DeepContour: A Deep Convolutional Feature Learned by Positive-sharing Loss for Contour Detection

draft version, CVPR2015

Wei Shen[1], Xinggang Wang[2], Yan Wang[3], Xiang Bai[2], Zhijiang Zhang[1]
[1] Key Lab of Specialty Fiber Optics and Optical Access Networks, Shanghai University
[2] School of Electronic Information and Communications, Huazhong University of Science and Technology
[3] Rapid-Rich Object Search Lab, Nanyang Technological University
wei.shen@shu.edu.cn,xgwang@hust.edu.cn,wyanny.9@gmail.com,xbai@hust.edu.cn,zjzhang@shu.edu.cn

Abstract

Contour detection serves as the basis of a variety of computer vision tasks such as image segmentation and object recognition. The mainstream works to address this problem focus on designing engineered gradient features. In this work, we show that contour detection accuracy can be improved by instead making the use of the deep features learned from convolutional neural networks (CNNs). While rather than using the networks as a blackbox feature extractor, we customize the training strategy by partitioning contour (positive) data into subclasses and fitting each subclass by different model parameters. A new loss function, named positive-sharing loss, in which each subclass shares the loss for the whole positive class, is proposed to learn the parameters. Compared to the sofmax loss function, the proposed one, introduces an extra regularizer to emphasizes the losses for the positive and negative classes, which facilitates to explore more discriminative features. Our experimental results demonstrate that learned deep features can achieve top performance on Berkeley Segmentation Dataset and Benchmark (BSDS500) and obtain competitive cross dataset generalization result on the NYUD dataset.

1. Introduction

In this paper, we investigate a classical and fundamental problem in computer vision contour detection in natural images. Accurately detecting object contours serves as the basis for many tasks, such as image segmentation [2], scene understanding [3] and object detection [17, 30, 48].

Contour detection is quite challenging and more difficult than edge detection. According to Martin's definition [34], the latter one aims to detect the characteristic changes in brightness, color, and texture; In contrast, the goal of the former one is to find the changes in pixel ownership from one object or surface to another. Therefore, contours involve the concept of objects, which casts an obstacle to us: How to distinguish the changes caused by cluttered textures and those that correspond to object boundaries? Many researchers have devoted their efforts to addressing this problem and achieved considerable progress [34, 32, 38, 2, 39, 30, 11, 23, 12]. However, obvious performance gap is still observed between contour detections given by the algorithms and human annotations.

The traditional framework for contour detection designs a variety of gradient features for each image pixel, followed by learning a binary classifier to determine whether an image pixel is passed through by contours or not. Although hand-designed features are widely used and support top-ranking algorithms on the standard contour detection benchmark [35, 2] in the past decade, we cannot ignore a fact that they are not discriminative enough to differentiate the semantic object boundaries and abrupt changes in low level image cues. Driven by the recent prevailing trend of deep neural networks, a few researchers try to learn the deep features to address the contour vs non-contour classification problem [23, 19, 33]. The impressive performances for many computer vision tasks have proved that the deep features outputted by deep neural networks are powerful [1, 37] and tend to replace the traditional hand-designed features, such like SIFT [31] and HOG [9]. Therefore, introducing the deep learning techniques into the contour detection problem is reasonable and feasible. However, how to learn discriminative and representative deep features for contour detection is not trivial. The current usage of deep learning for contour detection is to take deep networks as blackbox models to learn the probability of the contour [23] or the local contour map [19] for each pixel. By such a way, the flexibility of deep networks may be insufficient even when employing a very complex and deep architecture [19]. Why deep networks can achieve breakthroughs on so many gen-

eral object recognition tasks in a walk, while they are denied by contour detection? The reason is that the diversity of contours in a local patch is huge, as they contain many types of patterns with multiple orientations, such as straight lines, parallel lines, T-junctions and Y-junctions. Even though deep networks are powerful, to treat the data with so large variations as one class is not advisable.

Our goal is to learn discriminative features for contour detection by the well-known deep convolutional neural networks (CNNs) [18, 42, 26]. However, we do not just use the CNN as a blackbox feature extractor but derive it by considering the intrinsic properties of contours. We emphasize the following two points: (1) Partitioning contour patches into compact clusters according to their inherent structures is necessary for training an effective CNN model. Such a clustering process does lead to a mid-level shape representation for contour patches. The formed clusters are called shape classes, and each of them are assigned by a shape label. Fitting contour data of different shapes by different model parameters is followed by the spirit of divide-and-conquer strategy [4], which can ease the training difficulty due to the diversity of the data. (2) How to define the loss function of the CNN is significant for learning a discriminative feature for contour detection. The aforementioned clustering process converts the binary classification problem (i.e. predicting whether an image patch belongs to contour or non-contour) to a multi-class problem (i.e. predicting whether an image patch belongs to each shape class or the negative class), which seems can be well solved by minimizing the softmax loss, the loss function used in standard CNNs. However, softmax function penalizes the loss of each class equally, which is not suitable for learning the discriminative features between contour patches and background patches. Because, for contour detection, the misclassification among shape classes is ignorable, while a contour patch is classified as a background one is a considerable error or vice versa. Therefore, the losses for positive versus negative should be emphasized in the training. Based on this observation, we define a new objective function which combines an extra loss for contour versus non-contour with the softmax loss. As in this function the loss for the positive class is shared among each shape class, we name it positive-sharing loss. The extra loss brings in better regularization which lead to better contour feature learning. With the learned deep features, contour detection can be performed by feeding them into any classifiers followed by a standard non-maximal suppression scheme [6]. We take the computationally efficient structured forest [11, 12] as the contour versus non-contour classifier for our deep features, which achieves the-state-of-the-art result on the popular Berkeley Segmentation Dataset and Benchmark (BSDS500) [2].

