Edge Detection Algorithms with Machine Learning

Yu-Hsiang Tseng
Electrical and Computer Engineering
Purdue University, Indiana 47907

Abstract

Edge detection is a vital part in most of computer vision algorithms, including image quality enhancement and object segmentation. Instead of using the conventional bottom-up approach, three algorithms which are based on the combination of bottom-up and top-down approaches are introduced here. State-of-the-art learning base edge detection algorithms require lots of training data, and the three algorithms used different ways to extract feature and built their own training environment. In this paper, the introduced algorithms were compared and critique. In addition, the decision forests algorithm has also been implemented in this paper. The details of implementation and experimental results were provided.

Introduction

Edge detection is an old image processing technique, and bottom-up solution, such as SIFT and SURF, is commonly used in digital image processing and computer vision. Edge detection also plays a vital role in a lot of computer vision algorithms and applications, such as image quality enhancement and autofocus algorithms. However, the bottom-up approach has a major drawback which is a set of parameters (for example, α in Gaussian blur kernel or Hessian matrix used in SURF) are necessary to be decided, and to be applied to the whole image. These parameters are the thresholds for the strength of edges. Users can only decide whether they need are strong edges or weak edges but not the contour of foreground objects. Besides, noise may break the edges of objects which can lead to an incomplete connected object contours. On top of that, zero-crossing method produces double edges on each edge, or the edges on the map may not lie on the perfect places.

A series of approaches which amalgamates top-down and bottom-up approaches were published recently. These of algorithms can be evolved through machine learning methods. Based on the provided training dataset, algorithms gradually realize the users' demands and auto-updates on the algorithms are feasible. Also, the double edges problem can be solved because machine-learning algorithms search the best solution form the provided database without creating a new database. Feature extraction and comparison functions are the core of machine learning algorithms. They need to know which ones are similar, and which ones should have higher scores when they want to choose the best ones.

The three papers selected in this paper utilized three distinct approaches to extract and compare features. Lim et al (Lim, Zitnick, and Dollár 2013) used sketch tokens to construct a random forest classifier. Then, they matched the query patches of the images with the classifier, and decide whether the patches contain object contours or not. Contours on the hand-made tokens include some well-known patterns, such as straight lines, parallel lines, T-junctions, or Y-junctions.

Based on the work performed by Lim et al., Dollár et al. (Dollár and Zitnick 2015) proposed a generalized structured learning approach instead of using hand-made tokens. This approach exploits the advantage of inherent structures in edge patches, and it is considerably computationally efficient. Also, they built a different set of learning forests that uses structured labels to determine the splitting function at each branch in the tree. In addition, the authors also provided a novel forest training algorithm to build the random forest classifier, instead of the predefined classes of edge patches. Each leaf node of the forests predicts a patch of edge pixel labels, and the labels are combined across the image to obtain the final edge map. Dollár et al. also claimed that additional depth cues might be added into their feature extraction and mapping function.

Banica et al. (Banica and Sminchisescu 2015) reported an algorithm which uses RGB-D domains to perform the semantic segmentation. They set points at different spatial locations, and these fixated locations were assigned to be foreground. And the elements on the boundaries of the image were backgrounds. Then the binary partitions are solved on the images, and these binary partitions would be the edges of the objects. They combined the cost of local features and the effect of neighbor to be the new cost function in order to minimize the cost function. The features include SIFT, spin images, and Local Binary Patterns. Also, Banica et al. tried to build a 3D bounding box with a 44-dimensional descriptor. With these algorithms, they were also able to solve overlapping object-level problems. Like others, they also designed their own on-line training algorithm to update their cost model and segmentation results.

Methods and Critique

Sketch Tokens (Lim, Zitnick, and Dollár 2013)

Token Definition Human subjective training was used to select hand-draw binary contour sketches by asking testers which lines can be used to divide images into pieces. Patches centered on contours from the sketches were extracted and clustered to form a set of token classes. Approximately two million patches were created and used on the algorithm training. Daisy descriptors (Winder, Hua, and Brown 2009) (Tola, Lepetit, and Fua 2008) were then computed on the binary contour labels, and K-means algorithm $(k = 150)$ was used to cluster the descriptors. Then, the features was extracted, and they were used to compare with query patches. The patches were then classified as labels, background, or no contour.

