Duke University

Blurring and Thresholding in Face Detection

Xianjue Huang

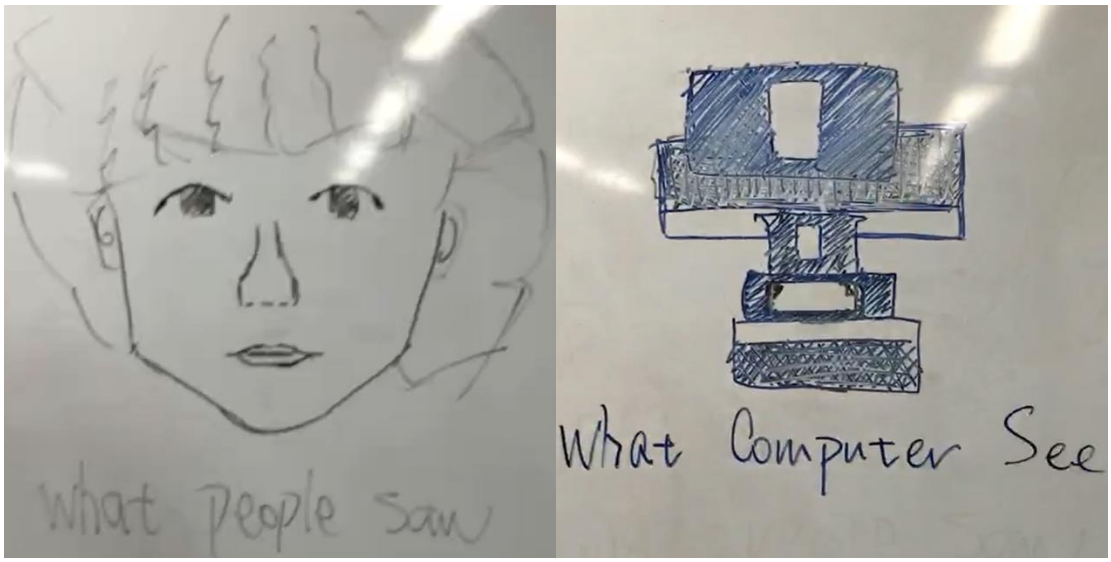
Math of the Universe

Hubert Bray

17 July 2017

Introduction

Face detecting is a common activity that involved in every human being's daily life, as before they can recognize who they are facing to, they need to find out where are the faces right? Another interesting thing is that there are also many phone applications that can add cute nose, ears and eyes to a person facing the screen automatically, and in both cases, face detecting is enrolled. It might seem to be a pretty easy thing for people to find others' faces because they can distinguish faces and other stuffs by finding the differences between the colors, shape, and the organs on faces (apparently, organs are different from stones); however, things get much more difficult for computers. What the computers can understand are series of 0 and 1, so it is like daydreaming to ask the computers to "see" the same world. Fortunately, scientists had invented algorithms to solve such problem. And many had been written into codes for people to use.



"What People See" VS "What Computers See"

Recently, some of my classmates chose face detection as their topic of project-based-learning, they utilized the code library called “OpenCV” to do their work. In the codes they study, the face detection uses the idea of blurring and thresholding to help find the faces in the picture or video.

Basic video and picture concepts

To understand how computer detect faces, the acknowledgment of concepts of video and picture are essential as videos and pictures are the most possible and most used ways to give the computers inputs. Without understanding the concepts of picture, confusion may come up in the coming explanations.

When camera captures a video, it does not capture a video, but captures a series of pictures, each picture named as “frame”. The smoothness of the video depends on the FPS, “frames per second” of the video. Basically, the higher the FPS is, the smoother the video is, because with more pictures showed to people within one second, they cannot see the time gap of pictures changing. Imagine, you are given a video with 4 FPS. This 4 FPS video is just like showing you pictures quickly (four pictures in one second). However, you can see the actions in the video smoothly in a video with 60 FPS. Similarly, it is easier to detect face when camera capture faces with higher FPS.

