Contour Detection

Project Final Report

Haiyan Yang, Shuo Han, Yuting Liu

Project webpage available at: https://fishball1121.github.io/Contour-Detection/

Abstract

Contours are the most informative parts of an image. With contours, we can recognize objects and separate different regions. Contour detection , as a fundamental process in different fields of computer vision, significantly reduces the amount of data and filters out useless information, while preserving the important structural properties in an image. To have a better understanding of contour detection, we systematically studied three advanced papers in this area and implemented them. We then compared these three methods in different aspects and presented an interesting application developed based on contour detection results.

1 Introduction

Contour detection of objects is essential in human vision. With damages in certain brain area responsible for the perception of contours, a person will be unable to recognize objects [1]. To develop a more powerful computer vision system, contour detection is also a must. In fact, contour detection is one of the most fundamental problems in computer vision. People can have a deep knowledge of the structural information in the images based on object contours, which many other tasks such as image segmentation, object recognition and scene understanding rely heavily on. Due to the importance of contour detection, among these years a lot of related work has been done and many methods have been developed to achieve better results.

Traditional methods such as Sobel and Canny worked just fine in edge detection tasks, but for contour detection they preserved too many details that are not important to us. Therefore in this project we investigated three new methods that can generate contour information in a more human understandable way.

This report is organized as follows. First we will briefly review some previous relevant work conducted on this area, and then we will delve deep into three advanced methods, probability of boundary (Pb),

sparse code gradient (SCG) and sketch tokens, which we chose for our detailed study. After that we will show the results of our implementation of these methods, with the comparison of the performances and an interesting application. Finally we will make a conclusion along with some improvement work that we can work on in the future.

2 Related Work

Early methods of contour detection focused on finding sharp discontinuities in brightness, such as Canny edge detector [2]. In contrast, recent methods tried to estimate the probability whether a pixel is on the boundary using information more than brightness.

A number of recent methods with satisfactory performance are based on the posterior probability of boundary (Pb) developed Martin et al. [3], which uses brightness, color and textures to detect both global and local contours. These techniques include the conditional random fields model developed by Ren et al. [4], the untangling cycle algorithm designed by Zhu et al. [5] and the idea of globalized probability boundary (gPb) raised by Maire et al. [6]. Other progressive work not based on Pb includes the min-cover approach developed by Felzenszwalb and McAllester [7], the boosted edge learning algorithm invented by Dollar et al. [8], the method using sparse code gradients (SCG) proposed by Ren and Bo [9] and sketch tokens designed by Lim et al. [10].

3 Our Approaches

We chose three methods recently developed in contour detection: probability of boundary (Pb), sparse code gradient (SCG), and sketch tokens. Pb was chosen since a number of recent methods of contour detection were based on it that combines different image information for more accurate classification. SCG is a method based on the technology of Pb and improved the performance by the dictionary learning algorithm. Sketch tokens has its own novelty for learning contour representations from mid-level human drawn sketches.

3.1 Probability of Boundary (Pb)

Early approaches of detecting contours aimed at finding sharp discontinuities in the brightness channel. But brightness alone is not enough. Recent methods of contour detection also included color and texture information to locate the correct boundaries. How to represent the changes in brightness, color and texture and how to combine them to predict the probability of boundaries are the problems which Pb focused on. Gradient operators for brightness, color and texture are introduced and some learning techniques are used to combine the gradients.

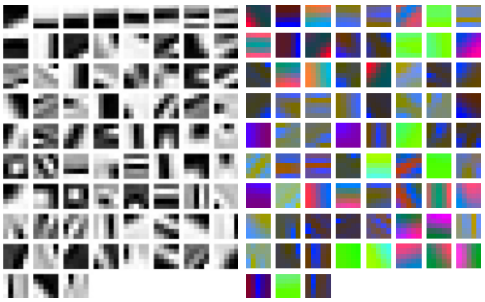
As for gradient, it is computed in the following way: draw a circle around a pixel point (x, y) and divide it two parts by a diameter with an angle θ, and then compute the histograms for the two halves, denoted as g and h and the gradient $G(x, y, \theta)$ is the chi-square difference $\chi = \frac{1}{2}\sum_i \frac{g(i) + h(i)}{g(i) - h(i)}$

between the two halves. This can be obtained by convolving the image with a half disk mask. Brightness and color are familiar concepts for us, but we still need to know how to represent texture. Textures are some patterns represented by clustering the responses with some filters. The responses are achieved by convolving the image with a collection of orientated derivative Gaussian filters and k-means algorithm is used to cluster the pixels.

