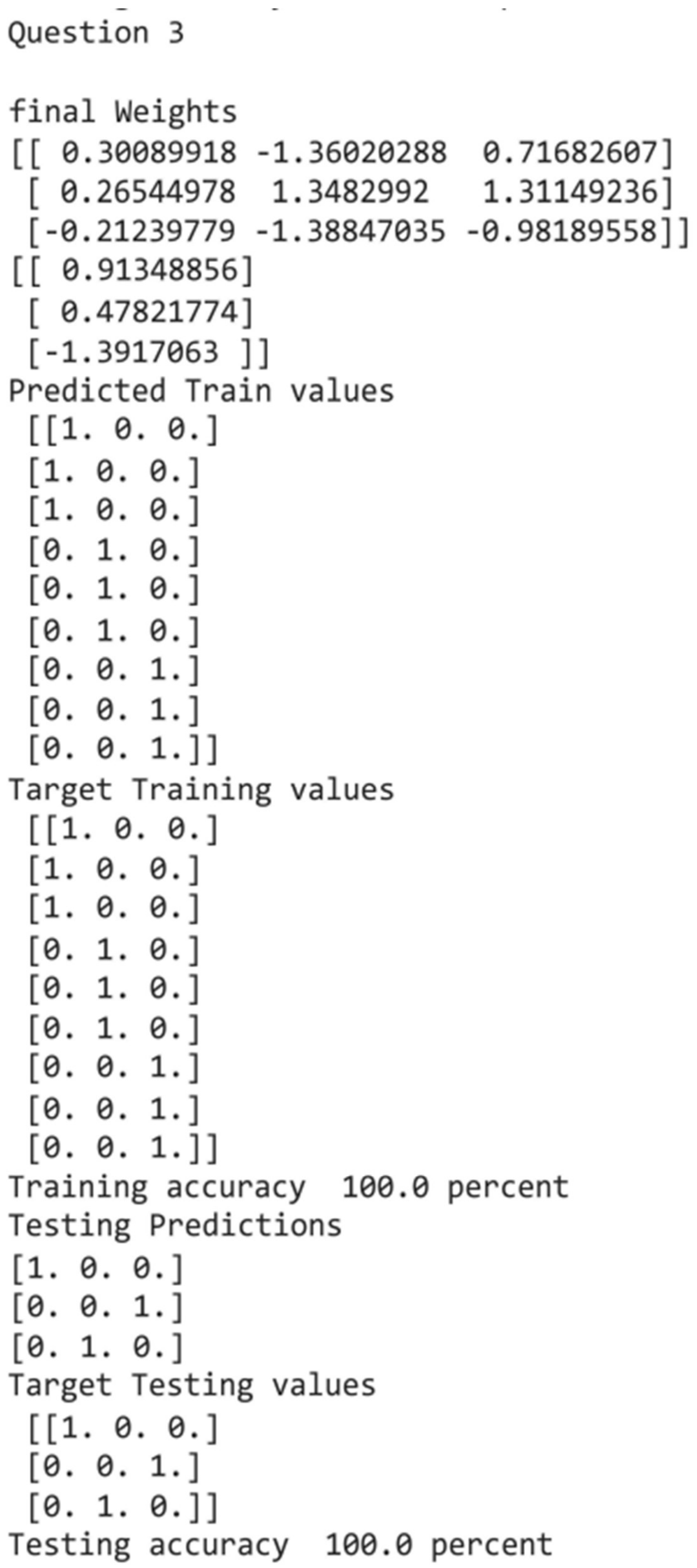
```
Question 1
Custom convolve matrix
[[0. 0. 0. 0. 0.]
 [0. 1. 2. 3. 0.]
 [0. 3. 4. 5. 0.]
 [0. 6. 7. 8. 0.]
 [0. 0. 0. 0. 0.]]
Custom Correlation Matrix
[[0. 0. 0. 0. 0.]
 [0. 8. 7. 6. 0.]
 [0. 5. 4. 3. 0.]
 [0. 3. 2. 1. 0.]
 [0. 0. 0. 0. 0.]]
Inbuilt function Convolution matrix
[[0. 0. 0. 0. 0.]
 [0. 1. 2. 3. 0.]
 [0. 3. 4. 5. 0.]
 [0. 6. 7. 8. 0.]
 [0. 0. 0. 0. 0.]]
```

2.



```
Question 2
Predicted Train values
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[0. 1. 0.]
[0. 1. 0.]
[0. 1. 0.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
Target values
 [[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
Training accuracy is  100.0 percent
Predicted test values
[1. 0. 0.]
[0. 0. 1.]
[0. 1. 0.]
Test Target values
 [[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]
Testing accuracy  is  100.0 percent
```

3.



