Edge Detection Using Convolutional Neural Network

Ruohui Wang(✉)

Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong, China
wr013@ie.cuhk.edu.hk

Abstract. In this work, we propose a deep learning method to solve the edge detection problem in image processing area. Existing methods usually rely heavily on computing multiple image features, which makes the whole system complex and computationally expensive. We train Convolutional Neural Networks (CNN) that can make predictions for edges directly from image patches. By adopting such networks, our system is free from additional feature extraction procedures, simple and efficient without losing its detection performance. We also perform experiments on various networks structures, data combination, pre-processing and post-processing techniques, revealing their influence on performance.

Keywords: Deep learning · Convolutional neural network · Image processing · Computer vision · Edge detection

1 Introduction

Edge detection is the task of identifying object boundaries within a still image (see Fig. 1). As a fundamental technique, it has been widely used in image processing and computer vision areas [1–5]. Accurate, simple and fast edge detection algorithms can certainly increase both performance and efficiency of the whole image processing system. However, edges form in diverse ways. Finding a universally applicable detection rule is hence not easy.

Conventional edge detection algorithms rely heavily on gradient computing [1]. Pixels with large gradient magnitude are labeled as edges. Other techniques, such as non-maximum suppression [6], are usually combined to yield a better result. These methods are all based on the assumption that color or intensity changes sharply on the boundary between different objects while it remains unchanged within one object. Unfortunately, this is not always true. Large color gradient can appear on texture within one object while small color gradient can also appear on object boundaries.

Having realized the limitation of local gradient cues, recent works start to introduce learning techniques when designing edge detection algorithms [7–10]. Correspondence between object boundaries and image are learned from data instead of based on man-made assumptions. However, traditional learning methods are usually not powerful enough to learn a direct mapping from image

L. Cheng et al. (Eds.): ISNN 2016, LNCS 9719, pp. 12–20, 2016.
DOI: 10.1007/978-3-319-40663-3_2

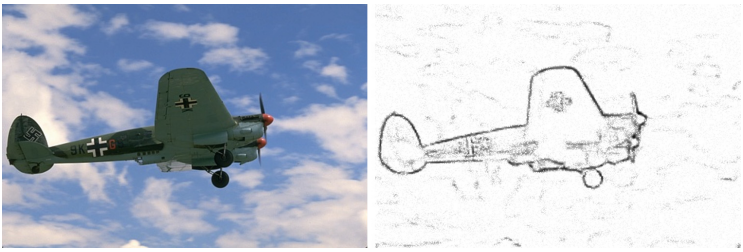

Fig. 1. An example of edge detection.

patches to edge predictions. People have to compute multiple color and gradient channels or extract self-similarity features in order to get a rich representation of the original image. As a result, such systems usually consist of multiple modules, which could be complex and inefficient. In the meantime, selecting proper features usually requires domain knowledge and can affect both efficiency and performance a lot.

Faced with such problems, we start to consider introducing powerful deep learning techniques, such as convolutional neural network (CNN) [11], into the edge detection problem. Unlike traditional shallow learning structures, deep neural networks can learn hierarchical feature representations in their multiple-layer structure. By adopting CNN, our edge detection system is free from extra feature extraction or multiple channel computation, thus becomes straightforward and efficient.

On the other hand, CNN tends to capture local patterns from images in its convolutional layers. This property also makes it a suitable tool for solving the edge detection problem, because edges are usually locally correlated and exhibit specific patterns, such as straight lines, corners, T-junctions and Y-junctions.

Motivated by these intuitions, we design a simple and efficient edge detection system with CNN being adopted as the central part. In order to further simplify the computation, we removed pooling layers in our networks. In order to select an optimal network structure, we performed a lot of experiments on the popular BSDS500 [1] dataset, comparing different network structures as well as data combination, preprocessing and post-processing techniques (see Sects. 2 and 3 for detail). The best performance is achieved on a simple three-layer network taking raw RGB color image patches as input without any preprocessing. By adding non-maximum suppression to the whole system, the performance can be further improved a little.