After computing the gradients for brightness, color and texture, we can treat the combination problem as a supervised learning problem and the following classifiers can be used: density estimation, classification trees, logistic regression, hierarchical mixtures of experts and support vector machines. Moreover, the local cues can be combined with global cues called gPb method.

3.2 Sparse Code Gradient (SCG)

Sparse coding gradient is a method based on the Probability of Boundary combining K-SVD dictionary learning and orthogonal matching pursuit (OMP) to extract sparse code at every pixel of different channels. Classification of boundary is improved by sparse code learning on a given training set to enable pixel representation and aggregation of pixel information in local neighborhoods. The learning process will be separately applied to different channels on the training set including: grayscale, chromaticity and surface normal. The learned dictionaries are visualized in Figure 1.



(a) Grayscale (b) Chromaticity

Figure 1: K-SVD dictionaries learned for different channels: grayscale and chromaticity.

K-SVD generalizes k-means and learns dictionaries of codewords from unsupervised data. Given a set of image patches $Y = [y_1, ..., y_n]$, K-SVD jointly finds a dictionary $D = [d_1, ..., d_m]$ and an associated sparse code matrix $X = [x_1, ..., x_n]$ by minimizing the reconstruction error:

$$\min_{a,b} \|Y - DX\|_F^2 \quad \text{s.t. } \forall i, \|x_i\|_0 \leq K; \forall j, \|d_j\|_2 = 1$$

where $||\cdot||_F$ denotes the Frobenius norm, $||\cdot||_0$ counts the non-zero entries in the sparse code x_i, and K is the predefined sparsity level (number of non-zero entries).

Once we got the learned dictionary, OMP is implemented to compute pixel-level sparse codes based on the choice of 5*5 image patches with dictionary size of 75. Here are an example of learned dictionary with k-SVD and OMP by a given image in gray-level scale channel and Lab channel.

With results of aggregate pixel information, our contour detection is based on oriented half-discs used

in gPb. Two half-discs N^a and N^b of size $s \times (2s+1)$ for each of 8 orientations at each pixel p and scale s are defined for the contour classification. The final representation of contrast at a pixel p is the concatenation of sparse codes pooled at all scales $s \in \{1, ..., S\}$ ($S = 2$ or $S = 4$ is commonly used):

$$D_p = [D(N_1^a, N_1^b), ..., D(N_S^a, N_S^b); F(N_1^a \cup N_1^b), ..., F(N_S^a \cup N_S^b)]$$

Here the union term $F(N_1^a \cup N_1^b)$ denotes the whole appearance of the whole disk (including two half discs) and is normalized by $\|F(N_s^a)\| + \|F(N_s^b)\| + \varepsilon$.

We can set a threshold α to k linear classifier we learned for each orientation in boundary detection. With k linear SVMs, we can implement the algorithm of Pb introduced above to conduct the stage of supervised learning.

3.3 Sketch Tokens

Sketch tokens, as its name suggests, refers to a set of token classes that represent local edge structures such as straight lines, corners, parallel lines, etc. in images. They are unsupervisedly learned using mid-level information in the form of hand drawn sketches in images. Technically speaking, sketch tokens are obtained by clustering patches that are extracted from binary sketch images. Patches with a fixed size are extracted only if their center pixels indicate a contour. Clustering is done on Daisy descriptors computed over patches using k-means algorithm. Some example of sketch tokens are shown in Figure 2.

