ENGN4528 Assignment Report

Junlin Han

U6835134

June 27, 2020

1 Task 1

1.1 Result of f * I

I do this calculation by hand.
Result=[1,4,9,20,27,20,11].
I also write some matlab code to do this.I have attached my code along here.

```
u = [0,1,2,4,10,3,4];
v = [1,2,1];
w = conv(u,v,'same')
```

The answers are same.

```
w =
     1     4     9    20    27    20    11
```

2 Task 2

2.1 Given A and B as follows, please plot $A\,\theta B$

This task is easy but can be tricky. If we pad the boundary with 1, which is gray here, the result of $\beta(A)$ is the boundary of white pixels. If we pad the boundary with 0, it may not fit the definition of erosion.
Though Lin told me that we should pad 1, I can't agree more with her. In this question, 1 is the gray, not the normal binary image that 1 is the white. So I believe things can be different here. I will attach some reference about erosion to support my idea here.
The basic effect of the operator on a binary image is to erode away the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels shrink in size, and holes within those areas become larger. [1]
We can see that the goal of erosion is to erode away the boundaries of white pixels,which is usually 1.Hence it's suitable to pad 1 and it can ensure that we won't affect the boundaries.For our question,what we want is to erode away the boundaries of white pixels.And our white pixel is 0 here,so it's reasonable to pad 0 instead of 1.
Besides,I read some slides other than ANU.I found one picture that can strongly

support my idea.

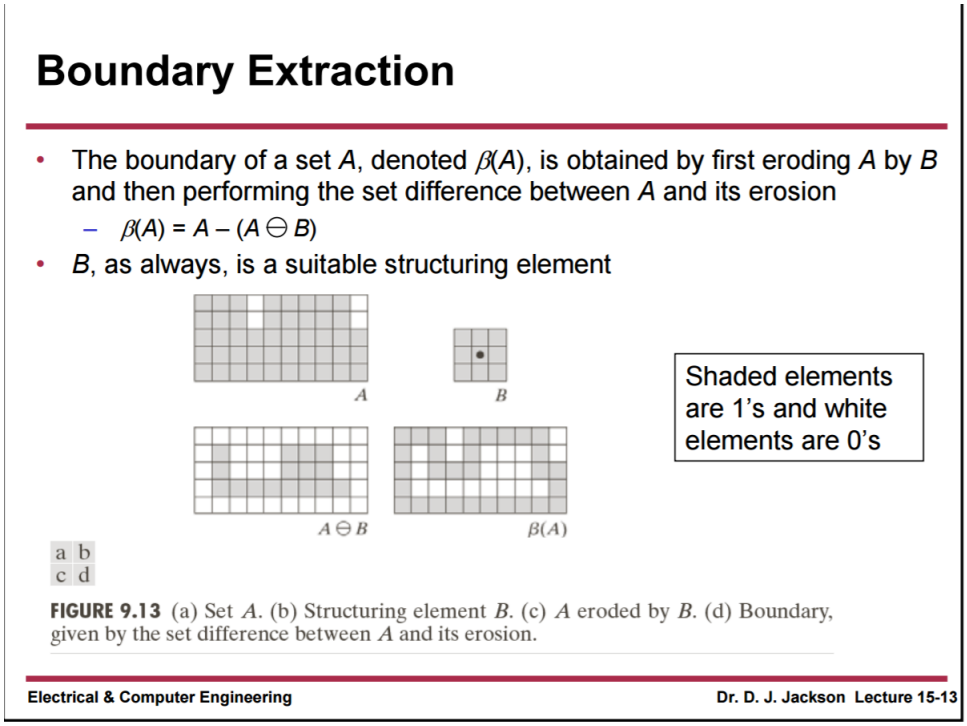


Figure 1: Erosion slide from Dr. D. J. Jackson [2]

As a conclusion,I would like to insist on my idea of padding 1.I calculate results by hand and plot them with Excel.
Result is shown as figure 2.

2.2 Please plot $\beta(A)$

Result is shown as figure 3.

3 Contour Detection

For this question, most of the code is written in python. But I used Matlab to get the groundtruth of images.

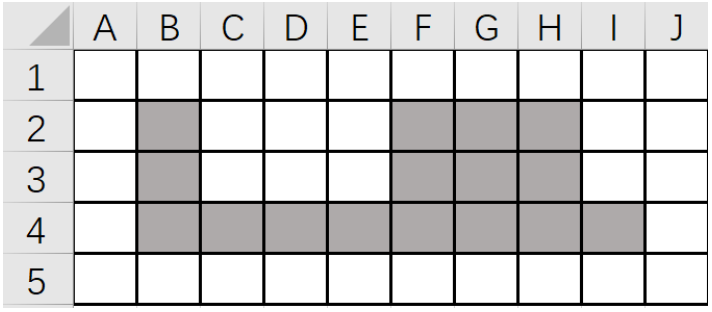


Figure 2: Plot of $A \, \theta B$

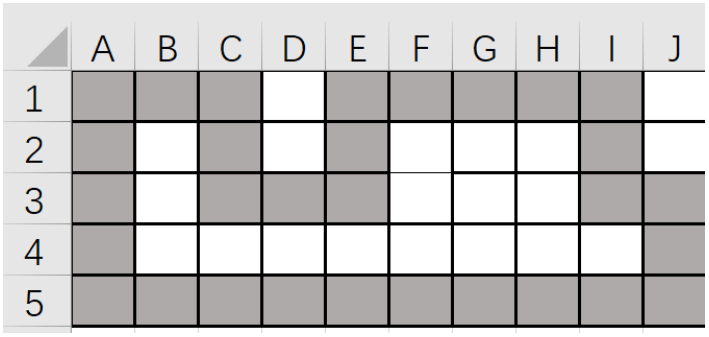


Figure 3: Plot of $\beta(A)$

3.1 Part a. Warm-up

For this question,we can modify how to handle the boundary in convolution. scipy.signal.convolve2d have 3 methods to handle the boundary.They are fill(default),wrap,symm. I will test all of them. From table 1,we can observe that symm works best with

