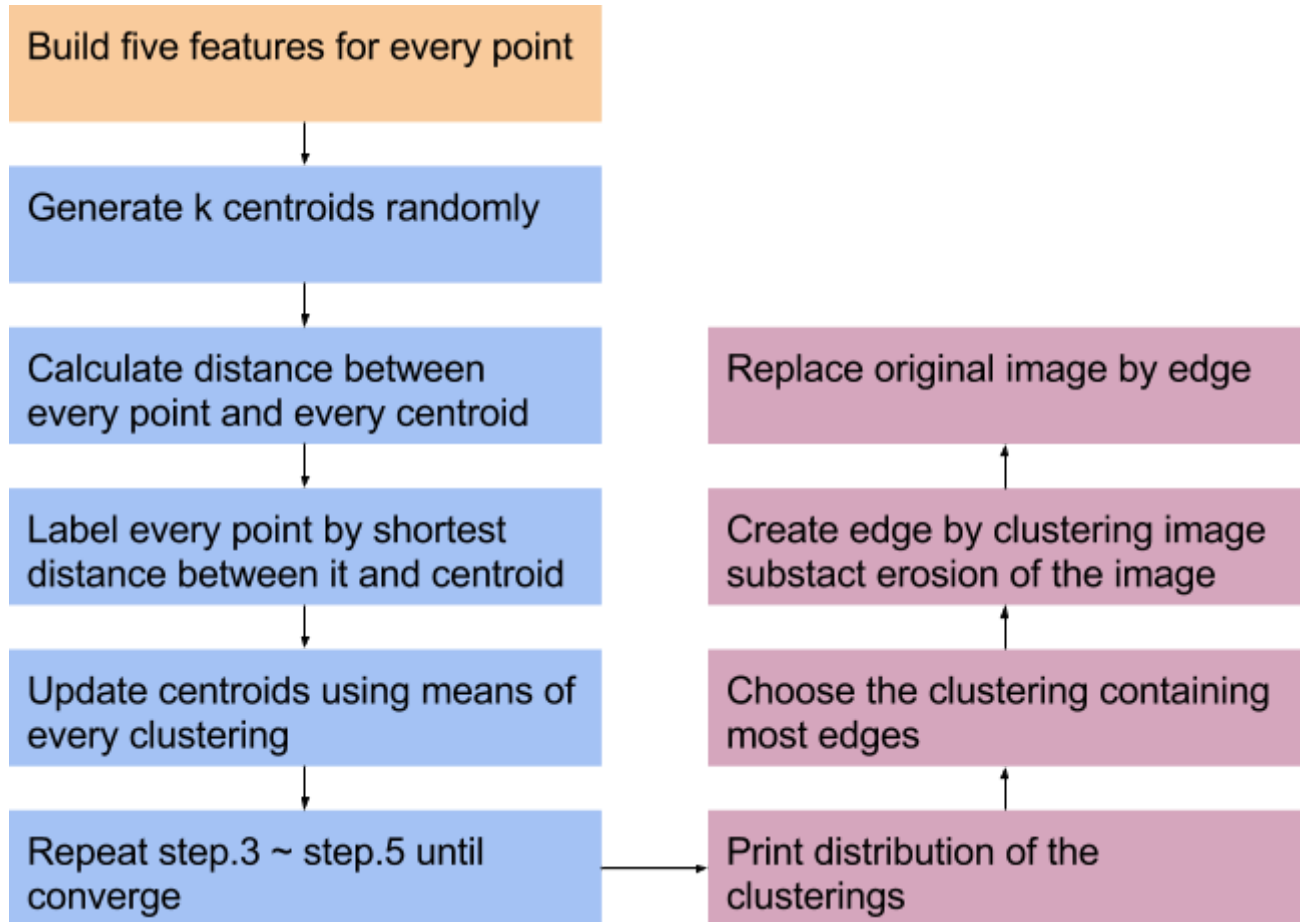


# Digital Image Processing HW #4

0310139 曾敏原

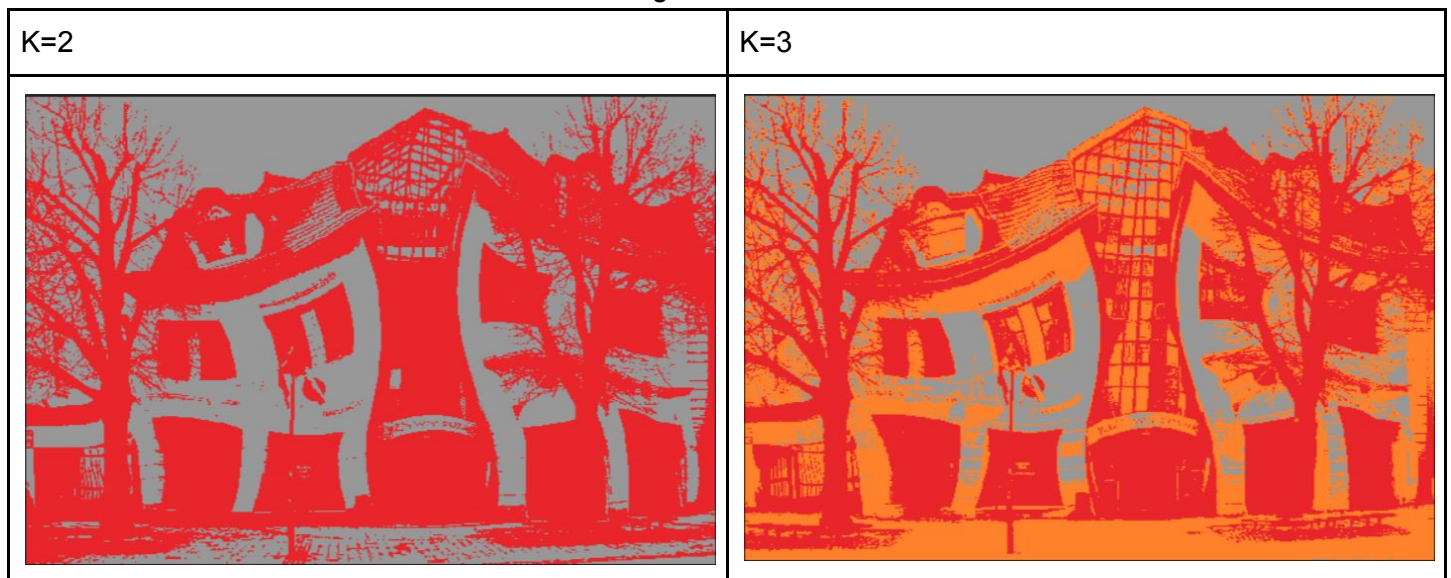
## KMeans Clustering

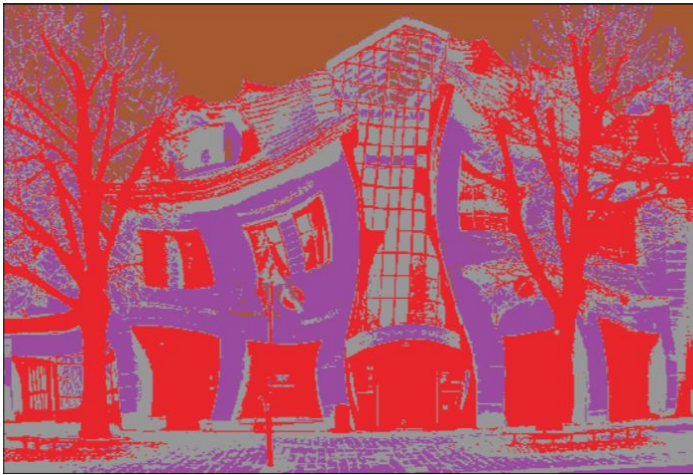

I use five features for every point to do clustering. The features are normalized location (x, y) pair and Lab color space. The following is the procedure of the algorithm.



### Result and Test





First, I will show how K makes influence on the segmetation.



K=4	K=10
	

From above table, we can conclude that the larger K we set, the more details we get. However, the "details" are not always good. We can find this situation from K=10, there are too many clusterings in the image and leading to a bad performance. Therefore, we must choose a K that can get enough details and not affects the performance. In this homework, I choose K=4.

Now, we already have K and distribution of the clusterings, so starting the generate the edges from it. In order to find edges, I first plot all clusterings with different label and look for which clustering contains the most number of edges.

Label=0	Label=1
	
Label=2	Label=3
	

We can find there are most edges when label=1, so I will use it to build edge image by the following equation.

$$Image_{edge} = Image_{clustering\_k} - erosion(Image_{clustering\_k})$$

where *erosion* is a function that can erode an image slightly



After performing the above equation, I get the following result. We can also increase the level of erosion and get more thick edges.