Figure 2: Examples of sketch tokens.

To detect sketch tokens in new images, a random forest classifier, which is efficient for multi-class classification problem, is trained to detect their occurrence. Features of the classifier are computed from patches into two types: channels and self-similarities.

Channels are composed of color, gradient and oriented gradient information in a patch. Three color channels are computed using the CIE-LUV color space. Three gradient magnitude channels are computed with different Gaussian blurs ($\sigma = 0, 1.5, 5$) and for the first two the gradient magnitude channels are split to create four additional channels each for a total of eight oriented magnitude channels. All channels are Gaussian blurred with $\sigma = 1$ and all the values in the resulting channels compose the first type of the feature.

Self-similarities are computed on a $m \times m$ grid over the patch in each channel in the first type. For example, if we pick $m = 5$ for a 35×35 patch, it yields 7×7 cells. For channel k and grid cells i and j, the self-similarity feature f_{ijk} is defined as $f_{ijk} = s_{jk} - s_{ik}$ where s_{jk} is the sum of grid cell j in channel k. The total number of such features is given by $\binom{m \cdot m}{2}$ due to symmetry.

Total class number equals to the number of sketch token classes from the clustering result plus 1,

which indicates the patch locates at a background (non-contour) pixel. The random forest classifier uses the features and class labels to predict the probability of each class.

After training the classifier, the probability of a contour at the center pixel of a patch in the test image is calculated by $1 - P(\text{background})$, that is to see if the patch belongs to any classes of sketch tokens. Once the probabilities of contour pixels are computed throughout the whole image, a standard non-maximal suppression scheme is applied to find the peak response of a contour.

4 Experiments

4.1 Benchmark

To evaluate contour detection algorithms in a quantitative manner, we will be using a dataset and its corresponding benchmark called Berkeley Segmentation Data Set and Benchmarks 500 (BSDS 500) [6]. It consists of natural images and their corresponding human-annotated ground truth contours. The evaluation of each algorithm is based on the F-score, defined as $F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$.

4.2 Implementation

For Pb, we first generate the texture map of the image using $k = 64$. Then we compute the gradients at 8 different orientations in each channel of Lab color space and the texture channel. Finally, we combine the gradients from 4 channels with the recommended weights and choose the largest value among all orientations to be the probability of boundary.

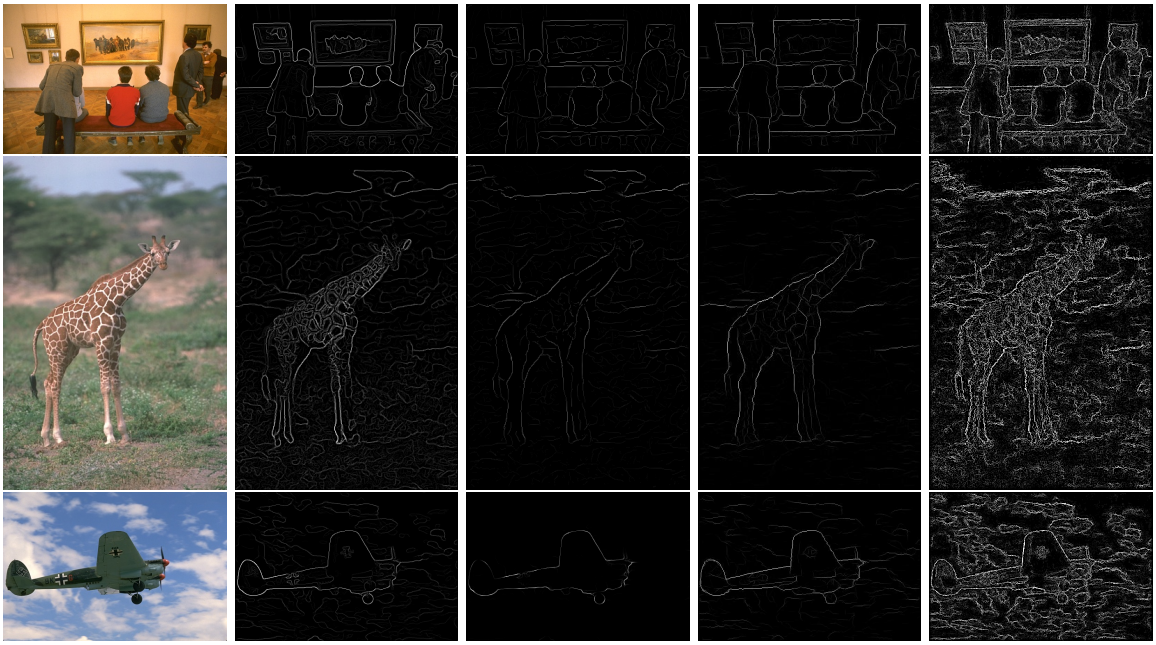
For SCG, the learning process combining K-SVD and OMP has finished with benchmark of BSDS 500 based on the gPb algorithm. Multi-scale neighborhoods coding is implemented for detection of contours with scale $S = 4$. The combination results of contour detection will be compared between gPb and SCG to see how performances are improved with dictionary learning.