Computers view the picture as a two-dimensional-array that include every small part of the picture. The little parts in the pictures are called pixels, each

composed by three color channels, the red, the green and the blue. The scale of each channels is the “grayscale”, with only white, gray and black as colors,

so that the brightness of each channel indicates how red, green or blue the pixel will be. The brightness is measured by numbers from 0 to 255, so if the brightness is dark, the numbers is somewhat near 0. By contrast, numbers near 255 are presenting bright one. Lastly, when pixels are arranged up together, they form pictures.

The pictures above shows the red, green and blue channel of a picture of blue sky. Take the left first one to explain, the sky part is almost black or in dark color range, that is because blue contains very little red color. Thus, the

magnitude of the sky part of red channel is in the range from 0 to 100, and the cloud part would be in the range from 200 to 255.

In face detection projects, people would more likely to convert the picture and video in to gray ones, because it decreases the work that computers need to do. The gray pictures are also range from 0 to 255.

Note: From now on I will only talk about detecting faces in pictures since video are picture collections, and the pictures will be the gray pictures.

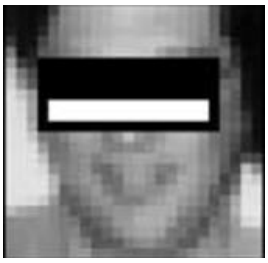
Blurring and Thresholding

Face detection by computers takes lots of work. First of all, human face is an object for sure, but finding it will be hard. After all, videos and pictures are two-dimensional-array for computers, so if you detect the face by its shape, it could lead to some ridiculous error like detecting a head-shaped stone or American football as a face. Then, even if you can detect a face, it is still hard to find where, let us simplify it, the eyes will be.

To overcome those difficulties, finding what should be the useful part for determination is the goal.

(1)

The picture above shows a blurred face. What we can distinguish is that his hair is black, his eyes, mouth and noses have clear contours (edges), and that is probably everything that we can derive from this picture. However, as shown on the right, the first most obvious feature on a person's face is that the eyes, compared with the region below the eyes, are darker. Then, the eyes regions are darker than that of the nose. (1)



(1)

(1)

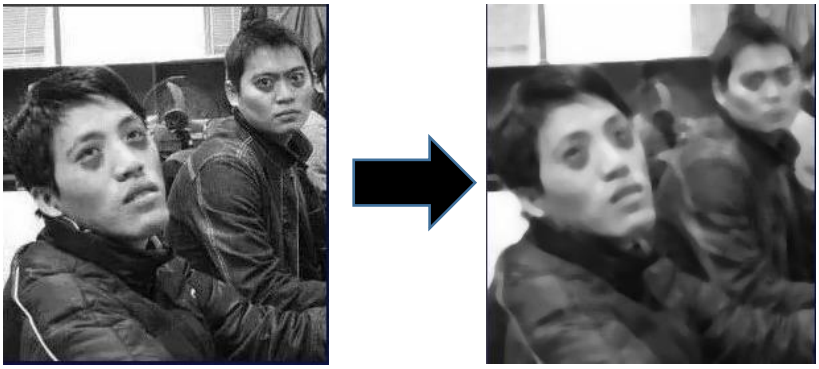
So basically, programmers are trying to find some invented algorithms that can do such distinctions. And the OpenCV was created.

Remember as what I mentioned before, as this face is blured? That is the first essential idea and technique that laid the fundation of successful face detection.

Image Blurring (Smoothing) (2)

This blurring idea came to stage when people realized that in a picture, many things are unrelated to the face, and blurring can help to neglect those noises. Also, a lot of parts on a person's face are not needed to concern a lot, like the hair. So blurring can help to wipe out those noises too. Furthermore, blurring can also make the dark area darker and wider, so that the

comparison between the region below the eyes and eyes and the nose are easier to find.



