ENGN 4528: ASSIGNMENT-1 REPORT

U6741351
Tanmay Negi

Task1:

image: [0,1,2,4,10,3,4]
Pad image with zeros on both side: [0,0,1,2,4,10,3,4,0]
f : [1,2,1]
f*I : [(1*0+2*0+1*1), (1*0+2*1+1*2), (1*1+2*2+1*4), (1*2+2* 4+1*10), (1*4+2*10+ 1*3), (1*10+2*3+1*4), (1*3+2*4+1*0)] = [1,4,9,20,27,20,11]

Task2:

```
A = np.array([ [1,1,1,0,1,1,1,1,1,0],
               [1,1,1,0,1,1,1,1,1,0],
               np.ones(10),
               np.ones(10),
               np.ones(10) ])

B = np.ones(shape=(3,3))
```

Eroding :

I have created a function 'erosion' in code file, here's the pseudo code:

1. Pad A with ones
2. Move kernel B across A (same as in convolution) but perform element-wise logical and of B and overlapped region of A
3. Perform logical reduction of resultant matrix and save at the center of overlapped A
4. Shift by stride=1 (similar as convolution)

```
erosion( A , B) :  [ [ 1, 1, 0, 0, 0, 1, 1, 1, 0, 0],
                     [ 1, 1, 0, 0, 0, 1, 1, 1, 0, 0],
                     [ 1, 1, 0, 0, 0, 1, 1, 1, 0, 0],
                     [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                     [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] ]

A – erosion(A,B): [ [0, 0, 1, 0, 1, 0, 0, 0, 1, 0],
                    [0, 0, 1, 0, 1, 0, 0, 0, 1, 0],
                    [0, 0, 1, 1, 1, 0, 0, 1, 1, 1],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] ]
```

Task 3: Contour Detection

Default code result:
threshold : 0.22
overall max F1 score : 0.514369
average max F1 score: 0.562687
area_pr: 0.408983

```
          threshold: 0.220000
overall max F1 score: 0.514369
average max F1 score: 0.562687
            area_pr: 0.408983
```

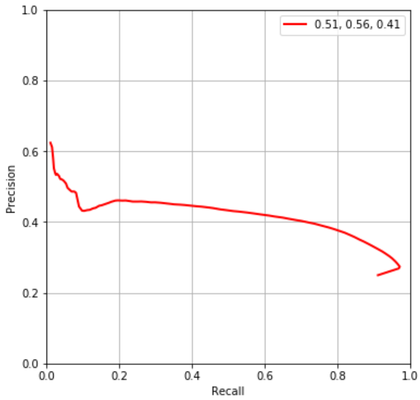


Fig 3.0 default settings

3.a: for minimizing edge artifacts I used 'symmetric' padding instead of default 'zero' padding as given. Results improvements:

```
          threshold: 0.240000
overall max F1 score: 0.542432
average max F1 score: 0.587287
            area_pr: 0.509132
```

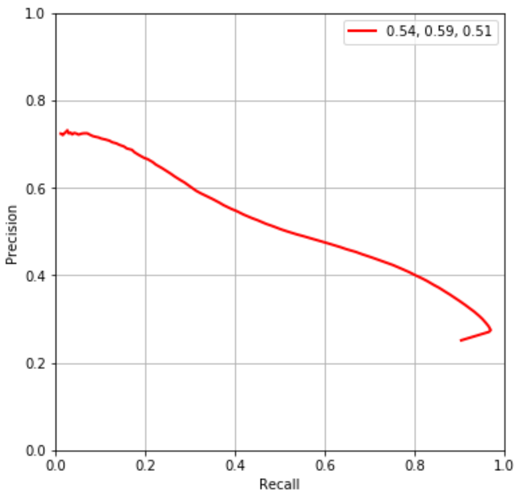
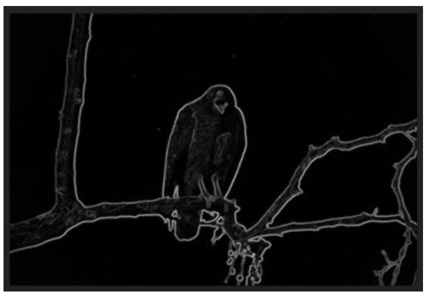


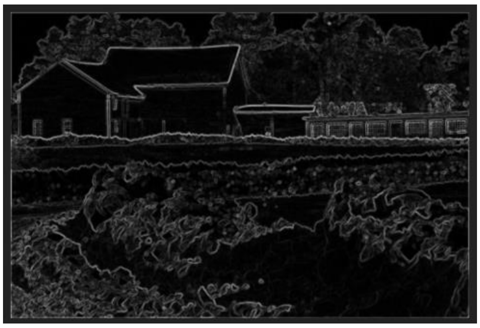
Fig 3.1 using symmetric padding

Picture improvements:

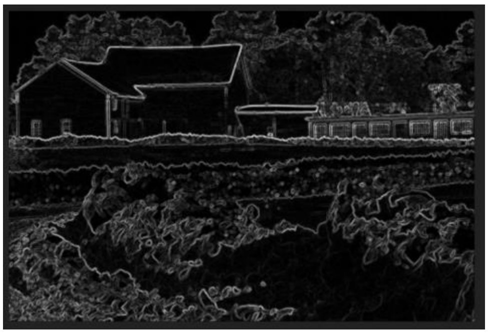
using zero padding (default)



using symmetric padding



Zero padding



Symmetric padding

3.b:

Results when using smoothing with gaussian gradients:

```
           threshold: 0.250000
overall max F1 score: 0.579702
average max F1 score: 0.618036
             area_pr: 0.572679
```

```
           threshold: 0.260000
overall max F1 score: 0.587045
average max F1 score: 0.615747
             area_pr: 0.575934
```

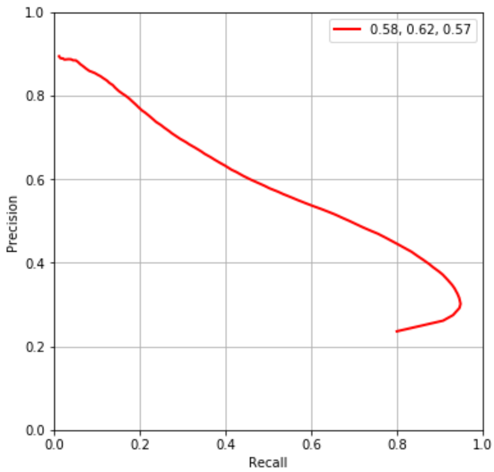


Fig3.2.1 Size=5 , sigma = 2

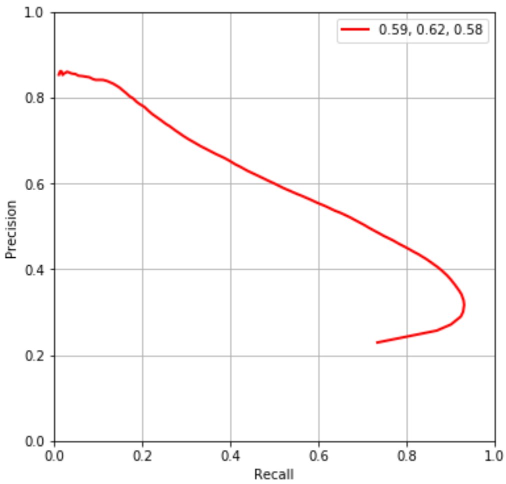


Fig 3.2.2 size=5 , sigma=5

Further increasing sigma started to decrease average max F1, thus I fixed my sigma at 5.

3.c:

I have included my non-maximal-suppression code *"support.py"*

Here the pseudo code:

1. dx , dy are the sobel gradients as provided
2. calculate gradient angle in degree by :angles = np.rad2deg(np.arctan2(dy,dx))
3. rotating -ve angles by 180^0 to ensure they represent their counterparts in positive quadrants: angles[angles<0]+ = 180 , as we are just interested in direction of slope and not it's sign
4. classifying directions:

angle ϵ [0, 22.5), [157.5, 180], we suppress along horizontal direction

angle ϵ [22.5, 67.5), we along 45^0 direction
angle ϵ [67.5, 112.5), we suppress along vertical direction
angle ϵ [112.5, 157.5), we suppress along 135^0 direction

```
         threshold: 0.260000
overall max F1 score: 0.587308
average max F1 score: 0.616519
           area_pr: 0.576140
```

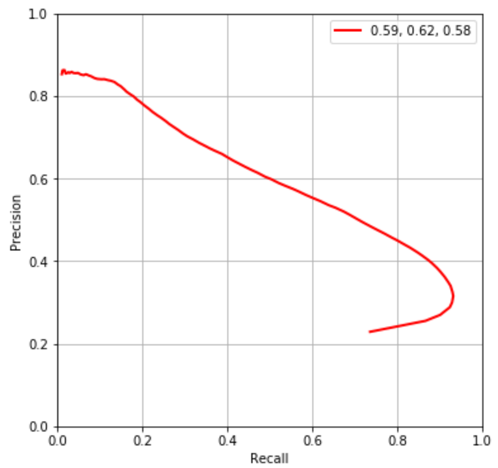


Fig 3.3.1 non-maximal suppression on 3.2.2

3.d:

1. To further improve contour detection, we can denoise image using gaussian filter before applying gradients

```
         threshold: 0.260000
overall max F1 score: 0.587381
average max F1 score: 0.616311
           area_pr: 0.576389
```