For sketch tokens, due to the limitation of our computing power, in our implementation we reduce patch size to 21×21 and number of clusters to $k = 20$. Furthermore we used less training examples ($\sim$80k) to train the random forest classifier. Our implementation yields an F-score of 0.70, compared to 0.73 in the original paper.

4.3 Results

Comparison of the contour detection results among various methods is shown in Figure 3. Note that comparing to traditional methods (Sobel and Canny), recent methods capture more contour information in human sense.

The F-score and the precision-recall curve of the comparison among various methods are shown in Table 1 and Figure 4. Note that our implementations beat several baseline methods like Sobel and



(a) Original (b) Canny (c) Pb (d) SCG (e) Sketch tokens

Figure 3: Comparison of different contour detection methods applied on several example images.

Canny but get outperformed by human vision. However due to the limitation of our computing power we cannot reproduce the results as good as in the original paper. The performance of SCG is better than Pb and this can be explained by the fact that SCG aggregates information about contours in a neighbourhood and it only shows salient contours. Sketch tokens learns much more contour details than any other methods because it directly learns from hand drawn sketch images.

	Canny	Pb	SCG	Sketch tokens
F-score	0.60	0.67	0.72	0.70

Table 1: F-score of early methods like Canny and our implementations of the three methods.

4.4 Improvement

For Pb, it took quite a while when we computed the texture map using k-means. This could be improved by calculating a general texture map by clustering responses from a set of training images. With the general texture map, we can consider the index of a pixel as the index of the nearest response in the general texture map. In this way, the runtime of Pb can decrease from a minute to about 20 seconds and the result is almost the same.

For SCG, we changed the size of image patches when learning sparse code matrix, which seems that the accuracy of classification is not improved but the runtime is increased a lot. Performances would also change with different channels of the same image. The dictionary learning via grayscale channel would perform a little worse than dictionary learning via color channel, which shows that the information of color channel would help to correctly classify the performance.

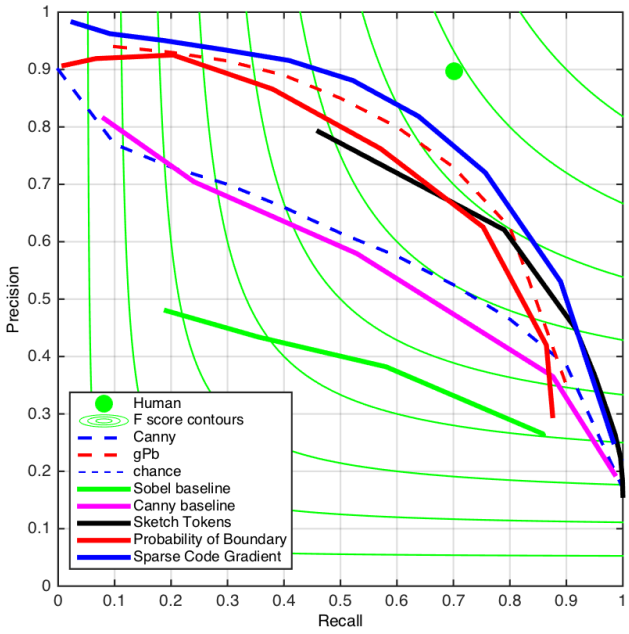


Figure 4: The precision-recall curve.