The major problem in this approach is that it is nearly impossible to allow the self-training of the system when different shapes of patches are queried. Lacking one or more kinds of contour shapes, the output edge-map can only be composed by other shapes, and the error will be large. That is, the output edge-map can only lie on the space spanned by the tokens. If the query patches are linearly independent from the tokens, the error space will be orthogonal to the token space, and the error can be largely depending on the angle between query patches space and token space. Therefore, the more comprehensive the token space is, the better the results are. About 2 million patches were used in Lim′s work.

Detecting Sketch Tokens SIFT-like feature extraction methods were used here. CIE-LUV color space, orientation, and scale with Gaussian blurs $(\sigma = 0, 1.5, 5)$, and post-blurred with =1 Gaussian were applied on Gaussian blurred images. There are 14 channels in a 35×35 patch, therefore an about 17,150 dimensional feature vector was generated.

In addition, Lim et al. introduced self-similarity. The self-similarity can be used to overcome the problem described. Because the contours occur not only the color edges but texture boundaries, self-similarity is used to compare whether the current patch contains similar texture with neighborhood or not. These features capture other portion of patches, which contains similar textures based on color or gradient information, in the query image. The image may contain a lot of contour shapes that do not lie on the token space. But if their texture are different from neighbors', they can still be marked as edges. There are 4,200 self-similarity features used, so the feature vector becomes 21,350 dimensions. This may alleviate the error problem mentioned above, because self-similarity compares portions in the same image. However, if the textures are too complicated, the error is still unneglible.

$m \times m$ grids were applied on the 35×35 patches, and self-similarity feature were computed by the L1 distance of texture information between each pair of grids. Here, smaller block size was used, so that the details of the images will not be neglected. In the self-similarity feature formula, not only the vector norms but also the positions of grids are considered. In other words, if a small object changes its position in one 35×35 patch, the self-similarity feature would be different.

It will be a burden task to make comparison using such large feature vectors. Therefore, Lim et al. randomly select some of the possible features, and use Gini impurity to measure and select the feature set. A random forest consisted by decision trees are trained by 31,000 patches which were randomly selected from the tokens. When a query patch was inputted, the probability t_{i0} of belonging to no contour class is computed, and $(1 - t_{i0})$ is the estimated probability of the patchs center containing a contour. However, there could be a way to obtain the basis of the token space first in order to minimize the feature vector space. For example, the token patches can be normalized by the orientation, so that keeping a comprehensive orientation of tokens is unnecessarily. Because random selection is used here, there must be some redundant feature elements in the feature vector space.

Structured Decision Forests (Dollár and Zitnick 2015)

Training Decision Trees The split function $h(x, \theta_j)$ is trained independently for every tree, so that they can split the data to different categories (left nodes and right nodes here) at a node of a tree. Dollár applied Shannon entropy and Gini impurity in their standard definition of information gain. The author stated that only one crucial component of each node is sufficient to get an adequate diversity of trees, as the basis I mentioned earlier. After his experiments and researches, he thought injecting randomness at the level of nodes would give better models. Therefore, as sketch tokens, partial features are selected here. Yet, I think that how to choose features may be a problem here. The decision tree always chooses the ones in highest score. What if the second one is also good, but their score is not high enough in the training set. However, is may be the most important one in query images. We cannot select a perfect set of training database.

It is not easy to compare the similarity of two segmentation map patches y_i, so Dollár et al. mapped the the patches to a long high-dimensional binary vectors space, $\mathbf{Z}$, which encodes whether every pair of pixels in 16×16 segmentation masks belongs to the same segment or not. It is easier to measure the distance. However Dollár et al. only partially sampled dimensions of $\mathbf{Z}$ because of the computational complexity. Principal component analysis is used to reduce the dimension of $\mathbf{Z}$. PCA is a tool to obtain the basis of the feature space, so this algorithm helps to keep the essential part of the feature space. Finally, At a leaf node, multiple labels are combined by selecting a z_n and labeling as y_n that minimizes the sum of L2 distances to all other z_i, where $z_i \in \mathbf{Z}$.