Table 1: Difference between different boundary handle methods

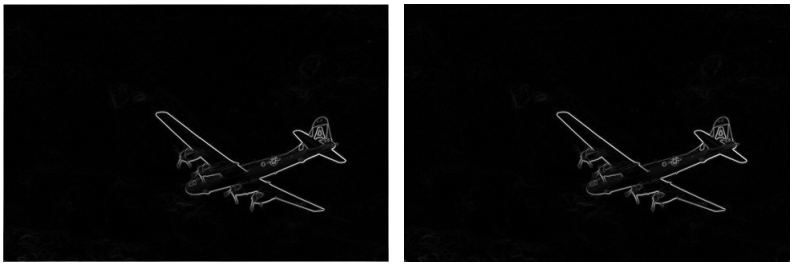
methods	run-time(evaluating)	overall max F1 score	average max F1 score	$area_{pr}$
Fill	5.24s/it	0.514330	0.563131	0.409045
Symm	4.19s/it	0.542432	0.587287	0.509132
Wrap	4.44s/it	0.527292	0.572467	0.461289

shortest run time.Symm,wrap can reduce the computation cost and symm is the

most convenient one.I will mostly use 3 images 3096(aircraft),42049(a bird in the tree),86068(2 fish in water with shadow) for comparison since these 3 images have a high representation of different object classes.***For all tables in this report, I use evaluating speed to measure run time*** since running detector is quite swift and it's hard to see difference. Evaluating speed can represent the influence after modification better. For actual run time,i.e.,the run-time speed is 4.19s/it, which means 4.19s for a single image and it will spend 50*4.29 seconds in total.
we can use code below to change the boundary handle methods.

```
1       dx = signal.convolve2d(I, np.array([[-1, 0, 1]]), ...
            mode='same', boundary='symm')
2       dy = signal.convolve2d(I, np.array([[-1, 0, 1]]).T, ...
            mode='same', boundary='symm')
```

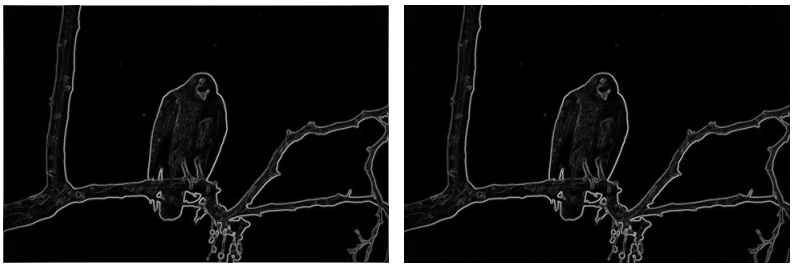
From figure 4,figure 5 and figure 6,though it's not very obvious, we can see that symm works best to minimize artifacts by zooming the example images.



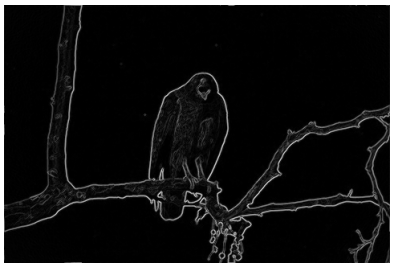
(a) default (b) symm

(c) wrap

Figure 4: Comparison of image 3096 with different boundary handle methods



(a) default (b) symm



(c) wrap

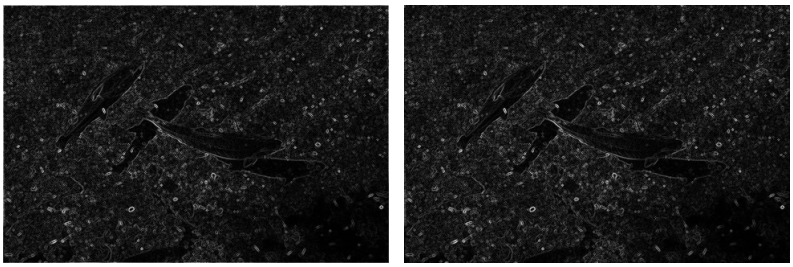
Figure 5: Comparison of image 42049 with different boundary handle methods

3.2 Part b. Smoothing

We can use code below to obtain more robust estimates of the gradient.I tried some different values of sigma and size. Gaussian filter can be separated into two directions and it can be done easily with ndi.gaussian.

```
1      #size must be an odd number.
2      size = 5
3      sigma = 9
4      truncate_value = (((size - 1)/2)-0.5)/sigma
5      dx = ndi.gaussian_filter(I,sigma,order=[1,0],
6      truncate=truncate_value) # x Derivative
7      dy = ndi.gaussian_filter(I,sigma,order=[0,1],
8      truncate=truncate_value) # y Derivative
```

From table 2, we can observe that though increasing sigma and size can improve the results. But it's not a wise idea to make them too big. When we choose sigma=10,size=5 we can get a pretty good result. It's hard to decide which one is



(a) default (b) symm

(c) wrap

Figure 6: Comparison of image 86068 with different boundary handle methods

the best since we have three performance metrics. If we focus on overall max F1 score and we can notice when sigma=5 and size=9, it reaches the highest score. For the run-time, large sigma value and small sigma value all tend to have a slower speed while larger size value tends to have a slower speed.

Figure 7,8,9 show the visual examples.We can see sigma value and width of

Table 2: Results of different sigma,size values

sigma	size	run-time	overall max F1 score	average max F1 score	$area_{pr}$
1	5	4.35s/it	0.570455	0.612028	0.562984
10	5	4.47s/it	0.579074	0.616193	0.566543
10	7	5.35s/it	0.592355	0.616289	0.576607
0.1	5	34.67s/it	0.347101	0.409234	0.114437
10	15	6.92s/it	0.568953	0.597100	0.515434
5	9	5.22s/it	0.593071	0.616045	0.577023
5	7	4.82s/it	0.590251	0.615963	0.581838

contour lines have a significantly positive correlation and size value and width

of contour lines also have a more significantly positive correlation. We need to notice that gaussian kernel with large sigma and kernel size may blur the image too much.