2. Related Work

Contour Detection: Contour detection is usually considered as a supervised learning problem. In the pioneer work of Konishi *et al*. [24], contour detection is formulated as a discrimination task specified by a likelihood ratio tested on the filter responses. Martin *et al*. [34] carefully design features to characteristic changes in brightness, color, and texture associated with natural boundaries, and learn a classifier to combine the features. Dollar *et al*. [10] use rich patch features and probabilistic boosting tree [46] to detect contours. Ren and Bo [39] find a sparse coding gradient feature which is effective for contour detection. Recently, a mid-level feature named sketch token and a random forest based structural classifier are proposed in [30] and [11] respectively. Besides of supervised learning, in [2], Arbelaez *et al*. combine multiple local cues into a globalization framework based on spectral clustering for contour detection.

Deep Learning: Deep learning methods have obtained great successes for various applications in computer vision, such as image classification [25], object detection [20], image labeling [15], and super-resolution [14]. Deep convolutional neural networks (CNNs) with GPU implementation and the tricks like rectified linear unit (ReLU) [36] and dropout [44] are popular for feature learning and supervised classification. Restricted Boltzmann machine (RBM), autoencoder [47] and their variants are popular for unsupervised deep learning.

Deep Learning for Contour Detection: As far as we know, there are two papers using deep learning for contour detection. The first one is proposed by Kivinen *et al*. in [23], in which image feature is learned by using RBM and classified multiple read-out layers. The other one is proposed by Ganin and Lempitsky in [19], in which feature for image patch is learned using a conventional CNN and then the feature is mapped to an annotation edge map using kd-tree. Different from these two methods, we learn deep features using shape labels and a CNN with a novel loss.

3. Data Preparation

In this section we describe how to prepare the training and validation data set for training a CNN model. Take the BSDS500 dataset as an example. The BSDS500 dataset contains 200 training, 100 validation, and 200 testing images. Each image has hand labeled ground truth contours. Following [30], we obtain the shape classes by clustering patches extracted from the binary images representing the hand labeled ground truth contours. Only patches whose centers are passed through by labeled contours are used for clustering. We make a customization that we use patches with the size of 45×45 pixels (rather than 35×35 in [30]) in our experiments, since CNNs have strong learning ability

to handle more information. This clustering process leads to K shape classes, whose patterns are ranging from straight lines to more complex structures, as visualized in Fig. 1.

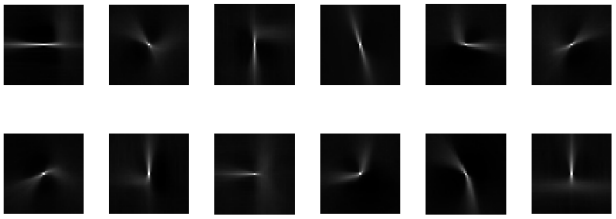

Figure 1. The visualization of some selected shape classes.

With the learned K shape classes, we can assign a label y to each color image patch x extracted from the images in the dataset. If x is a contour patch (we set a maximum slop equals to 3 pixels allowed for the thumbnail distance between a patch center to a contour), its class label is supplied by the shape clustering results, i.e. its class label $y = k(k \in \{1, \ldots, K\})$ if its corresponding patch extracted from the hand labeled ground truth binary image belongs to k-th shape class. Otherwise, if x is a background patch, its label is assigned by $y = 0$. For denotational simplicity, contour patch and background patch are also called positive patch and negative patch in the rest of the paper.

We sample 2,000,000 image patches from the training set of the BSDS500 dataset, in which the numbers of the positive and negative patches are equal, to form the training data for learning our CNN model. Another 2,000,000 image patches with the same numbers of the positive and negative patches are sampled from the validation set of the BSDS500 dataset for cross-validation when training our CNN model.

4. Learning Deep Contour Features by CNN

In this section, we describe how to learn the deep features for contour detection by our CNN model. First, we introduce the architecture of our CNN model. Then we discuss how to define proper loss functions for our task.

4.1. CNN Architecture

We train our CNN on a multi-class classification task, namely to classify an image patch to which shape class or the negative class. Fig. 2 depicts the overall architecture of our CNN, which contains six layers with parameters to be learned; the first four are convolutional and the remaining two are fully-connected. Only convolutional and fully-connected layers contain learnable parameters, while the others are parameter free.