The biggest limitation is that any prediction here must have been observed during training, that is, must lies in the training set plane. This problem would be same as Sketch Tokens.

Feature Extraction Instead of Using RGB or RGB-D images with 32×32 patches, Dollár et al. used same feature extraction method as Lim et al.. They applied gradient magnitude at the original and half resolution images, and kept 4 orientation channels for each resolution. So there were total of 3 colors, 2 magnitudes, and 8 orientation channels. Triangle filters and down-sampling were also used to reduce the dimension of the feature vectors, and the pairwise difference for self-similarity. The dimension of feature vector was only 7,228. The major concern on their approach is that the algorithm only keeps 4 orientation channels, while $\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, and \frac{7\pi}{4}$ were discarded while these seems to be important in Figure 4 in, Lims work. However, Dollár used a bilinear-interpolation-like way to approach it. A further discussion is shown below in the experiment result (Experiments).

Edge Detection Dollár defined a mapping function $\mathbf{Y} \Rightarrow \mathbf{Z}$ that checks whether two points $y(j_1)$ and $y(j_2)$ belongs to the same segment. That is, to check if $y(j_1) == y(j_2), \forall j_1 \neq j_2$, and $y \in \mathbf{Y}$ is the output segmentation masks. If they are not the same, there is a boundary between $y(j_1)$ and $y(j_2)$. Multiple overlapping edge maps will be averaged to yield soft edge responses. If merging masks is difficult, the averaged ones would also be stored at leaf nodes.

Three sizes of edge map will be generated simultaneously from 3 different resolutions. This is a good idea when the resolution of query image and the one in the forest are significantly different. However, a large resolution image always keeps more details, that is, more edges would be kept. Which should be preserved, details or contours of large objects, would always be a problem in edge detection algorithm.

A new edge sharpening technique is provided here. In order to put edges onto the best places, it is necessary to morph the predicted images to match the original ones. This is because the predicted edges may not be at the right positions as they are generated by the combination of training set. Dollár moved the segmentation masks at most a pixel to minimize the L2 distance between the image patch corresponding objects segmentation masks. Other than using low-pass filter to discard weak edges, this method also makes the edge map cleaner.

Semantic Segmentation in RGB-D Images (Banica and Sminchisescu 2015)

Parametric Generation First, set up constraints like setting which locations are foreground, and the rim of the picture would be the back-ground. Secondly, spatial energies form in the paper is used to obtain the labeling of the rest of the image. The concept here is trying to find a graph cut with a minimum cost. However, it is difficult to decide the relationship between the local cost and the penalty, because a tiny noise may give rise to a high value on the second term. The authors then attempted to use different spatial scales of the images to solve the tiny noise problem. Yet, in their experimental results, small objects still cause some errors. For the cost function, feature distribution of foreground and background are used to predict the probability of the query pixel belong to foreground (Carreira and Sminchisescu 2012). In addition, the foreground representative pixels are selected as centers resulting from a k-means algorithm.

This algorithm is distinctly different from the previous ones, because the users need to fixate some foreground pixels. Nevertheless, it is possible to set nearest point in the depth map as an initial foreground, and farthest point as an initial background. But using this method, only one object can be segmented at one time. The experiments mentioned in the paper used attention models to set the constraint. In this case, human perception model may be more suitable to be used at the initial state because human has tendency to pay attention to foreground objects, but the human perception models are not perfect nowadays. The output will be a ranked list for object plausibility, and only top K ($K = 500$) are kept here.

Local features were combined by second-order average-pooling matrix, and they are the average of square of element of descriptors in each dimension. Banica et al. also used Schur-Parlett algorithm and principal component analysis to reduce the dimension of the local feature matrices according to eigenvalues, and this is also the basis of the feature space which I described previously.