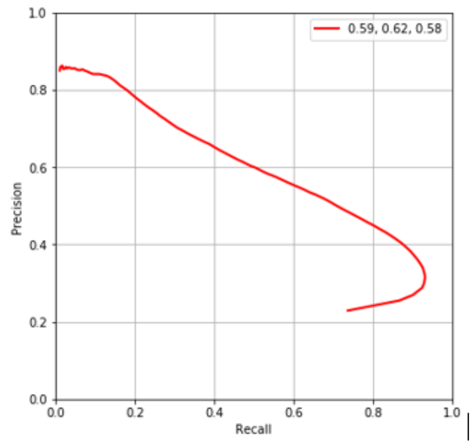


Fig 3.4.1 gaussian smoothed image + gradients

2. We can change the default gradients to sobel gradients,
i.e Gx = [1,2,1].T * ([1,0,1]*Img)
Gy = [1,0,1].T * ([1,2,1]*Img)

```
           threshold: 0.270000
overall max F1 score: 0.587766
average max F1 score: 0.618208
             area_pr: 0.576174
```

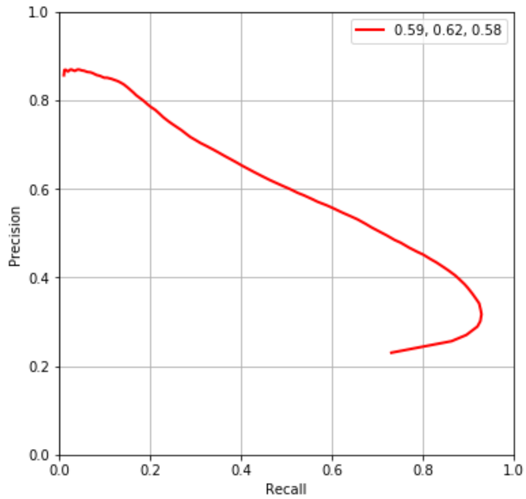


Fig 3.4.2 denoised image + sobel grads

Task 4:

1. super pixel: They are the group of pixels that are clustered/grouped based on their characteristics like intensities, color separation. Usually these are local groups i.e. pixel coordinates are also taken into account while clustering but they can be spread over an region (coordinates are not taken into account while clustering).

super-pixel representation is generally used to reduce image complexity as they partition the large image into small regions of pixels sharing similar characteristics thus image becomes a function of these regions rather than that of pixels.
It simplifies various image processing task like object detecting, tracking, image segmentation etc. as we have less variables to deal with.

SLIC algorithm (summarized) :

Selecting Features:
This algorithm generates super-pixels by clustering pixels based on color similarity and proximity. For color characteristics of pixel author chooses CIELAB color in contrast to RGB reason being it “is widely considered as perceptually uniform for small color distances”, thus difference between two lab vectors is less sensitive to small changes in fields and only deviates if there are high changes in vector properties.
Thus, each pixel now is a 5D vector of [l, a, b, x, y] , x,y being pixel cordnates.

Calculating distance b/w vectors:
Directly calculating the Euclidean distance b/w vectors was not optimal as the spatial component (x, y), is highly sensitive towards pixel location (an image is very big), thus author introduce a new distance measure

$$D_s = d_{lab} + m/S * d_{xy}$$

Where $d_{lab} = \sqrt{(lk - li)2 + (ak - ak)2 + (bk - bi)2}$ { l, a, b} of corresponding vecs
$d_{xy} = \sqrt{(xk - xi)2 + (yk - yi)2}$ { x,y} of corresponding vectors
$S = \sqrt{N/K}$; N: Total number of pixels, K: Total number of super-pixels
‘m’ is the weightage assign to proximity factor higher the m, higher D_s , thus super-pixel generated will be compact and will consists of more local pixels. S^2 denotes the spatial extent of super-pixel thus for a cluster center C_i the only pixels in the vicinity of 2S * 2S are searched, reducing the time complexity.

Algorithm Steps:
1. Initialize K clusters samples $C_k=[l,a,b,x,y]^T$ at regular grid steps S
2. Move centers in an n*n neighborhood to the lowest gradients position
Image gradients: $\mathbf{G(x,y)} = \|\boldsymbol{I}(\boldsymbol{x}+\mathbf{1},\boldsymbol{y}) - \boldsymbol{I}(\boldsymbol{x}-\mathbf{1},\boldsymbol{y})\|\mathbf{2} + \|\boldsymbol{I}(\boldsymbol{x},\boldsymbol{y}+\mathbf{1}) - \boldsymbol{I}(\boldsymbol{x},\boldsymbol{y}-\mathbf{1}\|\mathbf{2}$
Where *I(x,y) : lab vectors to pixel at coordinate x,y*
| | : L_2 Norm
3. For each cluster C_K *“Assign the best matching pixels from 2S*2S square neighborhood around the cluster”* (for the reason mentioned in above section) using the distance defined in above section
4. “*Compute new cluster centers and residual error E { L1 distance between previous centers and recomputed centers}*
5. Repeat steps 3-4 until convergence
Note: A situation may arise (rarely) where few pixels are left unconnected to any segment. Thus, as a last step we explicitly enforce connectivity by relabeling disjoint segments by with the label of the largest surrounding cluster.