The input of our CNN is a 3-channels (RGB) image patch of size 45×45, which is filtered by the first convolutional layer (COV1) with 32 kernels of size $5 \times 5 \times 3$ with a padding of 2 pixels. The resulting $45 \times 45 \times 32$ feature maps are then sequentially given to a local response normalization layer (LRN1) and a max-pooling layer (MAXP1) which performs max pooling over 3×3 spatial neighborhoods with a stride of 2 pixels. The output of the MAXP1 is then passed to the second convolutional layer (COV2). Except for the number of filter kernels, the parameter configurations for the four convolutional layers are the same. Each convolutional layer consists of a Rectified Linear units (ReLU) and is followed by a LRN, and a MAXP, except the last one, where only a MAXP is applied. The convolutional layers are the core of the CNN, which provide various feature maps [13], while the max-pooling layers make the activation features which are robust to slight shifts in contour placement. The output of the fourth max-pooling layer (MAXP4) is fed into the first fully-connected layer (FC1), which also consists of a ReLU. The fully-connected layer, in which each output unit is connected to all input nodes, is able to capture correlations between features activated in distant parts of the image patch [45], such like the boundary of the stamen in the example image patch in Fig. 2. To reduce the risk of overfitting, we use dropout [25, 44] in FC1, which consists of setting to zero the output of each neuron nodes with probability 0.5. The output of FC1 in our CNN will be used as the deep features for contour detection, which is a 128-dimensional feature vector. We can also use more units in this layer to form a feature vector of higher dimensions. However, it will induce heavier computational burden. In a standard CNN, the output of the last fully-connected layer will be fed into a softmax classifier to produce a distribution over class labels. In our case, assume that we have $K = 50$ shape classes, then the unit number of the final layer of the CNN should be 51.

According to the parameter configuration of each layer, the architecture of the CNN can be described concisely by layer notations with layer sizes:
COV1($45 \times 45 \times 32$) $\rightarrow$ LRN1 $\rightarrow$ MAXP1 $\rightarrow$
COV2($22 \times 22 \times 48$) $\rightarrow$ LRN2 $\rightarrow$ MAXP2 $\rightarrow$
COV3($10 \times 10 \times 64$) $\rightarrow$ LRN3 $\rightarrow$ MAXP3 $\rightarrow$
COV4($4 \times 4 \times 128$) $\rightarrow$ MAXP4 $\rightarrow$ FC1(128) $\rightarrow$ FC2(101).

Note that, the number of the layers in our CNN architecture is less than the generic one used for ImageNet LSVRC [5], as the contour is always represented by a local image patch with smaller size than generic objects. From our experiences, four convolutional layers are enough to capture the discriminative information between contour and background patches.

4.2. Positive-sharing Loss Function

The goal of training a standard CNN is to maximize the probability of the correct class, which is achieved by minimizing the softmax loss. Given a training set which contains m image patches: $\left\{x^{(i)}, y^{(i)}\right\}_{i=1}^{m}$, where $x^{(i)}$ is the i-th image patch and $y^{(i)} \in \{0, 1, \ldots, K\}$ is its class label.

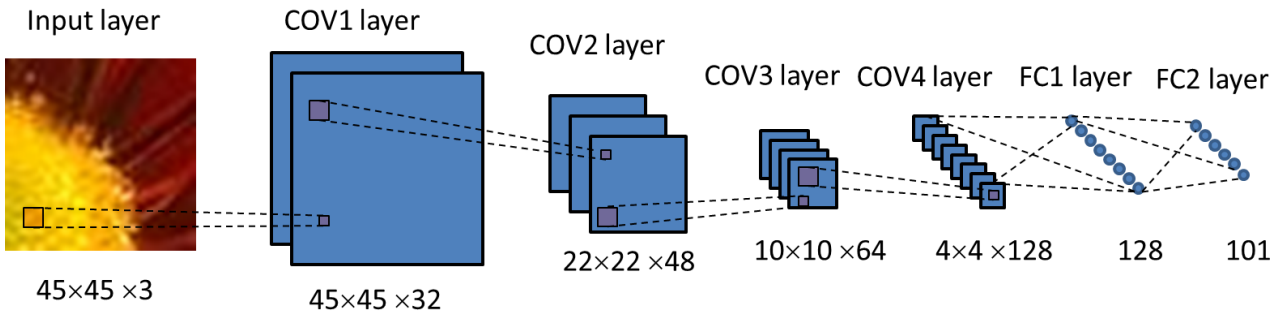

Figure 2. An illustration of the architecture of our CNN, explicitly visualizing the dimensions of each network layers. Due to the limited space, we only show the layers with learnable parameters. The convolutional layers are represented by blue squares, while fully-connected ones are marked by blue dots. The big and small light red blocks depict the convolutional kernels and results, respectively.

If $y^{(i)} = 0$, then $x^{(i)}$ is a negative patch; If $y^{(i)} = k > 0$, then $x^{(i)}$ is a positive patch and belongs to the k-th shape class. Let $(a_j^{(i)}; j = 0, 1, \dots K)$ be the output of unit j in FC2 for $x^{(i)}$, then the probability that the label of $x^{(i)}$ is j is given by

$$p_j^{(i)} = \frac{\exp(a_j^{(i)})}{\sum_{l=0}^{K} \exp(a_l^{(i)})}. \tag{1}$$

In a standard CNN, the output of FC2 is then fed into a $(K+1)$-way softmax which aims to minimize the following loss function:

$$J_0 = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=0}^{K} \mathbf{1}(y^{(i)} = j) \log p_j^{(i)}\right], \tag{2}$$