Feature Extraction Banica also used SIFT and other common descriptors, but he applied not only on the color domain, but also the depth. Spin images were also used for the 3-D models in order to describe the polygonal surfaces. These features were not adopted by Lim et and Dollár et work. Because of the depth images, 3D point features are feasible here. After discarding outliers, the objects can then be fitted into 3D bounding boxes. As shown in Figure 2 in their paper, without 3D point features, similar color or texture with different depth objects cannot be separated easily while the 3D point features are able to separate the bar chairs from the bar table pretty well. Therefore, it is rational to combine these two feature vectors in the analysis. However, the algorithm may still fail to separate the table and the chairs when these objects are overlap. Besides, the column 2 in Figure 2 here shows that the boundaries extracted from the intensity image are pretty good, only the contours of objects are shown, that is, the second term in their spatial energies form would be a dominant part.

Confidence Models For model training, the linear regression models (Carreira and Sminchisescu 2012) were trained to predict the best-matching objects of each class to obtain category predictors, and the data were composed of the features from the K masks of test images as described in Parametric Generation. However, the training steps were used for identifying objects, as shown in Figure 4 in the paper, and I think it is not necessary to train when we want to use the algorithm as an edge detector, as shown in Figure 2 in the paper.

For testing, the algorithm executed all category predictors on the K retained masks, and chooses the maximal estimated overlap one as the label. However, problems may occur when the algorithm merely adds these K masks together. If there are multiple foreground objects, we will obtain multiple regions for them, but the regions may be overlapped. Sequential Search-Based Structured Prediction was used by the authors to select the actions that most suitable given current state, and all regions were then combined together.

This algorithm is much different from others, because it separates objects one by one by using object-based operations while the other two papers only take the probability to be a contour into consideration. However, this algorithm tries to not only get the contour, but also segment objects. It adds objects one by one onto the segmentation plane, as shown in Figure 3 in the paper. The Sequential Search-Based Inference here is that, first, they start from a partial segmentation status. Secondly, it takes an action (add object labels or stop action) and gets the next partial segmentation status. Then, it repeats again and again until it gets an action-stop.

Implementation

In this section, I described my Python implementation of the edge detection method in decision forests (Dollár and Zitnick 2015). The test images and corresponding ground truth are from BSDS500 (Arbelaez, Fowlkes, and Martin 2007). Dollár et al. provided their trained forest in the form of mat files, so I implemented the design with the trained forest.

Feature Extraction

As shown in the paper, Dollár et al. used CIE-LUV color spaces with normalized gradient magnitude at two scales. L, U, and V channels in half solution $(W/2 \times H/2)$ are stored in the first three channels. I then applied gradient on the three color channels in original size, but for every pixel I only chose one with the maximum absolute gradient value from the three color channels.

After that, a magnitude image $(W \times H)$, and an orientation image can be generated. To keep every channel in a same size, the magnitude image is down-sampled as $(W/2 \times H/2)$ and is stored in channel 4. The histogram of orientation can then be acquired. I chose bin size as 2, and only 4 orientations $(0, \pi/2, \pi, 3\pi/2)$. If a pixel (i, j) with a orientation o, $((\pi/2) \times k < o < (\pi/2) \times (k+1))$ and the magnitude is m, I added $(m \times ((o - (\pi/2) \times k))/(\pi/2))$ in Histogram$(i/binSize, j/binSize, k)$, and added $(m \times (((\pi/2)(k+1) - o))/(\pi/2))$ into Histogram$(i/binSize, j/binSize, k+1)$. A histogram map with size of $(W/2 \times H/2)$ can then be acquired, and stored in channel 5 to 8.

Down-scale the image to $(W/2 \times H/2)$, and do the same gradient and histogram functions again. However, it is not necessary to down-sample the magnitude image, and the bin size is 1. And again, I store them in channel 9 to 13. Therefore, there are 13 $(W/2 \times H/2)$ feature channels.

As mentioned in the paper, to calculate self-similarity, I have added another feature channels with blurring the original feature channels with a radius-8 triangle filter.