For sketch tokens, we tried to change the parameters of the model with various patch sizes, different sketch token class numbers and additional features such as RGB color channels. However the performance didn't seem to improve a lot.

We implement our edge detection on a very interesting cartoonifying application. The process of cartoonifying an image can be illustrated as follows: 1)Smooth the image with certain smooth filter. 2)Implement the edge detector on the image to edge detection. 3) Classify the strong edge in edge image and fill the corresponding pixel in smoothed image to 1.

Example of an image by our implementation of cartoonifying an image is illustrated in Figure 5. Firstly the input image is Figure 5(a). With bilateral smooth filter and one of our edge detectors, we could get the final output image.

After that we can compare the performance with out different edge detections given the above results. From Figure 6 we can see that the sketch token presents more details about the image which first two images ignore. It seems the first image provides cleaner output than the other two.

5 Conclusion

In our project, we had a comprehensive study on three of the most advanced contour detection methods, probability of boundary, sparse code gradient and sketch tokens. We implemented them and tried to make our own improvements. Although we didn't improve the final results a lot, we did find an interesting application of using these methods to make cartoonified images, which can be

(a) Input image (b) Smoothed image (c) Edge image (d) Output image

Figure 5: The process of our cartoonifiying application.

(a) Pb (b) SCG (c) Sketch tokens

Figure 6: Cartoonifiying using different contour detection methods.

considered as the novelty of this project.

Although the accuracy of contour detection can be improved by recent methods, the runtime is still pretty long compared with traditional methods like Sobel and Canny. Thus for our future work, a possible approach is to reduce the runtime of recent method by modifying the way to measure changes.

In addition, instead of using oriented gradients, we may consider use derivatives as Canny. Another promising direction is to reduce the details captured by Canny, which may be achieved by SCG.

References

[1] Giuseppe Papari and Nicolai Petkov. "Edge and line oriented contour detection: State of the art". In: *Image and Vision Computing* 29.2 (2011), pp. 79–103.

[2] John Canny. "A computational approach to edge detection". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6 (1986), pp. 679–698.

[3] David R Martin, Charless C Fowlkes, and Jitendra Malik. "Learning to detect natural image boundaries using local brightness, color, and texture cues". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.5 (2004), pp. 530–549.

[4] Xiaofeng Ren, Charless C Fowlkes, and Jitendra Malik. "Scale-invariant contour completion using conditional random fields". In: *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*. Vol. 2. IEEE. 2005, pp. 1214–1221.

[5] Qihui Zhu, Gang Song, and Jianbo Shi. "Untangling cycles for contour grouping". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE. 2007, pp. 1–8.

[6] Pablo Arbelaez, Michael Maire, Charless Fowlkes, et al. "Contour detection and hierarchical image segmentation". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.5 (2011), pp. 898–916.

[7] Pedro Felzenszwalb and David McAllester. "A min-cover approach for finding salient curves". In: *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*. IEEE. 2006, pp. 185–185.

[8] Piotr Dollar, Zhuowen Tu, and Serge Belongie. "Supervised learning of edges and object boundaries". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 2. IEEE. 2006, pp. 1964–1971.

[9] Xiaofeng Ren and Liefeng Bo. "Discriminatively trained sparse code gradients for contour detection". In: *Advances in neural information processing systems*. 2012, pp. 584–592.

[10] Joseph Lim, C Zitnick, and Piotr Dollár. "Sketch tokens: A learned mid-level representation for contour and object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 3158–3165.