where $\mathbf{1}(\cdot)$ is the indicator function. The softmax loss function penalizes the classification error for each class equally. However, in our case, to estimate the label of a positive patch to be a wrong nonzero label is not a significant error, since it still be predicted as a positive one. That is to say, the losses induced by the incorrect estimation between the zero label and nonzero labels should be more concerned. To this end, we add a new term to regularize the loss:

$$J = J_0 - \frac{1}{m}\left[\sum_{i=1}^{m} \lambda\Big(\big(\mathbf{1}(y^{(i)} = 0) \log p_0^{(i)} + \sum_{j=1}^{K} \mathbf{1}(y^{(i)} = j) \log(1 - p_0^{(i)})\big)\Big)\right], \tag{3}$$

where λ is a controlling parameter. When λ is small, Eq. 3 tends to be the softmax loss function; While when λ is larger, the effect of the shape partition becomes weaker and Eq. 3 tends to be an objective function for addressing the binary classification problem, contour versus non-contour. We set $\lambda = 1$ as the default value in all our experiments, unless otherwise specified. The intuition behind Eq. 3 is that we should fit the data from different shape classes by different model parameters, as they are quite different in feature space; While we cannot treat them as "absolutely" distinct classes when computing the classification loss, since they all belong to the positive class. We call Eq. 3 positive-sharing loss function, as the loss for the positive class is shared among each shape class in it.

To apply standard back-propagation [41, 27] to optimize the parameters of the network, computing the partial derivatives of the new loss w.r.t. the output of FC2, $(a_j^{(i)}; j = 0, 1, \dots K)$, is required. The first term in Eq. 3 is the standard softmax loss, whose derivatives have been already provided in the literature. The rest thing is to derive the partial derivatives of the second term w.r.t $a_0^{(i)}$ and $a_{l(l=1,\dots K)}^{(i)}$. One can show that

$$\begin{aligned} &\frac{\partial \log p_0^{(i)}}{\partial a_0^{(i)}} = 1 - p_0^{(i)}, \frac{\partial \log(1 - p_0^{(i)})}{\partial a_0^{(i)}} = -p_0^{(i)}, \\ &\frac{\partial \log p_0^{(i)}}{\partial a_l^{(i)}} = -p_l^{(i)}, \frac{\partial \log(1 - p_0^{(i)})}{\partial a_l^{(i)}} = \frac{p_l^{(i)} p_0^{(i)}}{1 - p_0^{(i)}}. \end{aligned} \tag{4}$$

Then the partial derivatives of the new loss are obtained by

$$\frac{\partial J}{\partial a_0^{(i)}} = \frac{1}{m}\Bigg[(\lambda+1)\mathbf{1}(y^{(i)} = 0)(p_0^{(i)} - 1) + (\lambda+1)\sum_{j=1}^{K} \mathbf{1}(y^{(i)} = j) p_0^{(i)}\Bigg], \tag{5}$$

and

$$\frac{\partial J}{\partial a_l^{(i)}} = \frac{1}{m}\Bigg[\big(\lambda \mathbf{1}(y^{(i)} = 0) + 1\big)p_l^{(i)} - \mathbf{1}(y^{(i)} = l) - \lambda \sum_{j=1}^{K} \mathbf{1}(y^{(i)} = j)\Big(\frac{p_0^{(i)} p_l^{(i)}}{1 - p_0^{(i)}}\Big)\Bigg]. \tag{6}$$

One can verify the effectiveness of the CNN model by a per-patch classification accuracy on the validation set.

However, for detection problem, the performance is more likely related to the contrast between positive and negative samples. So we define a contrast score as the measure. Let $\{x_v^{(i)}, y_v^{(i)}\}_{i=1}^m$ be the validation set and $p_0^{(i)}$ is the probability of $x_v^{(i)}$ belongs to negative class outputted by the trained CNN. The contrast score is computed by

$$\gamma = \frac{1}{m}\sum_{i=1}^{m}\Big[(\mathbf{1}(y_v^{(i)}=0)-\mathbf{1}(y_v^{(i)}>0))(p_0^{(i)}-(1-p_0^{(i)}))\Big], \tag{7}$$

which ranges from -1 to 1, measuring the discrimination of the learned model between positive and negative samples. We run stochastic gradient descent (SGD) on the target loss functions, by setting a learning rate set to 0.001. After $100,000$ iterations, the contrast score of the standard CNN is about 0.56 , while ours is about 0.59, with 0.03 improvement.

5. Experimental results

We analyze the performance of our deep convolutional features for contour detection. To learn our CNN model, we take the publicly available modifiable implementation named "Caffe" [22] and modify the softmax loss layer to ours. The 128-dimensional feature vector outputted by the first fully-connected layer (FC1) is our deep features. The feature vectors for all pixels in a patch are concatenated into one to be fed into a structured forest classifier [11, 12] to perform contour detection. We start by visualizing the learned deep features. Next, we compare contour detection results on the BSDS500 dataset [2] to the state-of-the-art. Then, the cross dataset generalization of our deep features are validated on the NYUD dataset [43]. At last, we analyze the influence of parameters. For the structured forest classifier, we use its default parameter setting in all our experiments.