The two picture above showed how effective blurring is. Take the guy in the left for instance, the white line on his arms almost disappeared after blurring, and the eyes, compared with the regions around it, are much darker. Thus, by blurring pictures, the noise in the picture is reduced, and the area that the computer need to detect becomes more sensitive.

There are many kinds of blurring algorithms, and here, I want to talk about one algorithm with easy idea and one algorithm with higher practicability. Remember from the basic concept part, computers see pictures as collections of pixels? But it is still a big work load for computers. So programmers set the collections in sections, and created the “Kernel”, being used to calculate new pixels.

1	-2	1
2	-4	2
1	-2	1

(3)

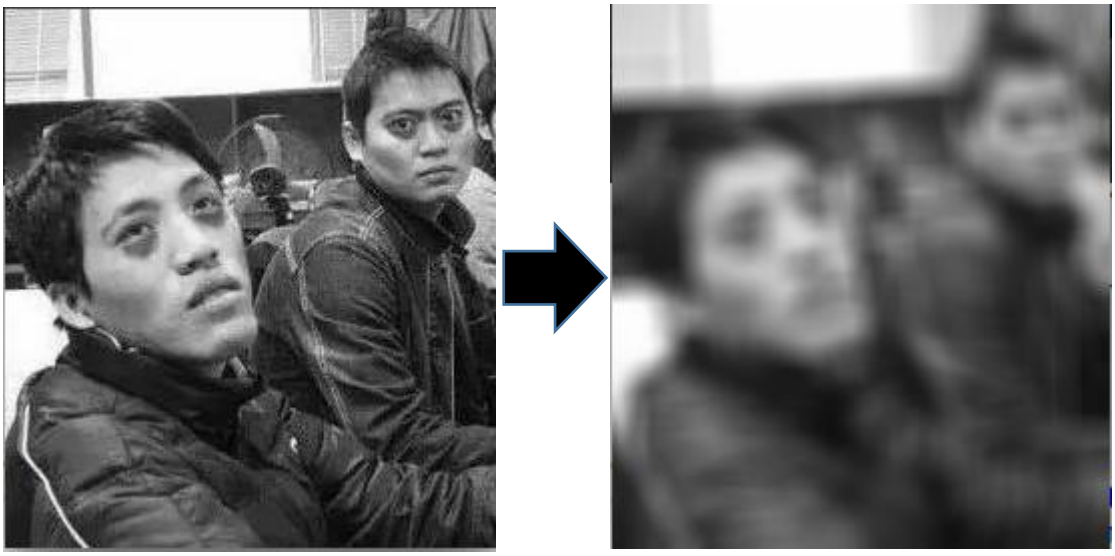
The picture shows a small “Kernel”, and it is essentially a fixed size two-dimensional-array of numerical coefficients along with an “anchor point”, the midpoint. Later, I will use “K” to represent “Kernel”. (3)

The first blurring I want to talk about blurs a picture with the idea of non-weighted averaging. (4)

As the formula on the right shows, every K is replaced by the mean value of all the values inside itself. In this blurring process, every single point has the same weights as the others. The drawback and reason for this kind of blurring being not so much used in processing large pictures are just simply because the weight of each values is the same, and because every value it treat weights the same, blurring might affect too much on the original to keep the edges or important features for objects. (4)

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

(4)



On the picture above, I used the parameter as (15, 15) for the blurring and it almost made the eyebrow for the first guy merged with his eyes, so that the edges of the eyes changes. Even though this blurring can reduce the noise, it changed the original too much.