3.3 Part c. Non-maximum Suppression

For this question, we follow the steps given in the assignment tutorial. We fist use a gaussian filter for smoothing. Then I use a sobel filter to get the gradients and apply NMS first. The code is shown here. To implement interpolation, which is necessary since two neighbors can in the direction perpendicular to the edge. weight can be obtained by dy/dx,x, y are gradients of the pixel. We can make degrees between 0-180 by adding 180 to the negative value. Hence there are four situations we need to handle depends on degrees.0-45,45-90,90-135, and 135-180. We can get two value p,r for each situation. During interpolation, we need to test which one is the maximum value among those three values. If magnitude is greater than p,r. We can keep it otherwise set it to 0. I also set a threshold value at first that can help us to better determine the edges.

```
1     def my_compute_edges_dxdy(I):
2     threshold = 0.425
3     I = I.astype(np.float32) / 255.
4     I= ndi.gaussian_filter(I, 1)
5     #dx,dy 321*481 double.
6     #try sobel kernel
7     dx = signal.convolve2d(I, np.array([[1, 0, ...
          -1],[2,0,-2],[1,0,-1]]), mode='same', boundary='symm')
8     dy = signal.convolve2d(I, np.array([[1, 2, ...
          1],[0,0,0],[-1,-2,-1]]), mode='same', boundary='symm')
9     #mag 321*481 double.
10    mag = np.sqrt(dx ** 2 + dy ** 2)
11    mag = mag / np.max(mag)
12    #Non-maximum Suppression
13    degree = np.arctan2(dy,dx) * 180 / np.pi
14    degree[degree < 0] += 180
15    #Non-maximum Suppression
16    #602308 max
17    #45 44 43 46 41 425 435 47   6015    425   602270
18    for y in range(1, mag.shape[0]-1):
19        for x in range(1, mag.shape[1]-1):
```