5.1. Deep Feature Visualization

Although it has been a common sense that CNNs are effective feature extractors, to see what we exactly learned from the millions of image patches is still necessary and it can help us to understand what is being learned. Given an image, the image patches densely sampled from it are inputted into our CNN, which results in 128 feature maps outputted by FC1. We randomly select some feature maps and visualize them in Fig. 3. Encouragingly, although some of them suffer from noise caused by textured regions, still many of them are quite sparse and well capture the contour fragments of the objects, which will facilitate contour detection.

5.2. Results on BSDS500 Dataset

The majority of our experiments are performed on the BSDS500 dataset. The details of this dataset has been introduced in Sec. 3. To evaluate the performance of a contour detection algorithm, three standard quantities are used: the best F-measure on the dataset for a fixed scale (ODS), the aggregate F-measure on the dataset for the best scale in each image (OIS), and the average precision (AP) on the full recall range. [2]. We compare our contour detection method against other leading methods, such as Structured Edge (SE) [11] as well as its Variants (SE-Var) [12], Sparse Code Gradients (SCG) [39] and Deep Neural Prediction Network (DeepNet) [23]. Precision/recall curves are shown in Fig. 4 and summary statistics are in Table 1.

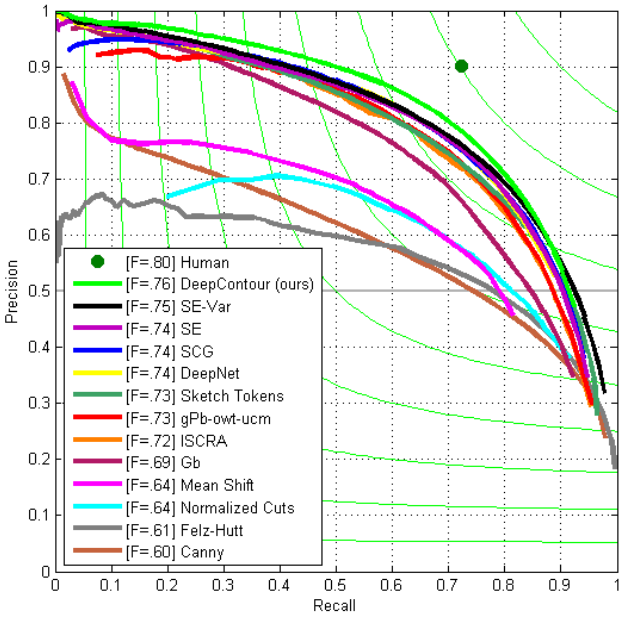

Figure 4. Evaluation of contour detectors on the BSDS500 dataset [2]. Leading contour detection methods are ranked according to their best F-measure (ODS) with respect to human ground truth contours. Our method, DeepContour achieves the top result and shows both improved recall and precision at most of the precision-recall regime. See Table 1 for more details ahout the other two quantities and method citations.

Our method, DeepContour, outperforms all state-of-the-art methods. We improve ODS/OIS by 1 point over competing methods and achieve comparable AP. The comparison between SE [11] as well as SE-Var [12] and ours directly prove that the learned deep features are more discriminative than the hand-designed features, such as the gradient channel features and the self-similarity features, used in [11, 12]. In comparison to other deep learning-based approaches to contour detection, we fairly outperform DeepNet [23] which unsupervised learns the contour features, because learning features with supervision improves the discrimination. Noticeable, to handle the transformation between contours of different shape classes, DeepNet is applied to 16 rotated versions of each image and the prediction results are averaged for enhancing the performance

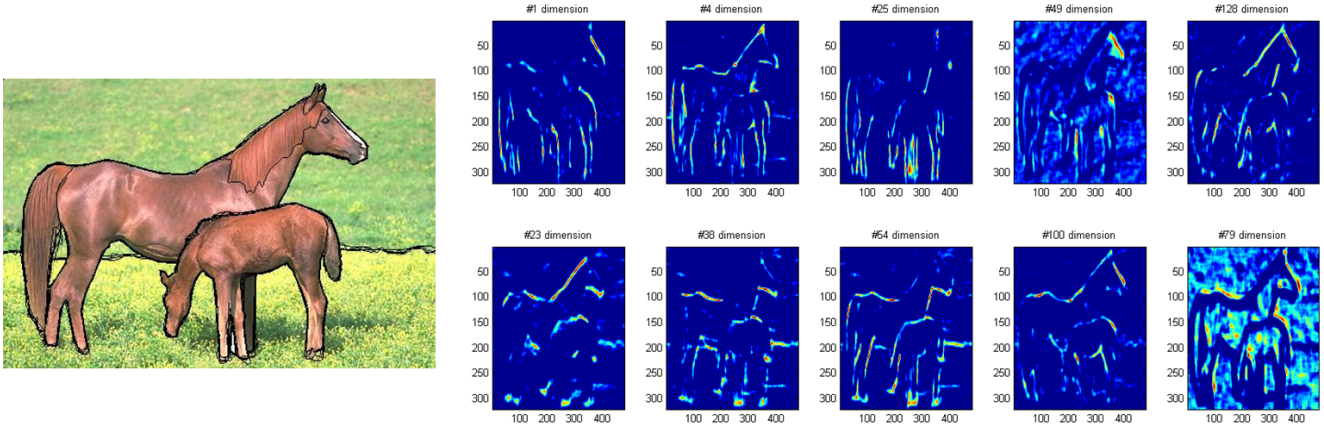

Figure 3. *Left*: The testing color image. The ground truth contours are visualized by black curves. *Right*: Randomly selected deep feature maps.