However, another blurring algorithms came to the stage with the utilization of Gaussian function:

Gaussian Blur (4)

This algorithm utilized the idea of weighted-averaging, which means that the center of the values in the K varies by its location. (5)

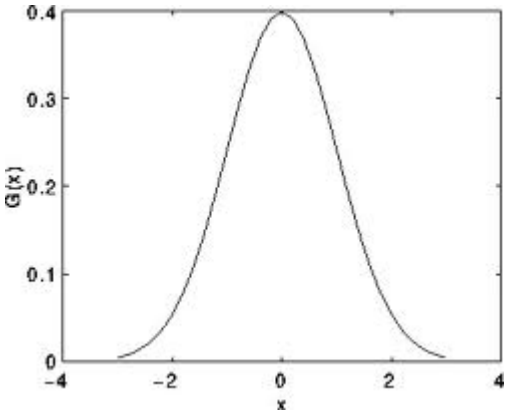
The parameter for this Gaussian Blur is the same as the last one (15, 15); however, the effect is much better than the last one. In this blurring, it maintains the edges, but with much less sharpness; it wipes out lots of noises, but keeps the important features. But how did this blurring actually made it? How is the data being weighted?

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (5)$$

The formula above shows how the weights for each point being calculated, where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, σ is the standard deviation of the Gaussian distribution, which will be decided by users. (5)

(-1, 1)	(0, 1)	(1, 1)
(-1, 0)	(x, y)	(1, 0)
(-1, -1)	(0, -1)	(-1, -1)

Here, suppose there is a K with 9 values, just like the list sits at the left, take the center as the position (0, 0), then the left one is (-1, 0). Then let us assume that the standard deviation of the Gaussian distribution, σ, is 1.5, so the weight of point (-1, 0) can be calculated by the formula, and we can find the answer as 0.05664, similarly, the weight of point (0, 0) is 0.07074. To only concern these 9 values, the sum of weights must be 1. Right now, the sum of the calculated 9 values is 0.4787147, so every value need to divide the sum, so the weight of point change to 0.147761. Then by multiplying every related grayscale value to its weight and then sum them up, the values for the center pixels will be recalculated. All of these weights ratio would fit in the chart on the right (only showing x direction). (6 – Translated what the web stated)



(4)

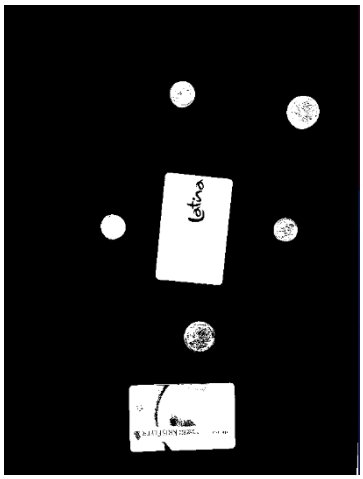
Furthermore, it can be concluded that the farther away the position lies from the origin, the smaller the σ will be. And after having these weights, the calculation almost bares the same idea as the non-weighted-averaging blurring.

Gaussian Blurring is mostly used when needed to reduce the noise or even the size of the picture, which matches the request for the face detection project, because after the blurring process, spurious high-frequency information will be greatly reduced or even won't appear. So much for blurring, the first big step for face detection is complete. (5)

Thresholding (7)

Another big step for face detection is "thresholding", meaning to convert the picture into black-white picture, so that the whole picture contains only the value of black, 0, and the value of white, 255. By doing this, the region for the important features will be much more detectable, and further reduce the noise.

So, how does computers made this binarization? The idea is that you give the computer a color value line, (in this face detection case, the grayscale value will be given) and the computer compare the value of each pixel to the line. Then, based on what the user like, it will convert the values above and under its line into black or white. As shown in the right, the background of the picture is white, so I made a line value of 200, so everything beyong the value 200 is converted in to black, and others into white (Just showing effect, actrual experience).



The most commonly used value for the color line is the integer value of (255 / 2) which is 127. But this value is not always that useful when detecting faces. Imagine, if a woman uses some APP on her phone to increase the lightness of her eyes to look prettier, then this 127 value will not be able to be the determine line. Therefore, here I want to introduce a better idea which can be applied in almost every face: Otsu's Binarization.

Otsu's Binarization / Method (8)

This kind of binarization is known as the best algorithm to find the threshold value. It has the feature of simple calculating, regradlessness of the brightness and the contrast. It divides the whole picture into two part, one is

the background, and the other one is the object by the value of grayscale. The formula of this algorithm is listed below.