Finally, I just replace the original image with edge image and get the following result.



### Discussion

Advantages:

- KMeans clustering is a general clustering algorithm and easy to be implemented.
- Get a good performance when using a suitable K.

Disadvantages:

- K must be chosen carefully in order to get good performance.
- Using only one clustering to generate edges will miss some edges existed in other clustering. Need an algorithm to merge the edges from different clusterings

## Canny Filter

Step 1.

Convert RGB image to Gray image: for this part, I will perform canny filter on gray image.

Step 2.

Use Gaussian filter to blur the image. The larger the width and sigma of the Gaussian mask, the lower is the detector's sensitivity to noise. The example of gaussian filter is shown below.

$\frac{1}{273}$ 

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

5x5 example

Step 3.

Use Sobel filter to get gradients on x axis and y axis. Sobel filter is shown below.

-1	0	+1
-2	0	+2
-1	0	+1

G<sub>x</sub>

+1	+2	+1
0	0	0
-1	-2	-1

G<sub>y</sub>

After **G<sub>x</sub>** filter, we get an image **I<sub>x</sub>**. After **G<sub>y</sub>** filter, we get an image **I<sub>y</sub>**. Then calculate the magnitude by following equation for future use.

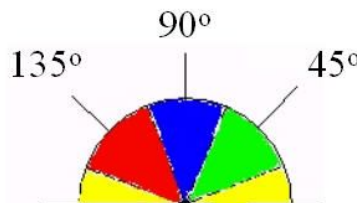
$$|I| = |I_x| + |I_y|$$

Moreover, we need the direction of the gradient on every pixel. We can calculate angle of the gradients by following equation.

$$\theta = \operatorname{arctan}\left(\frac{I_y}{I_x}\right)$$

Step 4.

Once we have magnitude and direction for each pixel, we can start to generate the edges in the image. For every pixel, it only has four direction like the figure below.



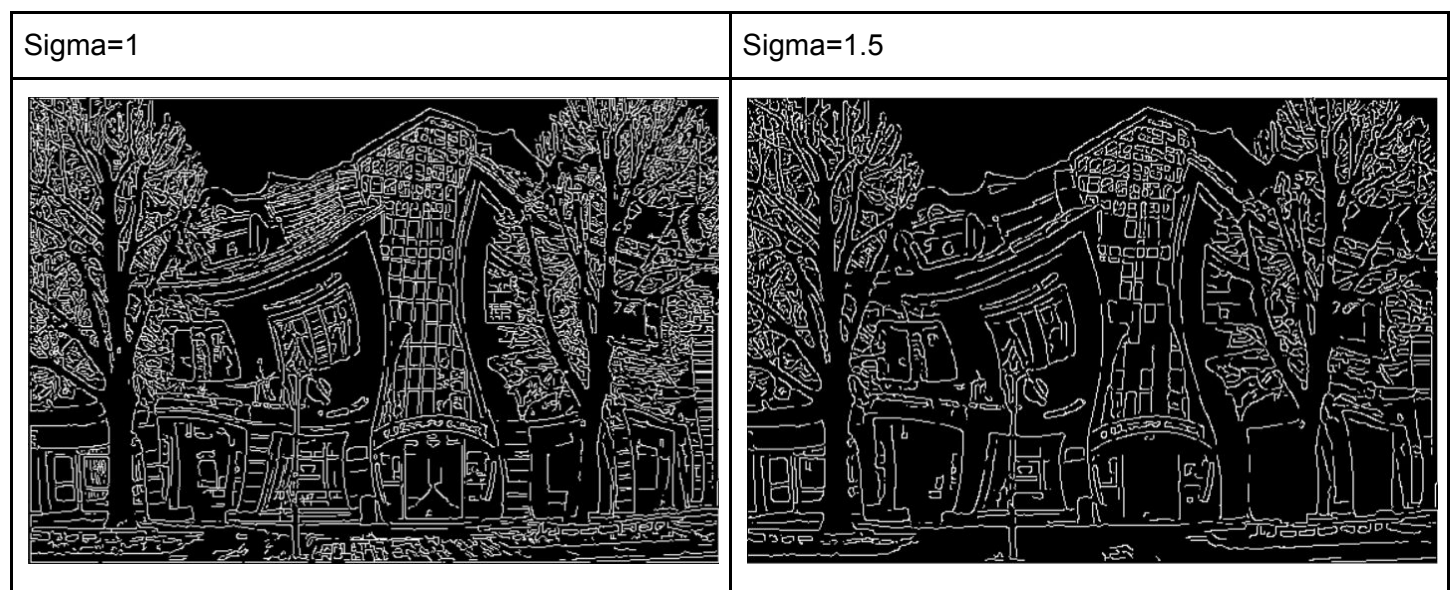
After the edge directions are known, non-maximum suppression now has to be applied. Non-maximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (sets it equal to 0) that is not considered to be an edge. This will give a thin line in the output image.