Table 1. Contour detection results on the BSDS500 dataset [2].

	ODS	OIS	AP
Human	.80	.80	-
Canny [6]	.60	.63	.58
Felz-Hutt [16]	.61	.64	.56
Normalized Cuts [8]	.64	.68	.45
Mean Shift [7]	.64	.68	.56
Gb [29]	.69	.72	.72
ISCRA [40]	.72	.75	.46
gPb-owt-ucm [2]	.73	.76	.73
Sketch Tokens [30]	.73	.75	.78
DeepNet [23]	.74	.76	.76
SCG-[39]	.74	.76	.77
PMI+sPb [21]	.74	.77	.78
SE [11]	.74	.76	.78
SE-Var [12]	.75	.77	.80
N^4-Fields [19]	.75	.77	.78
DeepContour (ours)	**.76**	**.78**	**.80**

in [23], while we have considered the diversity between different shape classes in training procedure. Unfortunately, the result of DeepNet before enhancement is not reported; so we are unable to perform more detailed discussion about the effectiveness of shape class partition. Our method also achieves better performance than another deep learning based approach N^4-Fields [19], which also adopts CNNs to learn contour features. Their CNN model is targeted on a local contour map, which implicitly performs shape class partition by CNN itself. However, they use nearest neighbor search in the CNN feature space to obtain the local contour map, which may perform poorly due to the noisy responses in the CNN feature maps, as shown in Fig. 3. As we apply random forest to our CNN features, the feature selection mechanism embedded in this classifier improves the robustness against the noise. As we obtain the shape classes according to the definition of sketch tokens [30], the considerable performance improvement between their and our method proves the superiority of our deep features. We illustrate the contour detection results obtained by several methods in Fig. 5 for qualitative comparison. These qualitative examples shows that our method fires stronger responses on the ground truth contours and meanwhile suppresses the false positive.

5.3. Cross Dataset Generalization

One may concern the deep features learned from one dataset might lead to higher generalization error when applying them to another dataset. To explore whether this is the case, we apply the deep features learned from the training set of the BSDS500 dataset to the NYUD dataset [43]. The NYUD dataset (v2) includes 1449 pairs of color and depth frames of resolution 480x640, with groundtruth semantic regions. This dataset is comprised of images from a variety of indoor scenes, while the images in BSDS500 mainly illustrates the outdoor scenes. Consequently, the objects in these two datasets are totally different. We use the same experimental setup proposed by [39], which chooses 60% images for training and 40% for testing with the images reduced to 320×240 resolution. As our deep features are learned from color images from the BSDS500 dataset, we only apply them to the color images of the NYUD dataset. For comparison, we list the cross dataset generalization results (applying the model learned on the BSDS500 dataset to the NYUD dataset) of our method and SE [11] in Table 2. The results of gPb and SCG are used for reference, while they are obtained by training on the NYUD dataset. Although supervised learning usually decrease the generality, the learned deep features achieves comparable or better cross dataset generalization result than SE and significantly outperforms gPb-owt-ucm, even if it is trained on NYUD dataset. The selected qualitative examples in Fig. 6