```
Question 3

final Weights
[[ 0.30089918 -1.36020288  0.71682607]
 [ 0.26544978  1.3482992   1.31149236]
 [-0.21239779 -1.38847035 -0.98189558]]
[[ 0.91348856]
 [ 0.47821774]
 [-1.3917063 ]]
Predicted Train values
 [[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
Target Training values
 [[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
Training accuracy  100.0 percent
Testing Predictions
[1. 0. 0.]
[0. 0. 1.]
[0. 1. 0.]
Target Testing values
 [[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]
Testing accuracy  100.0 percent
```

Although the weights are not exactly similar they are in scaled version, more training and test samples would have resulted prefect solutions.

In [2]:

```
import scipy.signal as ss
import numpy as np

# Inbuilt function
def conv2Dd_(image,W,stride,Conv):
    if (Conv):
        y = ss.convolve2d(image, W, mode='valid') ## valid padding
    else:
        y = ss.correlate2d(image, W, mode='valid')## valid padding
    Xdim = len(image[0])//stride
    x = np.zeros([Xdim,Xdim],float)
    if stride>1:                                  ## implement stride
        for i in range(0,Xdim):
            for j in range(0,Xdim):
                x[i,j] = y[i*stride,j*stride]
    else:
        x = y
    return(x)

# Custom function
def customconv2Dd_(f,g,stride,Conv):
    if Conv== True:
        g=np.rot90(g, 2)
    else:
        g=g

    m,n = np.shape(f)
    m1,n1= np.shape(g)

    # width - kernel width + 2*padding /stride +1
    m2,n2 = ((m-m1)//stride)+1 , ((n-n1)//stride)+1
    convmat=np.empty((m2,n2))

    a=0
    b=0
    for i in range(0,m2):
        for j in range(0,n2):
            convmat[i,j] = np.sum(np.sum(g*f[a:a+m1,b:b+n1], axis=1), axis=0)
            b=b+stride
        a=a+stride
        b=0
    return convmat


f=np.zeros((7,7));
f[3,3]=1
g=np.array([[1,2,3],[3,4,5],[6,7,8]])

print('Question 1')
print('Custom convolve matrix')
mat1= customconv2Dd_(f, g, 1, True)
print(mat1)

print('Custom Correlation Matrix')
mat2 = customconv2Dd_(f, g, 1, False)
print(mat2)

print('Inbuilt function Convolution matrix')
mat=conv2Dd_(f, g, 1, True)
print(mat)

#Question 2

W1 = np. array([[ 1.6975548, -0.07326141, -0.41880725],[ 0.12228276, -0.19572004, 0.81986898],
                [ 0.8876136, -1.8629187, -0.97661273]])

## And the ANN weights
W2 = np. array([ 1.10485759, 0.2120758, -1.31693339])


Image_dim = 7
stride = 2
Ydim = 3
Wdim = 3
target = np.array([[1,0,0],[1,0,0],[1,0,0],[0,1,0],[0,1,0],[0,1,0],[0,0,1],[0,0,1],[0,0,1]],float)

## training data

image7by7 = np.zeros([9,Image_dim,Image_dim],float)
image7by7[0,1,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,2,:] = np.array([0,0,1,1,0,0,0])
image7by7[0,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,4,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,5,:] = np.array([0,0,1,1,1,0,0])
## 1
image7by7[1,1,:] = np.array([0,0,1,1,0,0,0])
image7by7[1,2,:] = np.array([0,0,0,1,0,0,0])
image7by7[1,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[1,4,:] = np.array([0,0,0,1,0,0,0])
image7by7[1,5,:] = np.array([0,0,0,1,0,0,0])
## 2
image7by7[2,1,:] = np.array([0,0,0,1,0,0,0])
image7by7[2,2,:] = np.array([0,0,0,1,0,0,0])
image7by7[2,3,:] = np.array([0,0,0,1,0,0,0])
```

```
image7by7[2,4,:] = np.array([0,0,0,1,0,0,0])
image7by7[2,5,:] = np.array([0,1,1,1,1,1,0])
## 3
image7by7[3,1,:] = np.array([0,1,1,1,1,1,0])
image7by7[3,2,:] = np.array([0,1,0,0,1,0,0])
image7by7[3,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[3,4,:] = np.array([0,1,1,0,0,0,0])
image7by7[3,5,:] = np.array([0,1,1,1,1,1,0])