The rest of this paper is arranged as follow. In Sect. 2, we give an overview of our CNN edge detector as well as some preprocessing and post-processing techniques. In Sect. 3, we exploit different structures and data construction in our experiments, and obtain insightful observations. Section 4 concludes the paper.

2 Edge Detection System

Figure 2 provides an overview of our edge detection system. Given an image, we can first apply some preprocessing techniques [12] for noise removal. Then a convolutional neural network scans over the entire image, making edge prediction for every pixel based on the image patch centered on it. At last, non-maximal suppression [6] or morphological operations can be further applied as a post-processing step to thin the output edge map so as to increase localization accuracy.

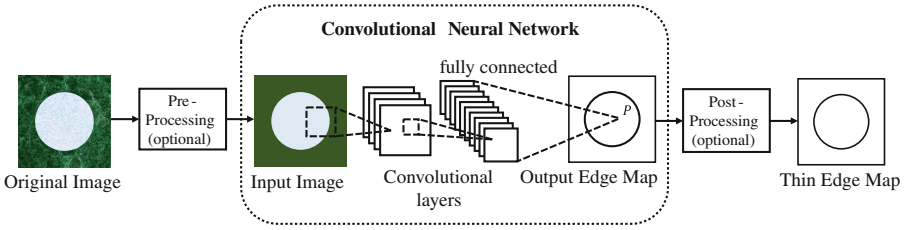

Fig. 2. An overview of our edge detection system.

2.1 Preprocessing

Natural images are sometimes degraded by noise. It is straightforward to apply some noise removal algorithm first. Here, we choose a slight and smart algorithm [12], which increases system complexity little.

2.2 Convolutional Neural Network

A Convolutional Neural Network works as the core component of our system. It takes image patches as input and makes predictions on whether their central pixels locate on an edge or not. Any network that fulfills this task can be adopted in our system, but choosing the best network structure is not straight-forward. We performed a thorough comparison on different network structures and the best performance was achieved on a three-layered network whose structure and parameters are summarized in Table 1.

Table 1. Network structure and parameters adopted in our system.

#	Input size	Layer type	Filter size	Stride	Nonlinearity	Output size
1	23*23*3	convolutional	5*5*3	3	ReLU	7*7*32
2	7*7*32	convolutional	3*3*32	2	ReLU	3*3*256
3	3*3*256	fully connected	N/A	N/A	Logistic	1

2.3 Post-Processing

As a common phenomenon, edges produced by detection algorithms usually cover several pixels and are considered to be inaccurate [6,9,10]. In this regard, non-maximal suppression [6] or morphological operation can be adopted to serve as a post-precessing step, rendering a thinner edge map in the final output.

3 Experiments and Results

3.1 Data Set and Evaluation Method

We selected the most popular BSDS500 data set [1,13] for training and evaluating our edge detector. This dataset contains 500 natural images. For each image, several people were asked to draw a contour map separating different objects based on their own understanding. All 500 images are divided into 3 subsets, with 200 for training, 100 for validation and 200 for testing. We strictly followed the official guidelines [13] to train and tune our system exclusively on the train and validation subsets, and to evaluate our results on the entire test subset with provided benchmarking code [13].

3.2 Preparing Training Data

Before training our network, we need to prepare image patches and corresponding ground truth that can be acceptable by our neural networks. The procedure is summarized in Fig. 3.

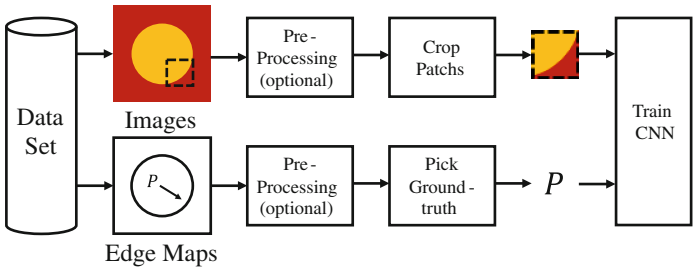

Fig. 3. Work flow on preparing training data.