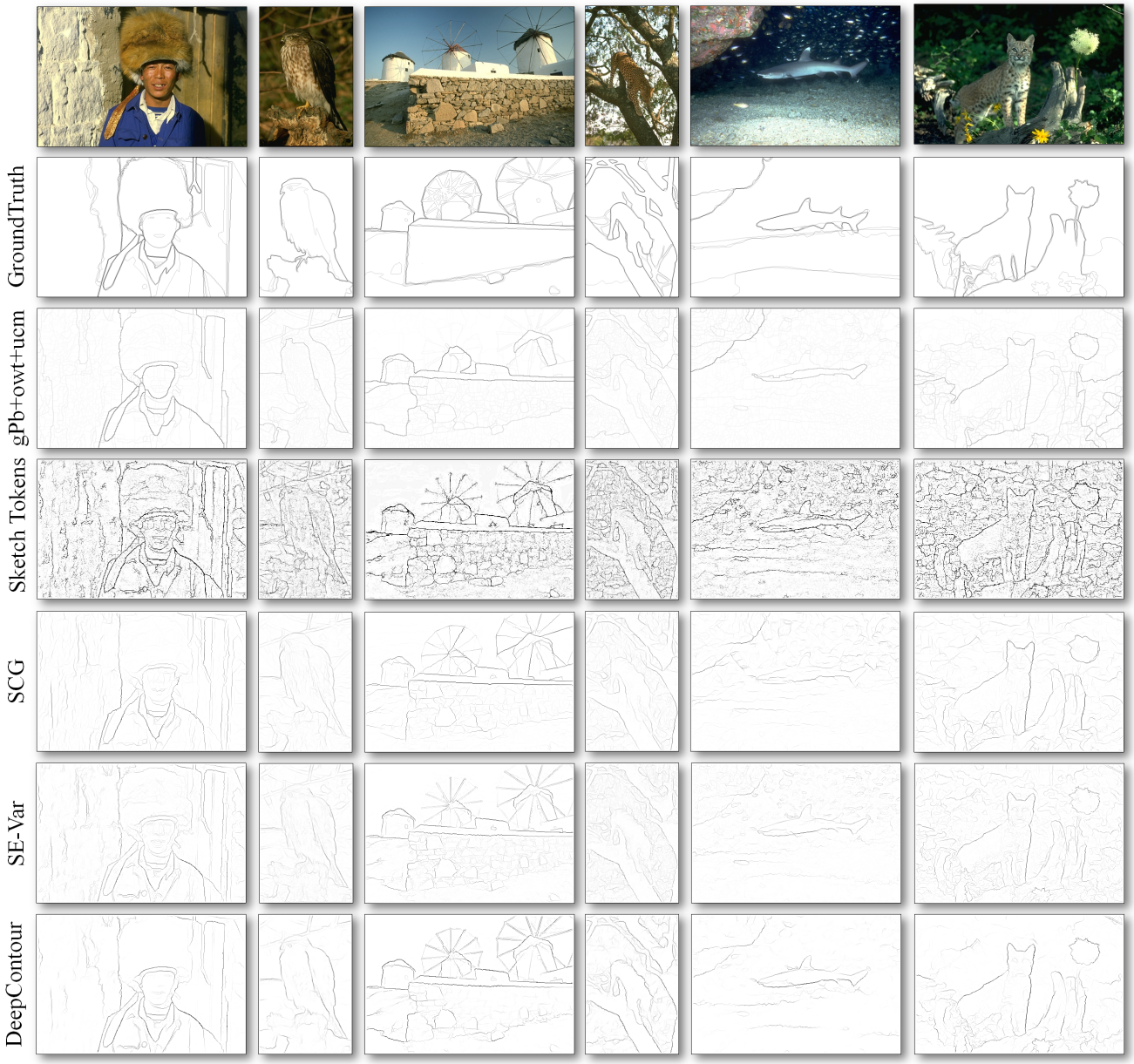

Figure 5. Illustration of contour detection results on the BSDS500 dataset for six selected example images. The first two rows show the original image and the ground truth. The next five rows depict the results for gPb-owt-ucm [2], Sketch Tokens [30], SCG [39], SE-Var [12] and our DeepContour. Note that our method fires stronger responses on the ground truth contours (such as the contour of the whale in the fifth column) and meanwhile suppresses the false positive (such as the edges of the little fishes in the fifth column). It's better to use viewer zoom functionality to see fine details.

are obtained by our deep contour features learned from the BSDS500 dataset, which evidence that our deep feature is a general and portable contour representation.

5.4. Parameter Discussion

We conduct to validate the influence of parameters introduced in our method on the test set of the BSDS dataset. There are two important parameters for learning our deep features: the number of the shape classes K and the controlling parameter λ in our loss function. In Fig. 7 we explore the effect of choices of these two parameters. The standard metric OSD is used to measure the accuracy. For saving time, we no longer apply the multi-scale strategy in the structured forest [11], which will result in a reduction (about 0.006) in OSD. Note that, setting $K = 1$ means the shape class partition is not performed, then our model is equiva-

Figure 6. Illustration of our contour detection results on the NYUD dataset [43] for five selected example images (Depth images are not used). In each example, we see the original image and the ground truth and our results, respectively. Although our deep contour features are learned from the BSDS dataset, they can represent the object contours in NYUD dataset well.

Table 2. Cross-dataset generalization results. TRAIN/TEST indicates the training/testing dataset used.

	ODS	OIS	AP
gPb [2] (NYU/NYU)	.51	.52	.37
SCG [39] (NYU/NYU)	.55	.57	.46
SE [11] (BSDS/NYU)	.55	.57	.46
DeepContour (BSDS/NYU)	**.55**	**.57**	**.49**

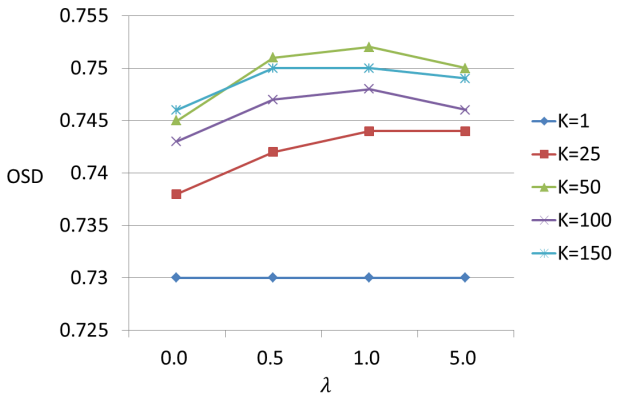

Figure 7. The accuracy obtained by parameter sweeping. K is the number of partitioned shape classes, and λ is the controlling parameter introduced in our loss function.

lent to learn a binary contour versus non-contour classifier with softmax loss, which reduces the performance considerably. This evidences that shape class partition is necessary. The best performance is achieved by setting $K = 50$. As for λ, if $\lambda = 0$, then our loss function is reduced to the softmax loss which also decrease the performance. Setting a considerable larger λ, our loss function tends to only focus on the loss for contour and non-contour classes, which also results in a reduction in performance.