## 4
image7by7[4,1,:] = np.array([0,0,1,1,1,0,0])
image7by7[4,2,:] = np.array([0,0,0,0,1,0,0])
image7by7[4,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[4,4,:] = np.array([0,0,1,0,0,0,0])
image7by7[4,5,:] = np.array([0,0,1,1,1,0,0])
## 5
image7by7[5,1,:] = np.array([0,0,1,1,0,0,0])
image7by7[5,2,:] = np.array([0,1,0,0,1,0,0])
image7by7[5,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[5,4,:] = np.array([0,0,1,0,0,0,0])
image7by7[5,5,:] = np.array([0,1,1,1,1,1,0])
## 6
image7by7[6,1,:] = np.array([0,1,1,1,1,1,0])
image7by7[6,2,:] = np.array([0,1,0,0,0,1,0])
image7by7[6,3,:] = np.array([0,0,1,1,1,1,0])
image7by7[6,4,:] = np.array([0,0,0,0,0,1,0])
image7by7[6,5,:] = np.array([0,1,1,1,1,1,0])
## 7
image7by7[7,1,:] = np.array([0,0,1,1,1,0,0])
image7by7[7,2,:] = np.array([0,1,0,0,0,1,0])
image7by7[7,3,:] = np.array([0,0,0,1,1,0,0])
image7by7[7,4,:] = np.array([0,1,0,0,0,1,0])
image7by7[7,5,:] = np.array([0,0,1,1,1,0,0])
## 8
image7by7[8,1,:] = np.array([0,1,1,1,1,1,0])
image7by7[8,2,:] = np.array([0,1,0,0,0,1,0])
image7by7[8,3,:] = np.array([0,0,0,1,1,1,0])
image7by7[8,4,:] = np.array([0,1,0,0,0,1,0])
image7by7[8,5,:] = np.array([0,1,1,1,1,1,0])


#Standardize the training data
image7by7 = (image7by7 - np.mean(image7by7))
image7by7 = image7by7/(np.std(image7by7))

def categorize(X):
    XX = np.zeros(len(X),float)
    maxind = X.tolist().index(max(X.tolist()))
    XX[maxind] = 1.0
    return(XX)

Y_training_pred = []

for i in image7by7:

    # CNN step
    X_train = customconv2Dd_(i,W1,2,True)

    # RNN step
    Y_train = np.dot(X_train,W2)

    # Quantizer either 0 or 1
    Y = categorize(Y_train)

    # Training prediction
    Y_training_pred.append(Y)

print('\nQuestion 2')
print('Predicted Train values')
for i in Y_training_pred:
    print(i)

Training_error=np.where(target!=Y_training_pred)
Accuracy=((len(target) - len(Training_error[0]))/len(target)) * 100
print('Target values\n',target)

print('Training accuracy is ', Accuracy,'percent')


## test data
targett = np.array([[1,0,0],[0,0,1],[0,1,0]],float)
image7by7t = np.zeros([3,7,7],float)
image7by7t[0,1,:] = np.array([0,0,0,1,0,0,0])
image7by7t[0,2,:] = np.array([0,0,0,1,0,0,0])
image7by7t[0,3,:] = np.array([0,0,0,1,0,0,0])
image7by7t[0,4,:] = np.array([0,0,0,1,0,0,0])
image7by7t[0,5,:] = np.array([0,0,0,1,1,0,0])
## print('image 1 \n', image7by7t0)
image7by7t[1,1,:] = np.array([0,0,1,1,1,0,0])
image7by7t[1,2,:] = np.array([0,1,0,0,0,1,0])
image7by7t[1,3,:] = np.array([0,0,0,1,1,1,0])
image7by7t[1,4,:] = np.array([0,1,0,0,0,1,0])
image7by7t[1,5,:] = np.array([0,0,1,1,1,0,0])
## print('image 1 \n', image7by7t1)
## test image a 2
image7by7t[2,1,:] = np.array([0,0,1,1,1,1,0])
image7by7t[2,2,:] = np.array([0,1,0,0,1,0,0])
```

```
image7by7t[2,3,:] = np.array([0,0,0,1,0,0,0])
image7by7t[2,4,:] = np.array([0,0,1,0,0,0,0])
image7by7t[2,5,:] = np.array([0,1,1,1,1,1,0])


#standardization of testing data
image7by7t = (image7by7t - np.mean(image7by7t))
image7by7t = image7by7t/(np.std(image7by7t))

Y_testing_pred = []

for i in image7by7t:

    # CNN step
    X_test = customconv2Dd_(i,W1,2,True)

    # RNN step
    Y_test = np.dot(X_test,W2)

    # Quantizer either 0 or 1
    X3 = categorize(Y_test)

    # Training prediction
    Y_testing_pred.append(X3)


testing_error=np.where(targett!=Y_testing_pred)

print('Predicted test values')
for i in Y_testing_pred:
    print(i)
print('Test Target values \n',targett)
Accuracy= ((len(targett) - len(testing_error[0]))/len(targett)) * 100
print('Testing accuracy  is ',Accuracy ,'percent')