First, we could apply some preprocessing techniques [12] to remove noise from the original images. For each image in BSDS500, there are multiple corresponding edge maps. In order to determine a single ground truth, we selected the most sparse one or averaging the most spare two or three. Then we cropped images into patches. For each image patch, we located it on the edge map and picked the value at its center as corresponding ground truth label. At last, these patch-label pairs were sent for training the neural network.

One key problem is that there are large numbers of negative samples, i.e. patches whose center do not belong to any edge, compared to positive samples (about 50:1). In order to balance the ratio between positive and negative samples, we selected all positive samples and a small random partition of negative samples for training the network, resulting the negative/positive ratio near 2:1.

3.3 Training Neural Networks

We used the cuda-convnet toolbox [14] to train our CNN on an NVIDIA Tesla K40c GPU. The training process would last about 40 min for a three-layer network, or nearly one day for a complex network. We always try to tune the network parameters, such as number of filters and filters' size, to avoid either underfitting or overfitting. We stopped the training process if the error curve converge or the error rate on the validation set achieve a minimum. Ordinary error curves during the training steps are provided in Fig. 4.

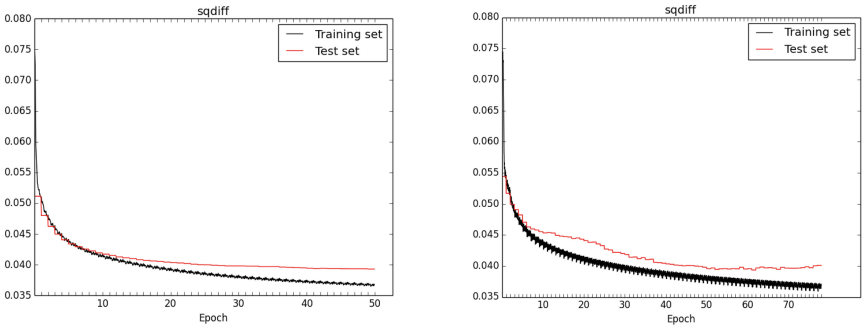

Fig. 4. Ordinary error curves on training and validation sets.

3.4 Result

We evaluated our CNN edge detector using the original benchmark code provided along with BSDS500 [13]. We tested on several networks with different structures and image color channel combinations. The best performance was achieved by a three-layered network taking raw RGB image patches as input. It is summarized in Fig. 5 and Table 2 with comparison to other mainstream methods, including the most widely used non-learning-based Canny edge detector [6] and several learning-based algorithms [1,8,9,15]. Figure 6 shows some sample results of our detector that of sketch tokens [9]. As shown in the comparison, our edge detector is fast while achieving comparable performance with state-of-the-art methods. Since our neural network contains only convolutional and fully connected layers, algorithms such as [16] can also be adopted in our system to further speed it up by several times or even tens of times.

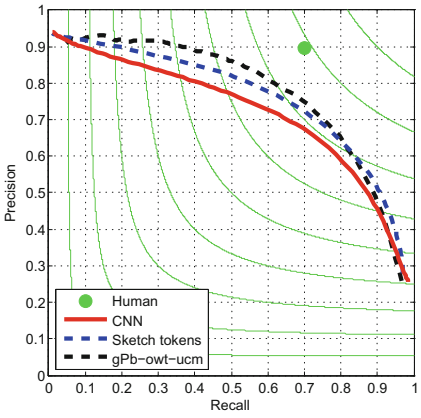

Fig. 5. Precession-recall curves.

Table 2. Performance measurements.

	ODS	OIS	AP	Speed
Human	.80	-	-	-
CNN (GPU)	.69	.71	.71	2 s
Canny [6]	.60	.64	.58	1/15 s
BEL [8]	.67	-	-	10 s
gPb [1]	.71	.74	.65	60 s
gPb-owt-ucm [1]	.73	.76	.73	240 s
SCG [15]	.74	.76	.77	280 s
Sketch tokens [9]	.73	.75	.78	1 s

Our network can also capture local edge structures. As shown in Fig. 7, filters in the first convolutional show different color changes, capturing step edges or textures. Both of them are local structures we want.

3.5 Comparison on Different Configurations

We have also tried different network structure, channel configuration, input size of image patches and preprocessing techniques in our experiments.