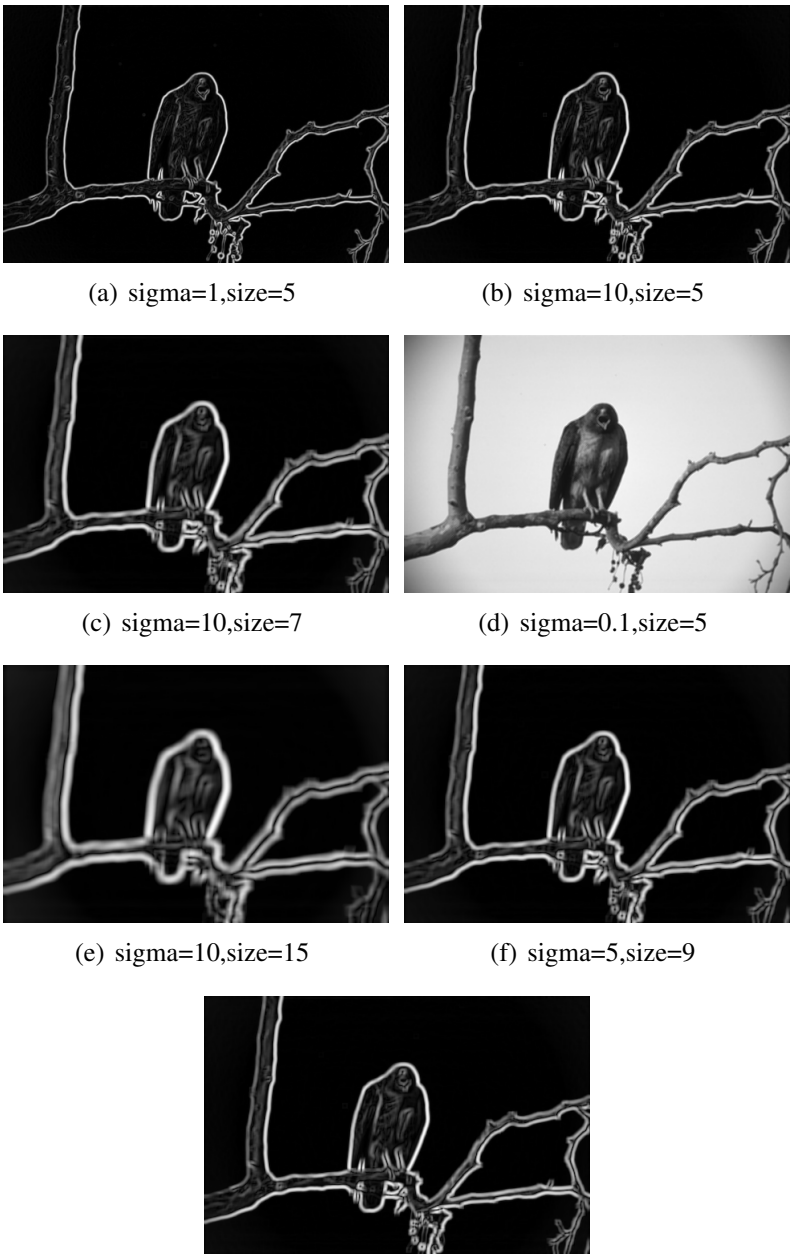
(a) sigma=1,size=5 (b) sigma=10,size=5

(c) sigma=10,size=7 (d) sigma=0.1,size=5

(e) sigma=10,size=15 (f) sigma=5,size=9

(g) sigma=5,size=7

Figure 7: Comparison of image 3096 with different sigma/size values



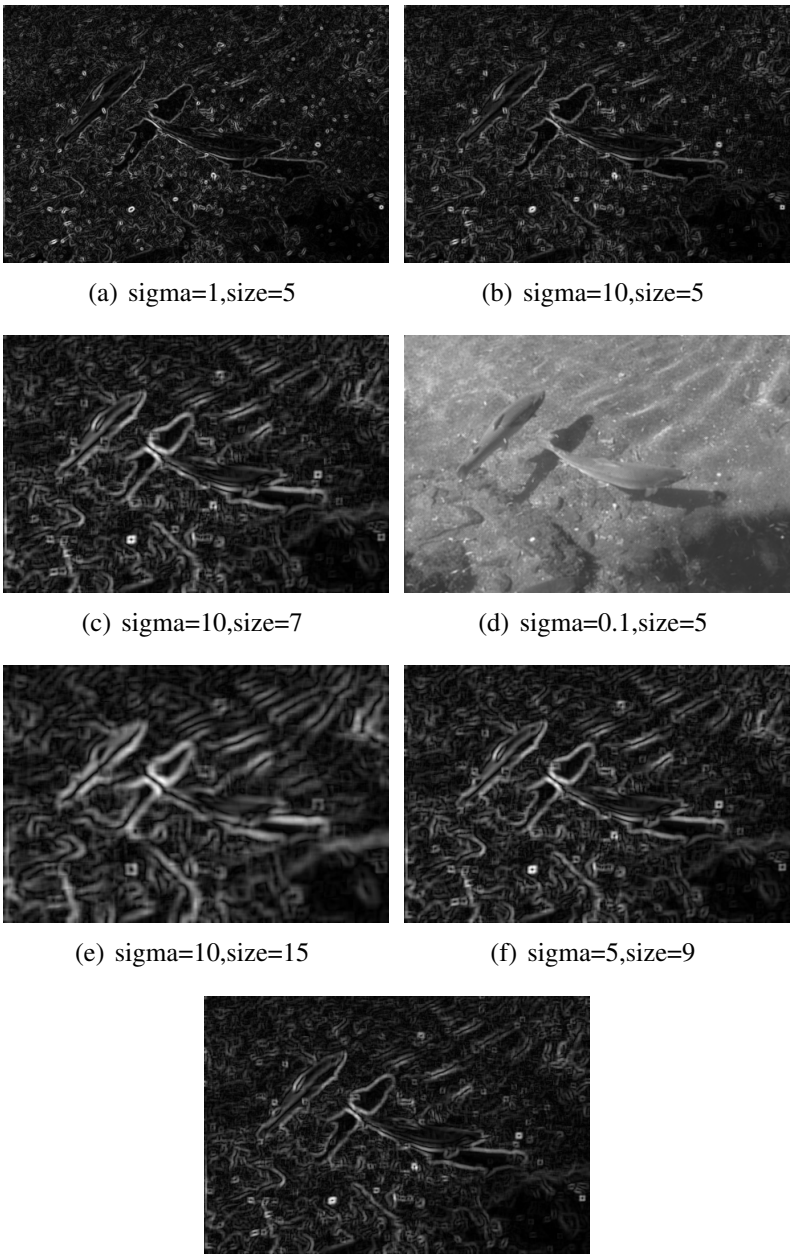
(a) sigma=1,size=5 (b) sigma=10,size=5

(c) sigma=10,size=7 (d) sigma=0.1,size=5

(e) sigma=10,size=15 (f) sigma=5,size=9

(g) sigma=5,size=7

Figure 8: Comparison of image 42049 with different sigma/size values



(a) sigma=1,size=5 (b) sigma=10,size=5

(c) sigma=10,size=7 (d) sigma=0.1,size=5

(e) sigma=10,size=15 (f) sigma=5,size=9

(g) sigma=5,size=7

Figure 9: Comparison of image 86068 with different sigma/size values

```
            if mag[y][x] > threshold:
                angle = degree[y][x]
                w1 = abs((dy[y][x])/(dx[y][x]))
                w2 = abs((dy[y][x])/(dx[y][x]))
                if (0 ≤ angle < 45):
                    p = w1 * mag[y-1][x-1] + (1-w1) * ...
                        mag[y][x-1]
                    r = w1 * mag[y+1][x+1] + (1-w1) * ...
                        mag[y][x+1]
                elif (45 ≤ angle ≤ 90):
                    p = w2 * mag[y-1][x-1] + (1-w2) * ...
                        mag[y-1][x]
                    r = w2 * mag[y+1][x+1] + (1-w2) * ...
                        mag[y+1][x]
                elif (90 < angle < 135):
                    p = w1 * mag[y-1][x+1] + (1-w2) * ...
                        mag[y-1][x]
                    r = w1 * mag[y+1][x-1] + (1-w2) * ...
                        mag[y+1][x]
                elif (135 ≤ angle ≤ 180):
                    p = w1 * mag[y-1][x+1] + (1-w2) * ...
                        mag[y][x+1]
                    r = w1 * mag[y+1][x-1] + (1-w2) * ...
                        mag[y][x-1]
                if mag[y][x] ≥ p and mag[y][x] ≥ r:
                    continue
                else:
                    mag[y][x] = 0
    mag = mag * 255.
    #set min=0,max=255
    mag = np.clip(mag, 0, 255)
    #convert to uint8
    mag = mag.astype(np.uint8)
    return mag
```

I tested it with different threshold value,0.425 is the best one. From Part b we know we have more robust estimates of the gradient.So I also tested it with Gaussian filters with different size and sigma.I tried some other filter but they work unwell.So I won't show the results of them.