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \qquad (8)$$

The w0 and w1 (cannot find the sign) are the possibility of the background and the object collection that separate by the threshold value t, and the σ represent the variances of these collections. By having exhaustive search, the threshold value that minimize the collection variance will be calculated. How w0 and w1 are calculated are listed under the formula. (8)

$$\mu_0(t) = \sum_{i=0}^{t-1} i\frac{p(i)}{\omega_0}$$

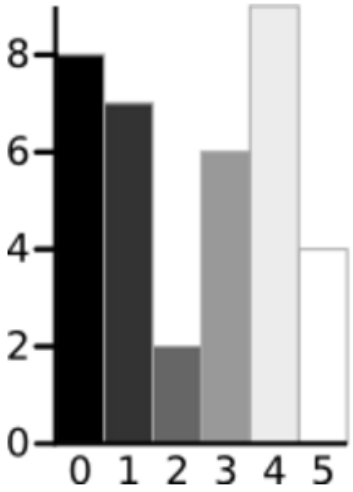
$$\mu_1(t) = \sum_{i=t}^{L-1} i\frac{p(i)}{\omega_1}$$

$$\mu_T = \sum_{i=0}^{L-1} ip(i)$$

Another mathematical thing involved in this process is the class mean, represented by u0, u1 and ut, and their formulas are listed on the left. (8)

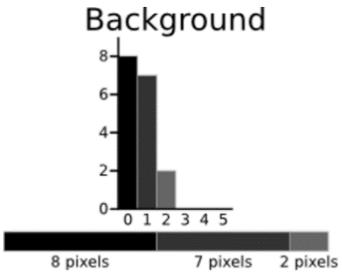
To better understand how this works, a simple 6*6 image on the right will help to demonstrate it. The figure below counts the number of the pixel with each brightness. (9)

(9)



(9)

(9)



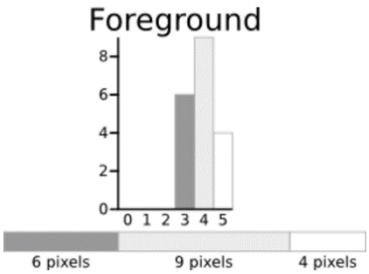
$$\text{Weight} \quad W_b = \frac{8+7+2}{36} = 0.4722$$

$$\text{Mean} \quad \mu_b = \frac{(0\times 8)+(1\times 7)+(2\times 2)}{17} = 0.6471$$

$$\text{Variance} \quad \sigma_b^2 = \frac{((0-0.6471)^2\times 8)+((1-0.6471)^2\times 7)+((2-0.6471)^2\times 2)}{17}$$

$$= \frac{(0.4187\times 8)+(0.1246\times 7)+(1.8304\times 2)}{17}$$

$$= 0.4637$$



$$\text{Weight} \quad W_f = \frac{6+9+4}{36} = 0.5278$$

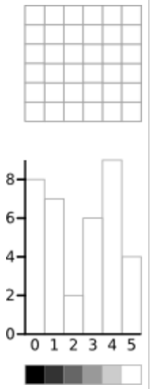
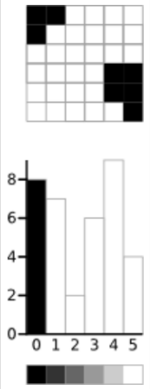
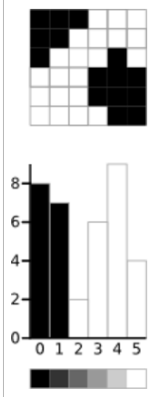
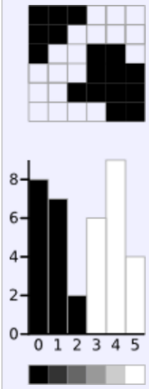
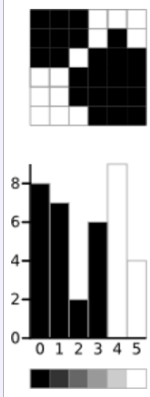
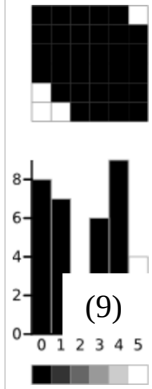
$$\text{Mean} \quad \mu_f = \frac{(3\times 6)+(4\times 9)+(5\times 4)}{19} = 3.8947$$