6. Conclusion and Future Work

In this work we successfully showed how to learn discriminative features from deep convolutional neural networks for contour detection in natural images. We emphasized two points: one was partition contour (positive) data subclasses was necessary for training an effective CNN model, the other was the proposed positive-sharing loss function, which emphasized the losses for the contour and non-contour rather than the loss for each subclass, facilitating to explore more discriminative features than softmax loss function. The experiments on the BSDS500 dataset [2] demonstrated that the proposed algorithm outperformed other competing methods in the literature. Through parameter sweeping, we verified the necessity of positive data partition and the effectiveness of the proposed loss function. We also validated the cross dataset generality of our deep features on the NYUD data set [43].

The contour detection performance might be further improved by applying other deep networks, such as Deeply-Supervised Nets [28]. This will be investigated in our future work.

Acknowledgement. This work was supported in part by the National Natural Science Foundation of China under Grant 61303095 and 61222308, in part by Research Fund

for the Doctoral Program of Higher Education of China under Grant 20133108120017, in part by Innovation Program of Shanghai Municipal Education Commission under Grant 14YZ018, and in part by the Ministry of Education of China Project NCET-12-0217. We thank NVIDIA Corporation for providing their GPU device for our academic research.

References

[1] P. Agrawal, R. B. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *Proc. ECCV*, pages 329–344, 2014.

[2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, 2011.

[3] J. T. Barron and J. Malik. Intrinsic scene properties from a single RGB-D image. In *Proc. CVPR*, pages 17–24, 2013.

[4] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[5] A. Berg, J. Deng, and L. Fei-Fei. Imagenet large scale visual recognition challenge 2012. `http://www.image-net.org/challenges/LSVRC/2012/`, 2012.

[6] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[7] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002.

[8] T. Cour, F. Bénézit, and J. Shi. Spectral segmentation with multiscale graph decomposition. In *Proc. CVPR*, pages 1124–1131, 2005.

[9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. CVPR*, pages 886–893, 2005.

[10] P. Dollar, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *Proc. CVPR*, volume 2, pages 1964–1971, 2006.

[11] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *Proc. ICCV*, pages 1841–1848, 2013.

[12] P. Dollár and C. L. Zitnick. Fast edge detection using structured forests. *arXiv preprint arXiv:1406.5549*, 2014.

[13] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proc. ICML*, pages 647–655, 2014.

[14] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *Proc. ECCV*, pages 184–199. Springer, 2014.

[15] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1915–1929, 2013.

[16] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[17] V. Ferrari, L. Fevrier, F. Jurie, and C. Schmid. Groups of adjacent contour segments for object detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(1):36–51, 2008.

[18] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[19] Y. Ganin and V. S. Lempitsky. N^4-fields: Neural network nearest neighbor fields for image transforms. In *Proc. ACCV*, 2014.

[20] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, pages 580–587, 2014.

[21] P. Isola, D. Zoran, D. Krishnan, and E. H. Adelson. Crisp boundary detection using pointwise mutual information. In *Proc. ECCV*, pages 799–814, 2014.

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[23] J. J. Kivinen, C. K. I. Williams, and N. Heess. Visual boundary prediction: A deep neural prediction network and quality dissection. In *Proc. AISTATS*, pages 512–521, 2014.

[24] S. Konishi, A. L. Yuille, J. M. Coughlan, and S. C. Zhu. Statistical edge detection: Learning and evaluating edge cues. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(1):57–74, 2003.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1106–1114, 2012.

[26] Y. LeCun, B. Boser, J. Denker, D.Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), 1989.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *Proc. AISTATS*, 2015.

[29] M. Leordeanu, R. Sukthankar, and C. Sminchisescu. Generalized boundaries from multiple image interpretations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(7):1312–1324, 2014.

[30] J. J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *Proc. CVPR*, pages 3158–3165, 2013.

[31] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[32] M. Maire, P. Arbelaez, C. Fowlkes, and J. Malik. Using contours to detect and localize junctions in natural images. In *Proc. CVPR*, 2008.

[33] M. Maire, S. X. Yu, and P. Perona. Reconstructive sparse code transfer for contour detection and semantic labeling. In *Proc. ACCV*, 2014.

[34] D. R. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):530–549, 2004.

[35] D. R. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. ICCV*, pages 416–425, 2001.

[36] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. ICML*, pages 807–814, 2010.

[37] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, 2014.

[38] X. Ren. Multi-scale improves boundary detection in natural images. In *Proc. ECCV*, pages 533–545, 2008.

[39] X. Ren and L. Bo. Discriminatively trained sparse code gradients for contour detection. In *Proc. NIPS*, pages 593–601, 2012.

[40] Z. Ren and G. Shakhnarovich. Image segmentation by cascaded region agglomeration. In *Proc. CVPR*, pages 2011–2018, 2013.

[41] D. Rumelhart, G. Hinton, , and R. Williams. Learning representations by back-propagating errors. *Nature*, 1986.

[42] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing*, 1:318–362, 1986.

[43] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *Proc. ECCV*, pages 746–760, 2012.

[44] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[45] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proc. CVPR*, pages 1701–1708, 2014.

[46] Z. Tu. Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering. In *Proc. ICCV*, volume 2, pages 1589–1596. IEEE, 2005.

[47] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[48] C. L. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. In *Proc. ECCV*, pages 391–405, 2014.