Saliency Detection Report

Daquan Lin 85610653
ShanghaiTech University
Shanghai, China
lindq@shanghaitech.edu.cn

Abstract

This report introduces two approaches to get a saliency map of image, they are Context-Aware Saliency Detection[?] and Spectral Residual[?]. For Context-Aware Saliency Detection, I used two methods , parallel programming and naive way, calculating all the pixels of a image at once and pixel by pixel, respectively. For Spectral Residua Approach, I utilies the OpenCV[?] library, it is very simple through some existance APIs. I main to talk about how to implement them, and some problems I face and how to fix them. Most improtant, I will discuss how to optimize the running time for each image. Finally, I test them with some images. They shown in ContextAwareSaliency-Detection.ipynb and SpectralResidualSaliency.ipynb.

1. Introduction

Saliency detection aims to locate the important and informative regions in an image, which plays an important role in further image or video analysis and has been widely used in many applications, such as object recognition, image segmentation, visual tracking, and so on. In the field of saliency detection, most existing methods can be categorized into bottom-up and top-down, and combination of bottom-up and top down methods. Top-down models usually require supervised learning that based on the manually labeled ground truth. On the contrary, bottom-up models exploit low-level cues to gain saliency maps and are typically unsupervised. Both of them, I discuss in this report, are bottom-up models.

In section 2, I will discuss how to implement Context-Aware model and some problem I faced and how to fix them. Then, I will compare the running times of naive way and vectorize way, and discuss some other speed up method. In section 3, I will talk about how to implement Spectral Residual approach.

2. Context-Aware Model

2.1. Local-Global Single-Scale Saliency

For each pixel in a image, let $d_{color}(P_i, p_j)$ be the euclidean distance between the vectorized patches p_i and p_j in CIE $L*a*b$ color space, normalized to the range $[0, 1]$. Pixel i is considered salient when $d_{color}(P_i, p_j)$ is high $\forall j$. In my implementation, patch size is $7 \times 7 \times 3$ as same as the original paper.

The K in there is also equal to 64 as large as the original paper. In there, I use KD Tree instead of sort all the neightbour patches. Fortunately, the API: NumPy.argpartition in NumPy package is used KD Tree. So, it is very simple to get the top K nearest patches.

the positional distance between patches is also an important factor.Background patches are likely to have many similar patches both near and far-away in the image. This is in contrast to salient patches which tend to be grouped together. This implies that a patch p_i is salient when the patches similar to it are nearby, and it is less salient when the resembling patches are far away.

Let $d_{position}(p_i, p_k)$ be the euclidean distance between the positions of patches p_i and q_k, normalized by the larger image dimension. Based on the observations above, we define a dissimilarity measure between a pair of patches as

$$d(p_i, q_k) = \frac{d_{color}(p_i, q_k)}{1 + c\dot{d}_{postion}(p_i, q_k)} \tag{1}$$

where $c = 3$ the same as the original paper.

As mentioned above, a pixel i is salient when $d(p_i, q_k)$ is high $\forall k \in [1, K]$. Hence, the single-scale saliency value of pixel i at scale r is defined as

$$S_i^T = 1 - exp\{-\frac{1}{K}\sum_{k=1}^{K} d(p_i^T, q_k^T)\} \tag{2}$$

2.2. Including the Immediate Context

I implement the visual contextual effect. First, the most attended localized areas at each scale are extracted from the saliency maps produced by (3). A pixel is considered attended at scale r if its saliency value exceeds a certain threshold ($S_i^T > 0.8$).

Then, each pixel outside the attended areas is weighted according to its eucliean distance to the cloest attended pixel. Let $d_{foci}^T(i)$ be the euclidean positional distance between pixel i and the closest focus of attention pixel at scale r, normalized to the range$[0, 1]$. The saliency of pixel i is redefined as

$$\hat{S}_i = \frac{1}{M} \sum_{r \in R} S_i^T (1 - d_{foci}^T(i)) \tag{3}$$

2.3. Multiscale Saliency Enhancement

For each image, I scale it to no more than the size of 250 pixels(largest dimension). Then, I use four scales : $R = \{100\%, 80\%, 50\%, 30\%\}$. The saliency map S_i^T at each scale is normalized to the range $[0, 1]$ and interpolated back to original image size.

Furthermore, we have

$$\bar{S}_i = \frac{1}{M} \sum_{r \in R} \hat{S}_i^T \tag{4}$$

2.4. Center Proir

I generate a 2D Gaussian at the center of image. But, when I use variance as the original paper, the Gaussian gray image seems just a very small light in center. However, when I use their variance as my standard deviation, it looks much like the image as the paper.

Final saliency of a pixel is defined as

$$S_i = \hat{S}_i G_i \tag{5}$$

where G_i is the value of pixel i in the map G.

2.5. Problems

1) When pad a image, I choose the replicate way in intuition.

2) Coming to how find the top K nearest patches. Because, I used KD Tree to get the Kth value, then get all the indices of values that less than it. When many patches have same value, it will be difficult to decide the length of indices. So, in there, I added a small random perturbation to the value of each patch to avoid the same value. Especially, when I dealed with bird.jpg, it has a large area of black, then a lot of patches are similar with each other.

2.6. Speed Up

First, I implemented a navie way, it used double loops(width and height), and seems very slow. My workstation has $XeonE5 - 2620v4 \times 2$, each one has 8 cores and 16 threads. But, when I test this program, it just take up a single thread. I guess it could be double loops prevent program from paralleling. Then, I write a vectorize code. Unfortuately, it slower than naive way. Originally thought that my workstation maybe lack some Linear Algebra Software, but it is existing. After it, I do a small test: let two 10000×10000 matrices A, B, operate dot product. This operation taken up all threads immediately.

Considering the whole process of Context-Aware method, we can find it doesn't have some complex computation. Therefore, I guess most of APIs in NumPy which I took can't be speeded up by Basic Linear Algebra Subprograms(BLAS) library.

Also, we can try other method to speed it up, such as NumExpr or GPU etc.

3. Spectral Residual Model

First, read a gray image from the original image and resize the gray image(width:128), keep the ratio of width than height. Then, operate the Fourier Transform in it, get the general shape of log spectra:$A(f)$ and phase spectrum $P(f)$. After that, get specral residual $R(f)$ by the rule below:

$$R(f) = log(A(f)) - h_n(f) * log(A(f)) \tag{6}$$

where h_n is $n \times n$ convolution kernel, in there $n = 3$. Finally, operate inverse Fourier Treansform, and add a normalization, we can get the saliency map.

Because I am a categorical newbie toward OpenCV, I reference uoip's work. Very simple implementation.

4. Conclusion

After implement two methods to get saliency map from a image, I find the first method is better than the second method, but cost more time. I think the second method is similar to SVM, we find a suitable kernel function and project the input data to high dimension space, and find the hypreplane to divide them. The second method may also useful to defend the attack to CNN models.

References

[1] A. Alpher. Frobnication. Journal of Foo, 12(1):234–778, 2002.

[2] A. Alpher and J. P. N. Fotheringham-Smythe. Frobnication revisited. Journal of Foo, 13(1):234–778, 2003.

[3] A. Alpher, J. P. N. Fotheringham-Smythe, and G. Gamow. Can a machine frobnicate? Journal of Foo, 14(1):234–778, 2004.

[4] Authors. The frobnicatable foo filter, 2014. Face and Gesture submission ID 324. Supplied as additional material fg324.pdf.

[5] Authors. Frobnication tutorial, 2014. Supplied as additional material tr.pdf.