$$\text{Variance} \quad \sigma_f^2 = \frac{((3-3.8947)^2\times 6)+((4-3.8947)^2\times 9)+((5-3.8947)^2\times 4}{19}$$

$$= \frac{(4.8033\times 6)+(0.0997\times 9)+(4.8864\times 4)}{19}$$

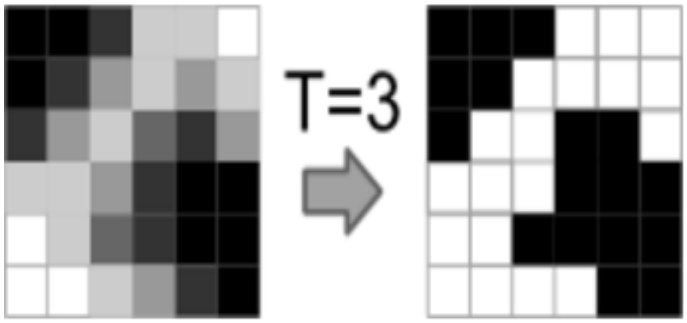
$$= 0.5152$$

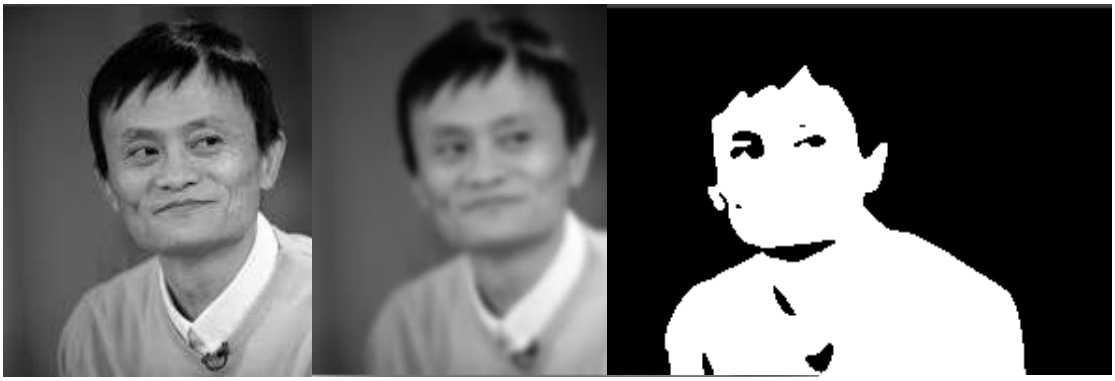
The figure above shows how the variance of the background and the foreground (object) are calculated. By doing calculation for T (threshold value) equals from 0 to 5, computers can determine the T minimum variance. (9)

