Abstract

Edge detection is utilized in a variety of computer vision applications, yet it remains a challenging problem on its own. Boosting has shown impressive performance in training offline classifiers for detection tasks. In this paper we propose the use of an online supervised learning algorithm for edge detection. The algorithm trains incrementally as new data becomes available, which has several advantages over offline methods, and opens the possibility for new applications. Our method combines a large number of features across different scales in order to learn a discriminative model using an online boosting classification framework. The resulting edge detector is adaptive with no parameters to tune. We test our algorithm on images from two different domains and demonstrate promising results with a relatively small number of training examples.

1. Introduction

Edge detection is a fundamental operation in computer vision. It is used in tasks such as object detection and recognition, as well as segmentation. A general edge detection method which can be applied to a wide range of image types with good performance is still an open research problem. Methods which rely on local gradients alone, such as the Canny [1] detector, require threshold tuning to select the edges of interest. In highly textured or noisy images, it can be impossible to achieve desirable results. Approaches which combine several local cues (texture, brightness, color) produce improved results on general edge detection. The application of learning algorithms to the edge detection problem has been demonstrated to be effective, especially in the case where labeled ground truth data can be obtained.

Boosting has been successfully applied to numerous machine learning and computer vision problems, including text recognition, natural language processing, and more recently, edge detection [2]. The majority of boosting based approaches work in an offline manner, or in a batch processing mode (collecting a set of training examples and then running an offline boosting algorithm).

Offline boosting has limited usefulness in tasks where enough training data cannot be a priori given, or which require adjustment to handle variations in the input. Object tracking is one such application which requires adaptive techniques for adjusting to possible variations of the target object over time. Recently, online boosting has been applied to the object tracking problem with successful results.

In this paper we apply an online boosting based feature selection framework in the context of edge detection. As a result, we can potentially handle real-time applications of edge detection previously not possible using offline edge learning methods.

2. Related Work

Our work is inspired by specific recent work in online boosting and offline learning based edge detection.

2.1. Boosting

The work of Viola and Jones [9] applied boosting to object detection and popularized it as a learning method in the area of computer vision. In this paper we focus on an online version of a popular offline boosting algorithm, discrete AdaBoost [3]. Although online versions of other learning algorithms exist, based on the results in [4], we decided to base our framework on their novel online formulation of AdaBoost.

Oza [6] proposed the notion of estimating the importance, or difficulty of a training sample by propagating it through a set of weak classifiers. He also proved that if offline and online boosting are given the same training set, then the weak classifiers (Naive Bayes classifiers) returned by online boosting converge statistically to the one obtained by offline boosting as the number of iterations approach infinity.
2.2. Edge Detection

Data driven algorithms such as Pb, proposed by [5] have some learning components, and demonstrated improved results on natural images compared to the Canny edge detector [1]. However, the BEL algorithm [2] relies entirely on learning to perform edge detection, and shows even better performance and versatility in multiple image domains. Dollar et al. report that a simple cascade of AdaBoost classifiers did not yield sufficiently good results (high error), even with a large number of bootstrapping stages. To improve performance, they chose to use an extension of the probabilistic boosting tree (PBT) proposed in [8].

The probabilistic boosting tree is similar to a decision tree, except that at each node, a boosted classifier is used to split the data. We choose to use a simple boosting framework to be explained following section, in order to simply demonstrate the online boosting based edge learning concept. However, extending the PBT to online training remains as a potential avenue for improving upon our work.

3. Background

We first review offline boosting and then proceed to show the online boosting approach.

3.1. Offline Boosting

Boosting is the general method for improving the accuracy of a learning algorithm by combining a weighted combination of $N$ hypothesis generated through repeated training with different subsets of the training data. Boosting transforms a combination of weak learning algorithms into a strong learning algorithm.

A weak classifier is one which performs only slightly better than random guessing. For a binary decision task, this is equivalent to an error rate of less than 0.5. The weak hypothesis $h_{weak}$ is generated by applying some learning algorithm.