From table 3, we can observe that our NMS works well and improves the quality performance of all metrics. We can compare it with table 2 and our overall max F1 score increases from 0.593071 to 0.6440531(with the same threshold 0.45, same size 9, same sigma5). ***Note that for the filter, gaussian is without pre smoothing while gaussian2 is with pre smoothing.***

Table 3: Results of NMS

Filter	threshold	run-time	overall max F1 score	average max F1 score	$area_{pr}$
sobel	0.425	3.72s/it	0.602274	0.635239	0.597780
sobel	0.5	3.80s/it	0.600498	0.632196	0.595676
sobel	0.4	3.82s/it	0.601791	0.634836	0.597697
gaussian	0.425	4.18s/it	0.6440531	0.667520	0.639758
gaussian	0.45	4.49s/it	0.643616	0.667495	0.634752
gaussian2	0.425	4.23s/it	0.644053	0.667520	0.639758

Though we reach a higher score overall. I can't totally agree the visual examples are improved. From figure 10,11,12 we can see the visual examples. The improvement of metrics can not tell all the quality of a contour detection algorithm. Gaussian filter with large size can affect the details quality of the image. But sometimes it can improve the score.Anyway, we reach a very high F score with NMS.Contour lines become clean and neat after applying NMS. We can conclude that for edge/contour detection, NMS is a very useful algorithm and produce thinner line. The canny detector is regarded as a very reliable algorithm, and one step of it is NMS. [3] This can also support our conclusion.

3.4 Part d. Extra Credit

For this part, I use a deep learning approach to do so. I use two different models with two different datasets.Hence there are four different implementations.
To output the contour of one image, we want to see the visual output thus we need an end-to-end network. Contour detection, given its axiomatic importance, researchers did a lot of work to explore it, FCN (Fully Convolutional Network) . [4]and FCN with U-net. [5] are good choices since they can do deconvolution to form a pixel-wise output. FCN and U-net model are the state-of-the-art result in many detection challenges in 2015/2016, though we have better models like RCN(RefineContourNet) based on Resnet [6],Gated-SCNN(Gated Shape CNNs) based on two-stream CNN architecture. [7]
I still want to implement CEDN(Fully Convolutional Encoder-Decoder Network) [8] for this task since this network focus mostly on contour detection instead of other tasks like semantic segmentation or edge detection.
CEDN uses Caffe as the framework, It's quite weird that I didn't find any imple-

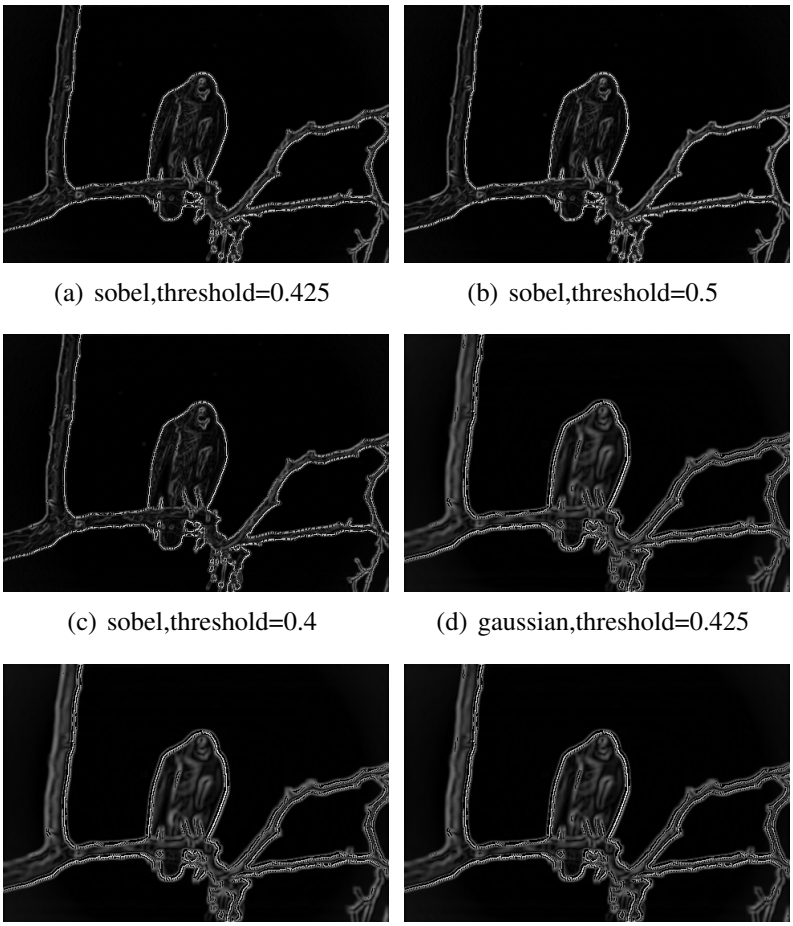
(a) sobel,threshold=0.425 (b) sobel,threshold=0.5

(c) sobel,threshold=0.4 (d) gaussian,threshold=0.425

(e) gaussian,threshold=0.45 (f) gaussian2,threshold=0.425

Figure 10: Comparison of image 3096 with different filter and parameters after applying NMS

mentations by Caffe, but I found implementations by PyTorch and TensorFlow. My code is adjusted from a Pytorch implementation [15]. CEDN uses PASCAL VOC [10] as dataset for traning.Then use MS COCO, BSDS500 [11], PASCAL VOC for testing. It's a little strange that they didn't use validation set. The biggest feature of CEDN is that it use VGG-16 net [9]as encoder, and fix the encoder parameters and only optimize decoder parameters. This allows the encoder to maintain its generalization ability and can be easily used in other tasks. The architecture of CEDN is shown as figure 13.