Threshold	T=0	T=1	T=2	T=3	T=4	T=5
						
Weight, Background	$W_b = 0$	$W_b = 0.222$	$W_b = 0.4167$	$W_b = 0.4722$	$W_b = 0.6389$	$W_b = 0.8889$
Mean, Background	$M_b = 0$	$M_b = 0$	$M_b = 0.4667$	$M_b = 0.6471$	$M_b = 1.2609$	$M_b = 2.0313$
Variance, Background	$\sigma^2_b = 0$	$\sigma^2_b = 0$	$\sigma^2_b = 0.2489$	$\sigma^2_b = 0.4637$	$\sigma^2_b = 1.4102$	$\sigma^2_b = 2.5303$
Weight, Foreground	$W_f = 1$	$W_f = 0.7778$	$W_f = 0.5833$	$W_f = 0.5278$	$W_f = 0.3611$	$W_f = 0.1111$
Mean, Foreground	$M_f = 2.3611$	$M_f = 3.0357$	$M_f = 3.7143$	$M_f = 3.8947$	$M_f = 4.3077$	$M_f = 5.000$
Variance, Foreground	$\sigma^2_f = 3.1196$	$\sigma^2_f = 1.9639$	$\sigma^2_f = 0.7755$	$\sigma^2_f = 0.5152$	$\sigma^2_f = 0.2130$	$\sigma^2_f = 0$
Within Class Variance	$\sigma^2_W = 3.1196$	$\sigma^2_W = 1.5268$	$\sigma^2_W = 0.5561$	$\sigma^2_W = 0.4909$	$\sigma^2_W = 0.9779$	$\sigma^2_W = 2.2491$

(9)

So it is easy to tell that the within class (collection) variance is the smallest when T = 3, so the threshold value is determined as T = 3, but not other values. Thus, what under 3 will be the background, and what greater or equal to it will be the object needed. This kind of thresholding is widely used in face detection projects.(9)





So much for explaining thresholding, let us look one real case. As shown above, by blurring pictures and then use the technique of threshold, the eyes on the face become dark and the other parts of the faces become white (the color can be switched). Then, it will be much easier for computers to determine where the eye lies because the whole purpose of doing this so much work is for “down sampling” the whole image from with many grayscale values to only with black and white color. Then the computer will be able to try and find a region where the black area set lies above the white area, and then each part of the black area set lies almost symmetrically across a white area.

Conclusion

Face detection done by computers is not an easy work, but with the process of blurring and thresholding, this is not a hard work anymore. What programmers need is just to build proper classifier and let the computers learn themselves by inputting them millions of faces. The purpose for using blurring and thresholding may applies to many other area, because it gives the idea that we should simplify the difficulty we are facing. With the advanced technology of face detection, face recognition will become easier and easier in the future, and maybe in one day, we will be able to do what the “Avengers” “did”, to find Loki by face recognition on that flying ship!

Bibliography

Alexander Mordvintsev & Abid K. Revision. 2013. Face Detection using Haar Cascades. OpenCV-Python Tutorials. July 12th 2017. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html#face-detection (1)

Alexander Mordvintsev & Abid K. Revision. 2013. Smoothing Images. OpenCV-Python Tutorials. July 12th 2017. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html#filtering (2)

OpenCV dev Team. (Keep updating). Making your own linear filters. OpenCV 2.4.13.2 Documentation. July 12th 2017. http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/filter_2d/filter_2d.html (3)

OpenCV Tutorials. Smoothing Images. OpenCV 3.3.0-rc. July 13th 2017. http://docs.opencv.org/master/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html (4)

Wikimedia Foundation. Gaussian Blur. Wikipedia. July 13th 2017.

https://en.wikipedia.org/wiki/Gaussian_blur#cite_note-NixonAguado-2

(5)

阮一峰 Frank, Nov/14/2012. 高斯模糊的算法 (The algorithm of Gaussian Blur). July 12th 2017.

http://www.ruanyifeng.com/blog/2012/11/gaussian_blur.html (6)

Alexander Mordvintsev & Abid K. Revision. 2013. Image Thresholding. OpenCV-Python Tutorials. July 12th 2017. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#thresholding (7)

Wikimedia Foundation. Otsu's method. Wikipedia. July 13th 2017.

https://en.wikipedia.org/wiki/Otsu%27s_method (8)

Dr. Andrew Greensted. (Keep updating). Otsu Thresholding. The Lab Book Pages. July 12th 2017

http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html (9)

Note: the picture, which is used to demonstrate how blur works, is one common joke picture in China, and the picture of Jack Ma, comes from Bing Image. And many conclusion comes from experiences.