Given a set of $N$ weak classifiers, a strong classifier is computed as a linear combination of weak classifiers.

$$h_{strong}(x) = \text{sign} \left( \sum_{1}^{N} \alpha_n \cdot h_{n}^{weak}(x) \right) \quad (1)$$

For the AdaBoost algorithm [3], the weight $\alpha_n$ for each weak classifier $h_{n}^{weak}$ is assigned based on the error $e_n$, computed from its performance on a weighted version of the training data (initialized with uniform weights). The weight assignment is such that if a training example is classified correctly, its corresponding weight is decreased for the next iteration. Likewise, if the training example is classified incorrectly, its weight is increased. As a result, the algorithm pays more attention in future rounds to the examples misclassified by previous rounds. Every iteration results in the selection of one weak hypothesis, and at the end of $N$ rounds of boosting, $N$ weak hypotheses are combined to form the strong hypothesis.

AdaBoost is a greedy optimization algorithm. It has been extensively studied, and for the case of binary classification, the training error has been shown to drop exponentially fast with respect to the number of boosting rounds. Tieu and Viola [7] introduced boosting methods for feature selection in image retrieval. They proposed the notion that each feature corresponds to a single weak classifier, and boosting can select the best features from a large pool of possible features.

3.2. Online Boosting

Online boosting is defined as an algorithm which operates on a single example and discards it after updating. Each of the steps performed by offline boosting must be performed, but in an online fashion. The online updating of weak classifiers is not a hurdle, since many online learning algorithms can be used to generate a weak hypothesis. Estimating the final voting weights $\alpha_n$ is straightforward since we can calculate the estimated error of the weak classifiers. The critical step for online boosting is computing the weight distribution for the samples. In offline AdaBoost, we are afforded the luxury of having the entire training set at each iteration. This simplifies the process of estimating the difficulty of any given sample, which is necessary for sample reweighting. In online boosting, we must attempt to estimate the difficulty of a training sample without knowing if we have seen the sample before.

Oza proposed the idea that the importance of a sample can be estimated by propagating it through a set of weak classifiers. According to this idea, every incoming sample is initialized with an importance weight $\lambda = 1$ and passed to one weak classifier. If the first weak classifier correctly predicts the label of the training sample, $\lambda$ is reduced in a fashion similar to that of offline boosting. Similarly, if the weak classifier incorrectly predicts the label, $\lambda$ is increased. The same training sample is then presented to a second weak classifier, along with its estimated importance weight from the previous weak classifier into consideration. This process continues in a similar fashion as the sample propagates through the classifiers. In essence, the first classifiers treat each sample with equal importance. Classifiers further down the chain, however, have gained some information about the difficulty of the sample based on the performance of classifiers preceding them in the pipeline.

Grabner et al. [4] take this concept and apply it to feature selection by introducing the concept of the selector. A selector is essentially an encapsulation of a set of $M$ weak classifiers. At any given time, the selector chooses the single weak classifier with the lowest estimated error. Similar to the offline case, weak classifiers correspond to features, and the selector chooses the best feature from a
subset of \( M \) features out of the total global feature pool. The idea is to apply on line boosting not directly to the weak classifiers, but to the selectors. Fig. 1 depicts the overall concept.

**Input:** training example \( x \), label \( y \), weights \( \lambda_{\text{correct}}^{n,m}, \lambda_{\text{wrong}}^{n,m} \)

**Output:** strong classifier \( h_{\text{strong}} \)

//for all selectors;

for \( n = 1, 2, \ldots, N \) do

//update the \( n \)th selector;

for \( m = 1, 2, \ldots, M \) do

//update each weak classifier;

\( h_{\text{weak}}^{n,m} = \text{onlineWeakLearn}(h_{\text{weak}}^{n,m}, x, y, \lambda) \);

//estimate errors;

if \( (h_{\text{weak}}^{n,m} = y) \) then

\( \lambda_{\text{correct}}^{n,m} = \lambda_{\text{correct}}^{n,m} + \lambda \);

else

\( \lambda_{\text{wrong}}^{n,m} = \lambda_{\text{wrong}}^{n,m} + \lambda \);

end

\( e_{n,m} = \frac{\lambda_{\text{wrong}}^{n,m}}{\lambda_{\text{correct}}^{n,m} + \lambda_{\text{wrong}}^{n,m}} \);

end

//selector picks best weak classifier;