Tree Tracking

First, divide the images into multiple disjointed 16×16 patches. Then every 16×16 patch will be divided to be 25 sub-patches for self-similarity. The self-similarity difference will then be conducted as the difference of 13 feature channels between query pixel and corresponding pixel in every other 24 sub-patches. As the paper demonstrated, there are 300 $(c_2^{5\times5})$ candidates features per feature channel.

Then, the algorithm starts to trace the tree nodes. There are eight trees in the forest, but every pixel only uses four trees while odd pixels and even pixels use four different trees. There are about 80,000 tree nodes in one tree. The information in each node contains the index of its children (every node has two or zero children), and if the feature value is larger than the threshold stored in the node, the algorithm will traverse its right child, and if not, traverses its left child. For the feature value, every node stores a feature ID to indicate whether this node uses self-similarity or not, and how to get the feature value with the threshold. It chooses one value as the feature value from 13 channels of pixels in a 16×16 patch of which the current pixel is top left corner.

For self-similarity, the node acquires the difference of two (chosen by nodes) pixels which are in different sub-patches, from the feature vector. If the difference is larger than the threshold, the algorithm will take the right child, and the difference is the feature value. For the channel feature, the node takes only one specific value in the feature vector to be the feature value. Recursively, every pixel may trace its four trees to get four leaf nodes.

Edge Acquirement

Each leaf node has one 16×16 segmentation map which is trained by a training set with ground truth, and each map may contain one or more segments. The algorithm used here is trying to find the edge between segments and fuse all partial edge maps to be a completed edge map.

For each pixel, we can obtain a 16×16 segmentation map S, and the map contains m segments. So, as mentioned in the paper, for segmentation ID = 1 to m, the average pixel values $\mu_{(seg\_id)}$ are computed separately in three colors LUV. For example, there are two pixels A and B marked as 1, and three pixels C, D, and E marked as 2 in this map. We can get the average pixel value of A and B, and of C, D, and E, in L, U, and V separately. To sharpen the edges, the algorithm will try to $min||\mu_{(seg\_id)} - I(x, y)||_2$, and $S(x, y) = seg\_id$. To do so, the algorithm changes every non zero seg_id pixel in the segmentation map into its 4-connected neighborhoods seg_id, and checks the L2 distance value is smaller or not. If the value gets smaller, keep the change and update the segmentation map, and if not, keep the original value. After we update the segmentation map, every pixel which has different seg_id 4-connected neighborhoods will be an edge, and we increase the edge value on it. Then, we can move to the next pixel, get another 16×16 segmentation map, and repeat the same procedure until the complete edge map is generated.

The final step is to normalize the edge map. Since the maximum value is not a constant, we need to obtain and a maximum and a minimum value, and normalize it to a range of 0 to 255. A small filter can then applied on it to smoothen the image.

Improvement & Applications

Edge detection algorithms can be applied in many algorithms. Here I have applied the decision forests algorithm to a simple image edge enhancement. After the edge map

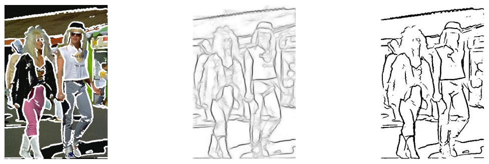

(a) Image with edge map (b) Original Edge map (c) Edge map with threshold

Figure 1: Discontinuations might be found in the contours of the foreground objects. In (a), white line shows the edges on the edge map, and they did not consist perfect contours. (b) shows the original edge map, and (c) shows if we apply a threshold for it to be a binary image.

was normalized to $[0, 255]$, the gradient of the image should be re-computed at the edge points. The main purpose of this algorithm is to enlarge the gradient at edge points. At each point $I(i, j)$ at each color channel, the four-neighborhood points will be updated like $I(i+1, j) += (I(i+1, j) - I(i-1, j)) \times \hat{E}(i, j)/512$, $I(i-1, j) += (I(i-1, j) - I(i+1, j)) \times \hat{E}(i, j)/512$, $I(i, j-1) += (I(i, j-1) - I(i, j+1)) \times \hat{E}(i, j)/512$, and $I(i, j+1) += (I(i, j+1) - I(i, j-1)) \times \hat{E}(i, j)/512$ where the $\hat{E}(i, j)$ is the normalized edge map.