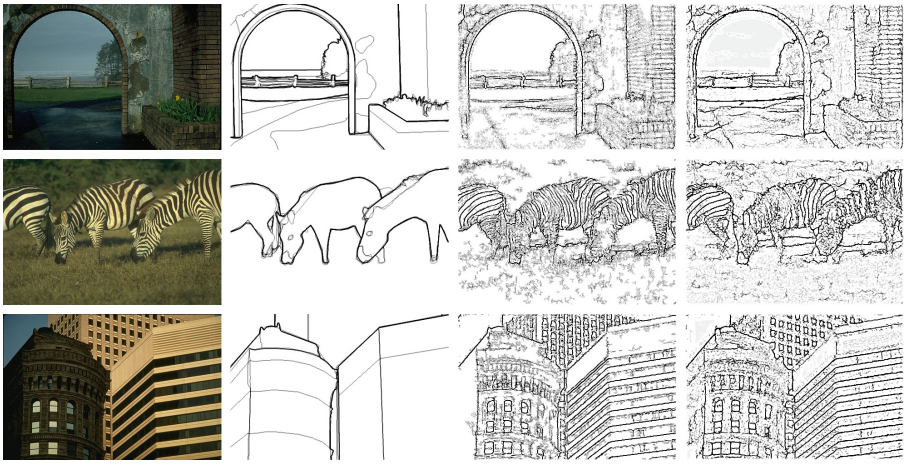

Fig. 6. Results on images with textures. The first and second column shows the original image and the ground truth respectively. The third and forth column shows the raw output of our CNN and sketch token [9] respectively.

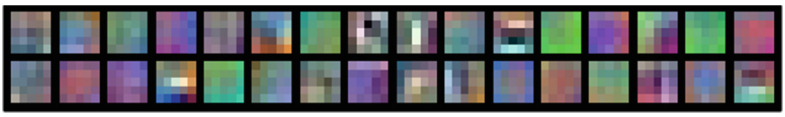

Fig. 7. Learned filters in the first convolutional layer.

Table 3 compares the performance when using different network structures. We tried a complex network architecture consisting of three convolutional layers, three pooling layers and two fully connected layers. However, this complex structure can hardly improve the performance while it takes much longer time on training and overfits easily. Usually a network with two convolutional layers and one fully-connected layer is enough for this problem. So we chose using this structure in most our experiments.

Table 3. Performance comparison on using different network structures. L = locally connected layer, P = pooling layer, C = convolutional layer, F = fully connected layer.

Image patch	Network	ODS	OIS	AP
gray	LPLPLPFF	.64	.66	.57
L0 smooth	CCF	.67	.68	.68
17*17*1	CCF(bias=0)	.66	.68	.66
CIE+grad	LPLPLPFF	.64	.65	.64
17*17*8	CCF	.64	.65	.63
CIE+grad	CCF	.68	.70	.70
21*21*8	CCFF	.68	.69	.66

In our experiments, we tried four types of channel configuration of the input image patches, (1) single gray scale channel, (2) RGB channels, (3) multiple gray scale channels on different scales and (4) a combination of CIR-LUV, one gradient magnitude and four gradient orientation channels, which is similar with [9]. We found that gray level inputs worked a little bit worse than RGB inputs. Multi-scale input hardly contributed to the performance. Multiple feature channels performed almost the same as RGB while requiring much more time on training and consuming huge amount of memory. Based on these results, we can demonstrate that there is no need to design extra feature extraction steps in our method.

For patch size and preprocessing techniques, we found that the performance would decrease if patch size was less than 17*17 and would not increase a lot if patch size was beyond 21*21. L0 smooth [12] would usually increase the performance when using single-channel gray-scaled image patches as input while decrease the performance when using RGB. Table 4 is a conclusion of our

Table 4. Performance comparison when using different input image patches.