\( m_{\text{best}} = \text{chooseBestWeakLearner}(n) \);

\( e_n = e_{n,m_{\text{best}}} \);

\( h_n^{\text{selector}} = h_{n,m_{\text{best}}} \);

//calculate voting weight;

\( \alpha_n = \frac{1}{2} \cdot \ln \left( 1 - e_n \right) \);

//update importance weight;

if \( (h_n^{\text{selector}} = y) \) then

\( \lambda = \lambda \cdot \frac{1}{2 \cdot e_n} \);

else

\( \lambda = \lambda \cdot \frac{1}{2 \cdot e_n} \);

end

//selector replaces worst weak classifier;

//with a random one from the global pool;

\( m_{\text{worst}} = \text{chooseWorstWeakLearner}(n) \);

\( m_{\text{random}} = \text{newWeakLearner}() \);

\( \text{replaceWorstLearner}(m_{\text{worst}}, m_{\text{random}}) \);

end

Algorithm 1: Online AdaBoost for feature selection

### 4. Classification Framework

Given a training image and a sequence of pixel coordinates which correspond to labeled edges, we sample positive example patches over a window centered on the labeled edge pixels. We use a large pool of Haar-like features at multiple locations, scales, and aspect ratios, calculated over a range of window sizes from 13x13 to 25x25. This strategy was employed to help the classifier to implicitly handle various kinds of edges over a variety of different scales. As a commonly used optimization, we compute integral images [9] so that our filter responses can be computed more efficiently. We compute filter responses on a per patch basis, however we acknowledge that this can be further optimized for the cases when adjacent patches must be computed by only computing filter responses once per image at a given window size.

Negative example patches can be provided in two ways. One option is to provide the algorithm with a sequence of pixel coordinates which correspond to non-edges. By default, our strategy is to randomly sample patches throughout the image. Unless the image contains a large percentage of edge pixels, there will only be a small probability that a random patch will be centered exactly on an edge pixel. However, if this occurs, it can pollute the priors built by the weak learners. Theoretically, having labeled negative examples would be more informative, but requires extra effort to obtain. Since we can foresee situations where user directed training of the definition of non edges may be useful, we have incorporated this feature as an option.

Our weak learners are simple Bayesian classifiers which maintain two 128 bin histograms each, with equally sized bins, ranging from the minimum to the maximum expected response values from their associated Haar-like feature. By incrementing the appropriate bin in the positive or negative histogram for each training example, each weak classifier builds a discrete approximation of the prior probabilities for positive and negative filter responses. When presented with the filter response for a novel image patch, the weak learner simply compares the count for the appropriate bin in its positive and negative histograms, and returns the label with the highest probability. In the event that there are zero entries in both histogram bins or a tie, the distance of the input example’s response to the mean of the positive and negative
histograms is used to determine the returned classification label.

5. Experimental Method

We envisioned this algorithm for use in interactive applications, where the training data is provided continuously by the user’s mouse movements, (e.g. tracing an edge). However, in order to provide some degree of repeatability in the results, we simulate this environment. We load the labeled ground truth image for a training image, select a predefined number of edge pixels in a scanline fashion from the label image, and feed the coordinates into the learning algorithm sequentially.

To simplify training and testing, we train on a small fraction of the ground truth data and use the resulting trained classifier to test on the entire training image. This also serves to show how quickly the algorithm begins to adapt to the image with relatively small numbers of training samples.

We train and test on natural images from the Berkeley Segmentation Dataset, and an image from an electron tomography dataset provided by the National Center for Microscopy and Imaging Research. The input edge maps from the Berkeley Segmentation Dataset were thresholded and binarized to keep only the strongest edges for training. The labeled edge data from the NCMIR dataset is already a binary edge map since it is the ground truth produced from a single expert.