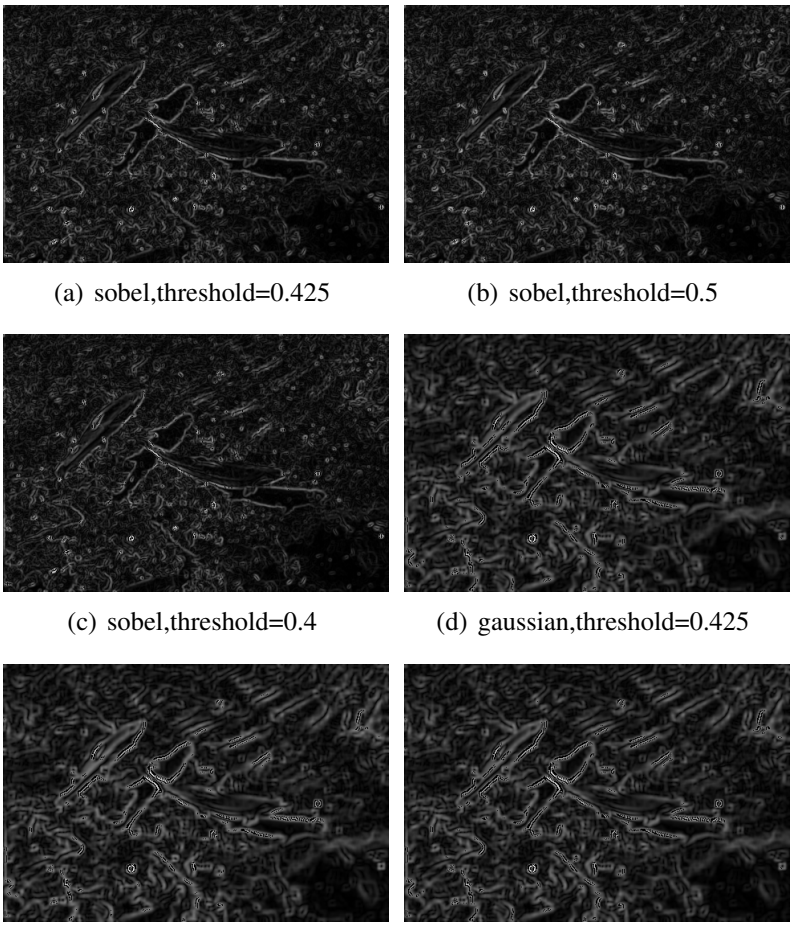
(a) sobel,threshold=0.425 (b) sobel,threshold=0.5

(c) sobel,threshold=0.4 (d) gaussian,threshold=0.425

(e) gaussian,threshold=0.45 (f) gaussian2,threshold=0.425

Figure 11: Comparison of image 42049 with different filter and parameters after applying NMS

CEDN focus on the contours of foreground objects. This is different from our contour detector,i.e., for image 86068(two fish in water with shadow). Our detector will also detect the contour of stone, shadow, and other irrelevant contours. Figure 12 shows the contours of image 86068 with NMS. I don't think they are successful since they detect a lot of things which are not what we want though the groundtruth of this image shows that it does have some irrelevant contours. I use Matlab to decode the mat data and output the groundtruth image. We can observe from figure 14 that the groundtruth does not correctly reflect the contour . We



(a) sobel,threshold=0.425 (b) sobel,threshold=0.5

(c) sobel,threshold=0.4 (d) gaussian,threshold=0.425

(e) gaussian,threshold=0.45 (f) gaussian2,threshold=0.425

Figure 12: Comparison of image 86068 with different filter and parameters after applying NMS

can't view the complete contour of the stone on the left part.
I will show the results in table 4 and discuss more about dataset later. Due to time and machine limits,I didn't train original CEDN with PASCAL VOC dataset. Every epoch on PASCAL VOC which contains more than 10000 images spends my laptop with a 1080 GPU more than an hour to train it. It's a little computation expensive for me to do so. And we know from the data of CEDN with U-net that PASCAL dataset is not suitable to our task.