However, there were two problems in the decision forests algorithm. Firstly, discontinuations were found in the contours of the foreground objects. Therefore, it could not segment objects out perfectly. Secondly, a threshold should be set to discard the noise, shows in Fig.1.

We can also the apply edge enhancement and segmentation algorithms to video codec. For example, rate control plays an essential role in H.264 or HEVC, because it controls the trade-off between bit rate and video quality. A large number of works tried to increase QPs on background or places where the distortions are less perceptual in order to save bits. Therefore, segmentation can be a suitable way to reduce bit rate. Furthermore, we can apply not only my edge enhancement algorithm to the foreground objects, but also edge burring to the background, in order obtain a synthesized short depth of field image.

Experiments

I have selected several images with different objects (humans, vehicles, buildings, and nature view) from BSDS 500, and the threshold was one-third of the dynamic range. The algorithm tested was functioned well even without rebuilding forests. This indicating the forests are comprehensive and the algorithm is robust. Then I applied my edge enhancement algorithm on these images. If the images were too complicated, the decision forests algorithm might fail to extract strong edges. For example, as shown in (Fig.2). The image is too complicated, hence it is almost impossible for the algorithm to separate the pumas and we can only get a messy edge map. If a threshold applied is to the image to remove weak edges, there would have been nothing left (Fig.2b). However, the edge enhancement algorithm still worked (Fig.2c). I scaled up 4 times of the original image and edge enhancement ones, (Fig.2e). The detail of the image was enhanced along with some noise, but the image became sharper.

The decision forests algorithm worked great even the colors in the image were similar. As Fig.3 shows, the image is full of khaki color, but the edge map is able to separate the fish and the stones perfectly. Therefore, if the texture in the image is simple, the algorithm works great.

Another example is shown in Fig.4. Even thought the giraffe is the only object in the image that has strong edges, discontinuations can still be found on its neck and legs. On the other hand, the spots appear as weak edges in the edge map This is the effect that we are looking for, because we wanted to separate the foreground, not the spots. Therefore, after the application of the edge enhancement algorithm, the giraffe became much more evident.

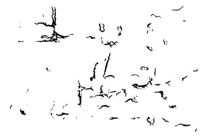
Edge enhancement can also be applied to enhance the quality of up-scaling images. When bilinear algorithm is used to interpolate and up-sample images, the images will be blurry. At this time, a good edge enhancement along with a good edge detection algorithm can help to sharpen the images. As shown in Fig.5, the image was processed by a fourfold scaling-up. The edge enhancement algorithm helped to enhance the image quality. Besides that, the textures of wooden walls and floors were not marked as strong edges, although they are salient in the picture. Only borders of objects were cut out, including the mountains. The mountain shares similar color with the blue sky and sea of clouds. However, the mountain on the right hand side was not cut out, although the ridgeline is clear in human eyes. Because the texture of the ridgeline is complicated and weak, there was no peak value in the histogram of gradient. Accordingly, no good feature could be used to describe it.

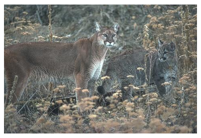
There were only four orientation regions used in the histogram of gradient in the decision forests algorithm. But Dollár used a bilinear-interpolation-like way to handle this. We could find some images whose textures or edges were not in 0°, 45°, 90°, 135°,or 180°. As shown in Fig.6a, 6b, 6c, the mask on his face contains obvious edges, but the edges barely exist on the edge map. We could find out that almost every orientation of the edges on the edge map was one of these five directions. The make-up on his hands is same as his hand with different texture, but on the edge map, we could only see the edges on the hands. Another example shown in Fig.6d, 6e, 6f, the folding on the mushrooms has strong edges in the picture, but the algorithm could not detect them well. This is because the edges are basically round and have multiple orientations. The algorithm experience difficulties in detecting intricate edges. Or the patch size is so large that the algorithm seemed these edges as textures not edges.