print('Question 3')
np.random.seed(1)
W1 = (np.random.rand(Wdim,Wdim))
W2 = (np.random.rand(Wdim,1))

# Standardize the training data
W1 = (W1 - np.mean(W1))
W1 = W1/(np.std(W1))

#Standardize the training data
W2 = (W2 - np.mean(W2))
W2 = W2/(np.std(W2))

epsilon = 0.1
alpha = 0.5
Z=np.zeros([Wdim,1])


for j in range(50):
    for k,i in enumerate(image7by7):

        # CNN step
        X_train = customconv2Dd_(i,W1,2,True)

        # RNN step
        Y_train = np.dot(X_train,W2)

        # Quantizer either 0 or 1
        YY = categorize(Y_train).reshape(3,1)

        #Error
        t = (target[k].T).reshape(3,1)
        Z = (YY-t)

        # Back propagation of ANN
        dzdW2 = np.dot(X_train,Z)

        # Back propagation of CNN
        dzdW1 = customconv2Dd_(i,np.dot(Z,W2.T),2,True)

        # Updation of weights
        W1 = W1 - epsilon * dzdW1
        W2 = W2 - alpha * dzdW2


W1 = (W1 - np.mean(W1)) / (np.std(W1))
W2 = (W2 - np.mean(W2)) / (np.std(W2))


Y_training_pred = []
for i in image7by7:

    # CNN step
    X_train = customconv2Dd_(i,W1,2,True)

    # RNN step
    Y_train = np.dot(X_train,W2)

    # Quantizer either 0 or 1
    X3 = categorize(Y_train)
```

```python
    # Training prediction
    Y_training_pred.append(X3)

Y_training_pred = np.array(Y_training_pred).reshape(9,3)

print('\nfinal Weights')
print(W1)
print(W2)

print('Predicted Train values \n',Y_training_pred)
print('Target Training values \n',target)

Training_error = np.where(target!=Y_training_pred)
Accuracy=((len(target) - len(Training_error[0]))/len(target)) * 100

print('Training accuracy ',Accuracy ,'percent')



Y_testing_pred=[]
for i in image7by7t:

    # CNN step
    X_test = customconv2Dd_(i,W1,2,True)

    # RNN step
    Y_test = np.dot(X_test,W2)

    # Quantizer either 0 or 1
    X3 = categorize(Y_test)

    # Training prediction
    Y_testing_pred.append(X3)


testing_error=np.where(targett!=Y_testing_pred)
Acuuracy=((len(targett) - len(testing_error[0]))/len(targett)) * 100
print('Testing Predictions')

for i in Y_testing_pred:
    print(i)

print('Target Testing values \n',targett)
print('Testing accuracy ', Accuracy,'percent')
```

```
Question 1
Custom convolve matrix
[[0. 0. 0. 0. 0.]
 [0. 1. 2. 3. 0.]
 [0. 3. 4. 5. 0.]
 [0. 6. 7. 8. 0.]
 [0. 0. 0. 0. 0.]]
Custom Correlation Matrix
[[0. 0. 0. 0. 0.]
 [0. 8. 7. 6. 0.]
 [0. 5. 4. 3. 0.]
 [0. 3. 2. 1. 0.]
 [0. 0. 0. 0. 0.]]
Inbuilt function Convolution matrix
[[0. 0. 0. 0. 0.]
 [0. 1. 2. 3. 0.]
 [0. 3. 4. 5. 0.]
 [0. 6. 7. 8. 0.]
 [0. 0. 0. 0. 0.]]

Question 2
Predicted Train values
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[0. 1. 0.]
[0. 1. 0.]
[0. 1. 0.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
Target values
 [[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
Training accuracy is  100.0 percent
Predicted test values
[1. 0. 0.]
[0. 0. 1.]
[0. 1. 0.]
Test Target values
 [[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]
Testing accuracy  is  100.0 percent
Question 3

final Weights
[[ 0.30089918 -1.36020288  0.71682607]
 [ 0.26544978  1.3482992   1.31149236]
 [-0.21239779 -1.38847035 -0.98189558]]
[[ 0.91348856]
 [ 0.47821774]
 [-1.3917063 ]]
Predicted Train values
 [[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
Target Training values
 [[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
Training accuracy  100.0 percent
Testing Predictions
[1. 0. 0.]
[0. 0. 1.]
[0. 1. 0.]
Target Testing values
 [[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]
Testing accuracy  100.0 percent
```