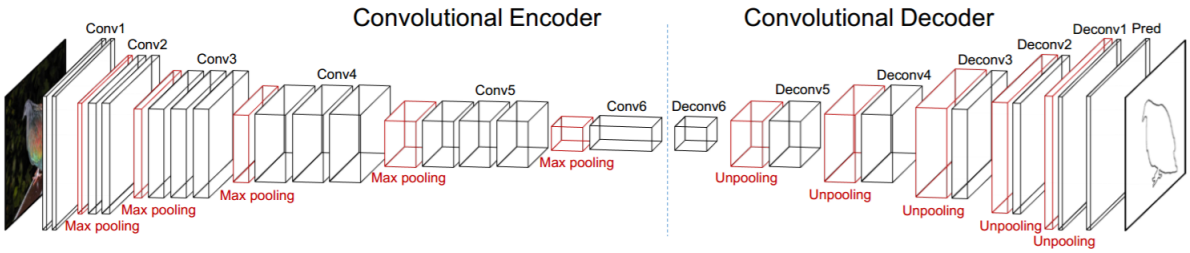


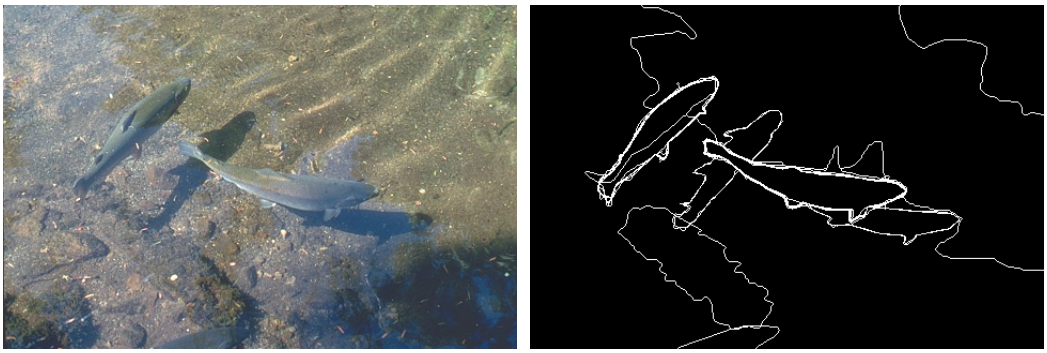
Figure 13: Architecture of CEDN

The visual results of deep learning approach are shown as figure 15,16,17. We can observe that contours obtained by deep learning approaches are clean and neat. However, they are not always continuous. From figure 15(a), we can see the contour of aircraft is very clean. PASCAL VOC dataset contains classes like aircraft. Thus we get a very optimal result. As we can see in figure 17(a), it fails to detect the contour. The ground truth of image 86068 will be treated as "back-ground" in the PASCAL VOC training set. That's why it almost detects nothing. Figure 18 shows the ground truth in PASCAL VOC dataset, which is very different from figure 14. We can see its groundtruth just care about the birds themselves and ignore all other objects like trees and clouds. So it's not a good choice to use PASCAL VOC for our task since we need to detect all the objects' contour like figure 14. Other images in PASCAL VOC all share the same property.That's why I can't get good scores with PASCAL dataset. But I do think it's a good idea to ignore the "back-ground" and focus on the main object itself. Traditional approaches can't meet such a requirement and we still need to detect the "back-ground" of images thus I tried to use another dataset for training.

From figure 15(b) and figure 16(b). Contours obtained by deep learning approaches with BSDS500 dataset keep clean and neat, but contours are not continuous. However, I think they detect the contour pretty well, the only drawback is the continuousness of lines. From figure 17(b) ,we can see the contour focus on the foreground objects and detects the fishes roughly. This is the key idea of CEDN [8]. Hence I can conclude my deep learning approach successfully im-

Table 4: Results of deep learning approach

Dataset	Network	epoch	run-time	overall max F1 score
PASCAL	CEDN with U-net	26	18.03it/s	0.389862
BSDS500	CEDN with U-net	40	16.78it/s	0.689049
BSDS500	original CEDN	15	17.28it/s	0.650544

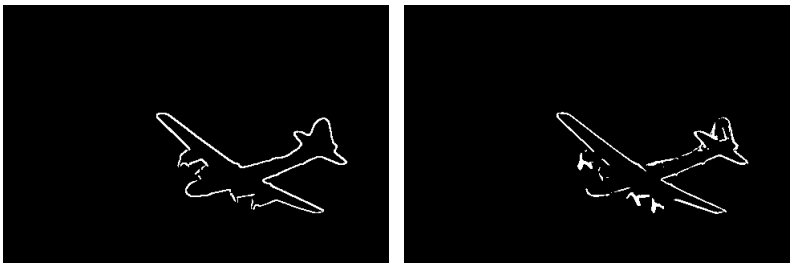


(a) image 86068 (b) groundtruth of image 86068

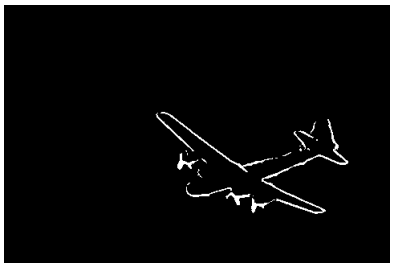
Figure 14: Image 86068 and its groundtruth

plement CEDN and reaches a very competitive F1 score. As shown in table 4, the overall max F1 score reaches 0.689. This shows our deep learning approach works well.

Reflection and discussion:
Object contours detection is fundamental for numerous vision tasks like segmentation, occlusion, and depth reasoning. It's hard to adjust traditional approaches to meet some requirements like detect the foreground and ignore the "back-ground" or effectively detect one specific object class. But deep learning approaches support different requirements and can get more robust results with proper training and hyperparameters adjustment. Though contour detection is a very mature research topic, deep learning approaches can be more competitive. Other research topics like optimal flow[13], even superpixel can be obtained with deep learning [12]. All these show that computer vision today is more likely to do research



(a) PASCAL,using U-net (b) BSDS500,using U-net



(c) BSDS500,original CEDN

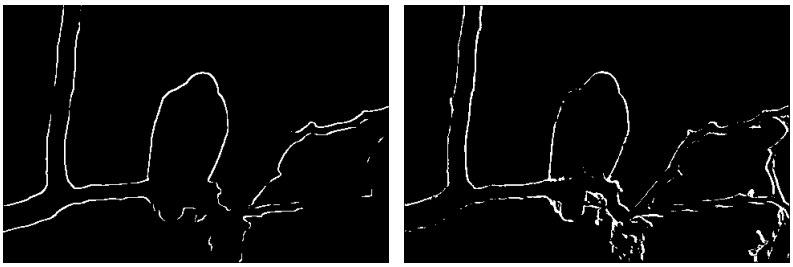
Figure 15: Deep learning result of image 3096

with deep learning.

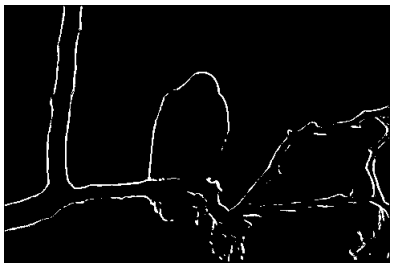
4 SLIC

4.1 Explain what superpixel is and its applications in computer vision applications

Superpixel is a combination of some pixels that share common characteristics like pixel intensity,color similarities,spacial distance,texture... thus form a 'big' pixel with better representation of characteristics.
It's useful as a pre-processing step which can speed up existing pixel-based algorithms,reduce the complexity of some image processing operations and even improve results in some cases for image processing .For specific applications,it can be applied to object class recognition,medical image segmentation.For graph-based superpixel algorithm,it can be used in body modal estimation,skeletonization and depth estimation.For gradient-ascent-based superpixel algorithm,it can be used



(a) PASCAL,using U-net (b) BSDS500,using U-net



(c) BSDS500,original CEDN

Figure 16: Deep learning result of image 42049

in object localization and motion segmentation.[14]