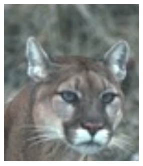
Conclusion

Although machine learning based edge detection or segmentation algorithms are powerful, they still need large sets of test patterns, and humans are usually required to construct some golden truth or initial start points. It takes considerable

(a) Original image


(b) Edge map with threshold


(c) Enhanced image


(d) Zoom-in original image

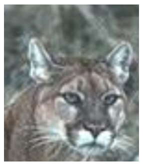
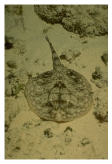
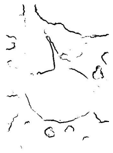

(e) Zoom-in enhanced image

Figure 2: After applying the proposed edge enhancement algorithm, edges are sharper. However, some noise in the original edge map causes some noise in the edge enhance image.


(a) Original image


(b) Edge map with threshold

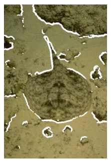
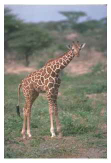
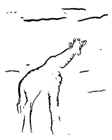

(c) Add edge map to the original image

Figure 3: Edge map is combined with the original image for illustration. The white thick lines are the edges on the edge map.


(a) Original image


(b) Edge map with threshold

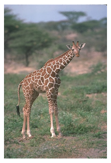
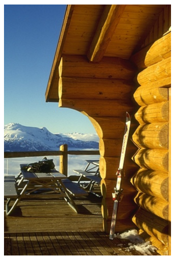

(c) Edge enhancement image

Figure 4: Sharper image is produced by my edge enhancement algorithm.


(a) Original image

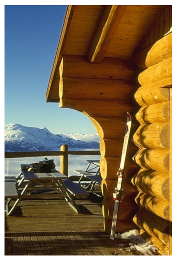
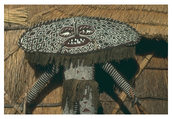
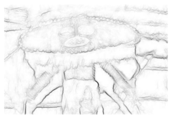
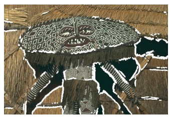

(b) Edge enhanced

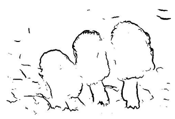
Figure 5: Applying the proposed edge enhancement algorithm on a up-scaled image helps to make image sharper and clearer.


(a) Original image


(b) Edge map


(c) Add edge map to the original image

(d) Original image


(e) Edge map with threshold

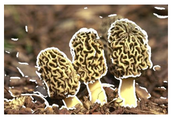

(f) Add edge map to the original image

Figure 6: Edge detection failed on multiple-orientation edges.

amount time and effort to handle this. However, the training sets need to be built only once, and algorithms may train themselves after that, if the training sets and algorithms are robust. Three algorithms reported in Lim et al., Dollár et al., Banica et al. utilized different training sets, and feature extraction methods. Nevertheless, SIFT and gradient are one of the best approaches in feature ex-traction and comparison, because all of them use modified SIFT to be one of their main features. Also, to reduce computational complexity, they used PCA or random selection to reduce the dimension of feature vectors.

References

Arbelaez, P.; Fowlkes, C.; and Martin, D. 2007. The berkeley segmentation dataset and benchmark.

Banica, D., and Sminchisescu, C. 2015. Second-order constrained parametric proposals and sequential search-based structured prediction for semantic segmentation in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3517–3526.

Carreira, J., and Sminchisescu, C. 2012. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34(7):1312–1328.

Dollár, P., and Zitnick, C. L. 2015. Fast edge detection using structured forests. *IEEE transactions on pattern analysis and machine intelligence* 37(8):1558–1570.

Lim, J. J.; Zitnick, C. L.; and Dollár, P. 2013. Sketch tokens: A learned mid-level representation for contour and object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3158–3165.

Tola, E.; Lepetit, V.; and Fua, P. 2008. A fast local descriptor for dense matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 1–8. IEEE.

Winder, S.; Hua, G.; and Brown, M. 2009. Picking the best daisy. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 178–185. IEEE.