Network	Image Patch		ODS	OIS	AP
	Channel	Size			
CCFF	CIE+grad	21*21*8	.68	.70	.70
CCFF	RGB	35*35*3	.66	.68	.62
CF	RGB	35*35*3	.63	.65	.60
CCFF	RGB	25*25*3	.68	.70	.68
CCF	RGB	23*23*3	.68	.70	.70
CCF	RGB+L0	23*23*3	.66	.68	.62
CCF	RGB	21*21*3	.68	.69	.65
CCF	RGB	17*17*3	.67	.69	.63
CCF	gray+L0	31*31*1	.67	.69	.70
CCF	gray+L0	25*25*1	.67	.69	.69
CCF	gray+L0	17*17*1	.67	.68	.68
CCF	gray+L0	13*13*1	.65	.66	.60
CCF	gray 3 scale	21*21*1	.65	.67	.65
CCF	gray 4 scale	21*21*1	.64	.65	.60

experiment results. These results were achieved when applying morphological thin in the post-processing step.

4 Conclusion

In this work, we developed a deep learning method for solving the edge detection problem by using convolutional neural networks (CNN). Unlike previous work, our approach does not need extra feature extraction process and can be very simple and fast while achieving good result. It is also very easy for people to implement and integrate our simple algorithm into their own computer vision systems. Moreover, with deep learning becoming more and more popular, people try using CNN in every field of computer vision. It is probably that our network can be concatenated in front of the network used in other application. The whole network can then be jointly fine tuned. This is a particular advantage of CNN and have been studied in recent papers [17].

References

1. Arbelaez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **33**, 898–916 (2011)
2. Ferrari, V., Fevrier, L., Jurie, F., Schmid, C.: Groups of adjacent contour segments for object detection. IEEE Trans. Pattern Anal. Mach. Intell. **30**, 36–51 (2008)

3. Yokoyama, M., Poggio, T.: A contour-based moving object detection and tracking. In: 2nd Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance 2005, pp. 271–276. IEEE (2005)
4. Yilmaz, A., Li, X., Shah, M.: Contour-based object tracking with occlusion handling in video acquired using mobile cameras. IEEE Trans. Pattern Anal. Mach. Intell. **26**, 1531–1536 (2004)
5. Vacchetti, L., Lepetit, V., Fua, P.: Combining edge and texture information for real-time accurate 3d camera tracking. In: Third IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR 2004, pp. 48–56. IEEE (2004)
6. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**, 679–698 (1986)
7. Zheng, S., Yuille, A., Tu, Z.: Detecting object boundaries using low-, mid-, and high-level information. Comput. Vis. Image Underst. **114**, 1055–1067 (2010)
8. Dollar, P., Tu, Z., Belongie, S.: Supervised learning of edges and object boundaries. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 1964–1971 (2006)
9. Lim, J.J., Zitnick, C.L., Dollár, P.: Sketch tokens: a learned mid-level representation for contour and object detection. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3158–3165 (2013)
10. Dollár, P., Zitnick, C.L.: Structured forests for fast edge detection. In: 2013 IEEE International Conference on Computer Vision (ICCV), pp. 1841–1848 (2013)
11. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**, 541–551 (1989)
12. Xu, L., Lu, C., Xu, Y., Jia, J.: Image smoothing via l 0 gradient minimization. ACM Trans. Graph. (TOG) **30**, 174 (2011)
13. Berkeley Segmentation Data Set and Benchmarks 500 (bsds500). (http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
15. Xiaofeng, R., Bo, L.: Discriminatively trained sparse code gradients for contour detection. In: Advances in Neural Information Processing Systems, pp. 584–592 (2012)
16. Li, H., Zhao, R., Wang, X.: Highly Efficient Forward and Backward Propagation of Convolutional Neural Networks for Pixelwise Classification (2014). arXiv preprint arXiv:1412.4526
17. Ouyang, W., Wang, X.: Joint deep learning for pedestrian detection. In: 2013 IEEE International Conference on Computer Vision (ICCV), pp. 2056–2063 (2013)

http://www.springer.com/978-3-319-40662-6

Advances in Neural Networks – ISNN 2016
13th International Symposium on Neural Networks, ISNN 2016, St. Petersburg, Russia, July 6-8, 2016, Proceedings
Cheng, L.; Liu, Q.; Ronzhin, A. (Eds.)
2016, XX, 741 p. 278 illus., Softcover
ISBN: 978-3-319-40662-6