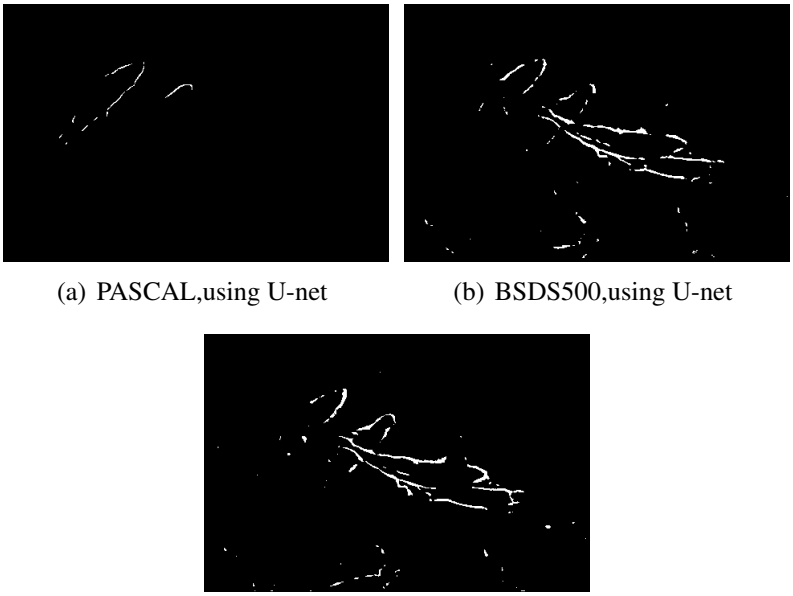
4.2 Please describe the proposed algorithm (simple linear iterative clustering) generate superpixels

The steps are literally similar to K-means algorithm.We follow those steps in algorithm 1 to generate superpixels.

References

[1] Robert Fisher, Simon Perkins, Ashley Walker and Erik Wolfart. *Hypermedia Image Processing Reference (HIPR2)* Erosion,chapter Morphology. 2

[2] D. J. Jackson. *Computer Vision Digital Image Processing*. Morphological Image Processing II,page 7. 3



(a) PASCAL,using U-net (b) BSDS500,using U-net

(c) BSDS500,original CEDN

Figure 17: Deep learning result of image 86068

(a) bird image (b) groundtruth

Figure 18: Bird image and its groundtruth from PASCAL dataset

[3] John Canny. *A computational approach to edge detection*. IEEE Transactions on pattern analysis and machine intelligence,page 679-698.1986. 13

[4] Evan Shelhamer, Jonathan Long, Trevor Darrell. *Fully Convolutional Net-*

works for Semantic Segmentation.In CVPR,2015. 13

[5] Olaf Ronneberger, Philipp Fischer, Thomas Brox *U-Net: Convolutional Networks for Biomedical Image Segmentation*.In MICCAI,2015. 13

[6] Andre Peter Kelm, Vijesh Soorya Rao, Udo Zoelzer *Object Contour and Edge Detection with RefineContourNet*.In arXiv:1904.13353,2019. 13

[7] Towaki Takikawa, David Acuna, Varun Jampani, Sanja Fidler *Gated-SCNN: Gated Shape CNNs for Semantic Segmentation*.In arXiv:1907.05740,2019. 13

[8] Jimei Yang, Brian Price, Scott Cohen, Honglak Lee, Ming-Hsuan Yang *Object Contour Detection with a Fully Convolutional Encoder-Decoder Network*.In CVPR,2016. 13, 17

[9] Karen Simonyan, Andrew Zisserman *Very Deep Convolutional Networks for Large-Scale Image Recognition*.In CVPR,2014. 14

[10] Everingham, M. and Van Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman, A *"The PASCAL Visual Object Classes Challenge 2012*. 14

[11] D. Martin and C. Fowlkes and D. Tal and J. Malik *"A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics*.Proc. 8th Int'l Conf. Computer Vision,PAGE 416–423,2001. 14

[12] Jampani, Varun and Sun, Deqing and Liu, Ming-Yu and Yang, Ming-Hsuan and Kautz, Jan *Superpixel Samping Networks*.In ECCV,2018. 18

[13] Xiang Pan, Tianyu Zhao, Minghua Chen *DeepOPF: Deep Neural Network for DC Optimal Power Flow*.In arXiv:1905.04479,2019. 18

[14] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine S¨usstrunk *SLIC Superpixels*.In EPFL Technical Report 149300,2010. 20

[15] dlam4h *OCD-PyTorch*.Code from github,dlam4h/OCD-PyTorch,2019. 14

Algorithm 1 SLIT generate superpixels

Input:5-D lab image I, desired number of superpixels N , compactness m.Optional inputs are iteration times k and threshold value thresh,one of them is enough.
Default value: m=10 and k=10.
Output: Label matrix L.
Complexity: O(N).

1. Compute grid interval $S = \sqrt{Number\ of\ pixels\ of\ I/N}$

2. Define a distance D_s measure for 5D space.
   $d_{lab} = \sqrt{(L_k - L_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$
   $d_{xy} = \sqrt{(X_i - X_k)^2 + (Y_k - Y_i)^2}$
   $D_s = d_{lab} + \frac{m}{S} dxy$

3. Choose N superpixel cluster centers C_k,$C_k = [l_k, a_k, b_k, x_k, y_k]^T$ and moving centers to seed locations corresponding to the lowest gradient position in a n * n(default 3 * 3) neighborhood.

4. Create Label matrix L to store the assignment of every pixel for each center and a distance D to store the D_s measure by calculating the D_s distance of pixel to it's superpixel center.

5. For every superpixel center C,assign the best matching pixels to minimize D within a 2S * 2S area around the center C on the xy plane.If D is smaller than it's D_s of original centers,it means this pixel belongs to C.Update L and D.

6. Recompute new cluster centers and residual error E,the L1 distance between previous centers and recomputed centers for every superpixel centers.

7. If k is given,repeat steps 5 and 6 until k times.Else if thresh is given, repeat steps 5 and 6 until $E \leq thresh$.

8. Return L.