Step 5.

Finally, hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold.

## Result and Test

In order to find how sigma of gaussian filter affect the result, I try some different sigmas. Here is the result.





Sigma=2



Sigma=3



We can conclude there is a tradeoff using canny filter. The larger sigma, the more details can be found, but noises will affect the performance. The smaller sigma, the less details can be found, but influence of noises decreases.

For this homework, I choose sigma=2 and here is my final result.



Although it can find almost all edges in the picture, there are some things miss like windows of the house.

## Discussion

Advantages:

- The concept of canny filter is easy to understanding
- The execution speed is faster than other method

Disadvantages:

- Need a fine-tuned sigma to achieve the best performance
- The performance is not very good, some edges can not be detected
- Implementation is a little bit complex
- There are some edges appear in the edge of image. It can be solved by setting a mask on the edge of the image

## Threshold Based Method

In this part, I follow the procedure of the slides to implement this method.




## Basic global thresholding

1. label each pixel as object or background depending on whether the gray level of that pixel is greater or less than the threshold.
2. useful in highly controlled environment.

### Iterative Algorithm



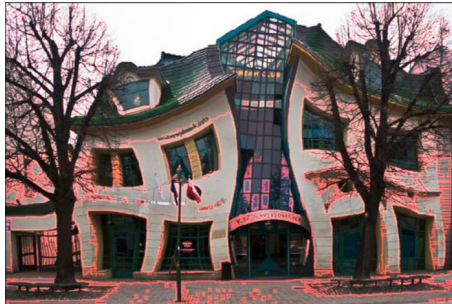
1. Select an initial estimate for the global threshold  $T$ .
2. Segment the image using  $T$ . This will produce two groups of pixels:  $G_1$  consisting of all pixels with intensity values  $> T$ , and  $G_2$  consisting of pixels with values  $\leq T$ .
3. Compute the **mean intensity** values  $m_1$  and  $m_2$  for the pixels in  $G_1$  and  $G_2$ , respectively.
4. Compute a new threshold value:  $T = 0.5(m_1 + m_2)$ .
5. Repeat Step 2 through Step 4 until the difference between values of  $T$  in successive iterations is smaller than a predefined parameter  $\Delta T$ .

I will perform threshold based method on global image and several local images respectively. Here is the result of performing of global image. The initial threshold  $T$  will not affect result obviously.

Binary image after threshold	Edge image after erosion	Result
		

Directly using threshold based method of entire image get a bad result. A lot of noises are seen as edge. I think it is because this image is too complex to using only two value to represent it. It is like KMeans method with small  $K$ . A lot of details lose by this method. Therefore, directly using threshold method of entire image is not a good idea.

A modified way is to split the original image into several parts then performing threshold method on each part of image. I split the image into four parts and get the following result.

Binary image after threshold	Edge image after erosion	Result
		

Obviously, threshold based method of several local images is better than of entire image. We can find some noise in "binary image after threshold". We can use a median filter to filter out the noise and get better performance.

## Discussion

Advantages:

- The execution time is fast
- The initial value do not affect the result seriously
- Have a good performance when performing on several local images

Disadvantage:

- Directly using this method of entire will lead to a bad result

- Need to split original image into several parts
- Some noise will occur. Using median filter to get rid of it.

# Deep Learning

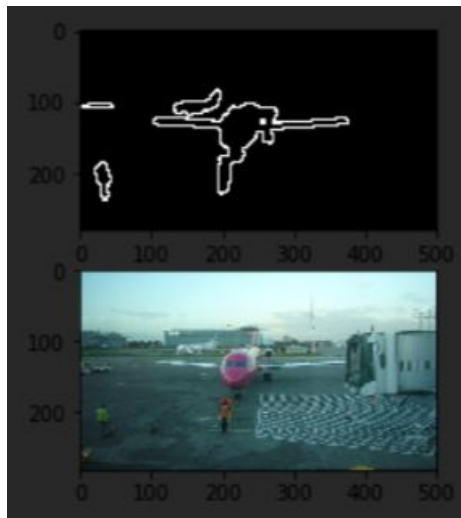
"Image Segmentation" is a very popular issue in deep learning. For this homework, I try to implement fully convolutional network(FCN) proposed by [1]. Original FCN is used to image segmentation. I simplify it largely because we just need edge detection for this homework.

## Dataset

VOC dataset from [Visual Object Classes Challenge 2012 \(VOC2012\)](http://www.robots.ox.ac.uk/voc2012/)

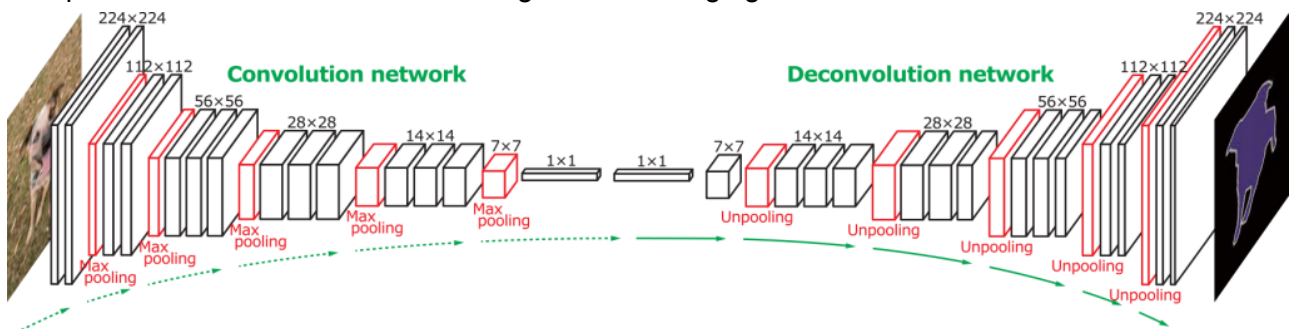
## Data Preprocessing

Because I want to detect the edge of an image, I discard the labels for classification and leave the edge of each object in the dataset. Here is an example after data preprocessing.



## Model Architecture

The concept of FCN can be understood through the following figure.



(ref: <http://rnd.azoft.com/object-detection-fully-convolutional-neural-networks/>)

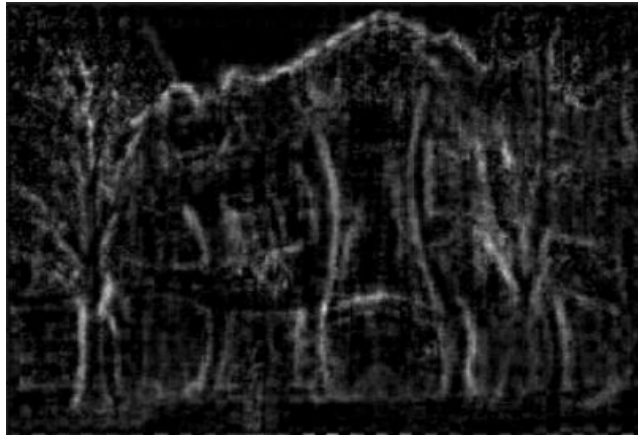
Convolution network is also as known as "feature extractor", and deconvolution network is used to "upsample" to image in order to generate prediction from lots of feature maps.

Typically, we use some pre-train models(e.g. VGG, ResNet...) to build the convolution network. I also do some residual operation to feed some output of convolution network to input of deconvolution network. The loss function I use is mean-square-error between output of model and target image.

## Result

After training 8 epochs on 1464 images, I get loss 0.0273. I feed input.bmp into the network and get the following result.





It seems it can catch some edges of the image, but I encounter a problem that how to extract edge from this image.

### Discussion

Although I get a poor performance by deep learning, it is still a powerful tool in image segmentation. Here are my ideas why I can't get a good performance by deep learning.

1. Targets are all sparse matrix

After data preprocessing, I get targets of training that contain only edge with 1 and other region with 0. This situation will cause the network prefer the output with all zero when loss function is MSE.

2. Training data and epoch

In order to finish the homework quickly, I use a small data and short epoch. However, this data size and epoch are not enough for deep learning

3. Simplified model

Original model use cross-entropy as its loss function for classification, but I use MSE to detect the edge. It may be the **main** problem of poor performance