We compute the precision, recall, and F-measure for each output edge map. Precision is the probability that a machine generated boundary pixel is a true boundary pixel. Recall is the probability that a true boundary pixel is detected. The F-measure distills the performance of the algorithm into a single number, and is equivalent to the harmonic mean of precision and recall. In the literature, the Berkeley Segmentation Dataset boundary maps are usually thresholded at 30 different levels to produce precision and recall curves, from which the maximum F-measure value is calculated. We only use a single threshold and thus only compute and report one point from the precision-recall curves, which may not be the maximum for our algorithm. For a given image, the threshold value we choose is the one at which the BEL algorithm achieved its highest F-measure.

Our global feature pool was limited to 7200 features for testing. We evaluated various combinations for \( N \) (number of selectors) and \( M \) (number of features per selector) but fixed \( N = 60 \) and \( M = 120 \) for testing. All images shown in the results section were generated using this configuration. We tested each image with the first 250 positive samples from the input edge map. The total number of training samples the algorithm sees is actually 500 due to the fact that we automatically sample one negative image patch at a random image location after every positive sample is processed.

6. Results and Discussion

Figures 1, 2, and 3 show our performance on three images from the Berkeley Segmentation Dataset. For each of these images, 250 labeled ground truth points were used for training, which is always under 25 percent of the total number of edge pixels in the image. Visually, the training set roughly corresponds to the left most quarter of the ground truth image, while the testing occurs on the entire original image.

We saw our best performance on the elephant image. In all cases except for the bird image, we outperform the Canny edge detector (run with default settings in MATLAB) according to the F-measure. For the surfer image, we see that the algorithm clearly rules out almost all of the water, and almost exclusively returns positive edge matches for the surfer. The results of our edges for Figure 3 look satisfactory visually, because we can see that they are part of the surfer. However, the high number of false positives result in an extremely low F-measure.

The axon image was used to provide an example of a real world "hard" example, where Canny is completely useless. Our method outperforms Canny and picks up most of the relevant edges, however, it again suffers from an extremely high false positive rate.

6.1. Computational Cost

The computational overhead of the AdaBoost framework is negligible compared to the cost of updating the thousands of weak classifiers for each training sample. Per training example, the computational cost is \( O(N \cdot M) \) where \( N \) is the number of selectors and \( M \) is the number of features in a given selector at any time. With no optimizations besides using integral images, our MATLAB implementation is capable of learning at a rate of between 1-2 examples per second using 7200 weak learners. An optimized C++ implementation would likely be able to learn online at a rate faster than a human could produce reliable training data.

7. Future Work

Due to computational limitations, only a limited number of images and conditions were evaluated using our online edge detection algorithm. The preliminary results warrant further investigation and also show that there is clear room for improvement. A fair comparison with other offline edge detection methods should be performed, which would require training using the entire 200 Berkeley Segmentation Dataset training images.

There are numerous avenues for improvement of the existing algorithm. The size of the global feature pool could
be vastly expanded, as well as diversified to include more than simple Haar-like features. We would like to investigate the effects of various priming and bootstrapping methods to further improve the performance of the algorithm in the early stages when few training examples have been seen.

A more detailed analysis of the effects of the configuration of the online boosting framework, by varying the number of selectors and the number of features per selector should be performed.

Finally, we would like to pursue the theory behind the sample importance estimation technique proposed by Oza and used in this algorithm. It would be interesting to see if alternative methods exist, and to experiment with using the importance estimate to inform other components of the algorithm, such as the deciding how many weak classifiers to replace after each training example.

8. Conclusion

In this work we applied an online boosting classification framework to the edge detection problem, and demonstrated a small sample of its capabilities. The boosting paradigm does not require any manual feature selection, as the relevant features are automatically chosen from the feature pool, and become optimized for the image incrementally as more training data are provided.

Given the small amount of input training data utilized, and the simplicity of our classification framework, we did not expect to outperform the any of the leading learning edge detection methods such as BEL. However, we successfully demonstrated an online edge learning algorithm which warrants further investigation.

Our online boosting classifier has a limit to its discriminative capability due to the fixed number of features it combines, however it has the advantage of being able to adapt to handle variation in the input data over time. Also, it requires no training data prior to the start of the algorithm. With a relatively small number of sequentially provided training samples we are able to produce plausible edge maps for images in two different domains. Our method has potential applications in interactive segmentation, active contours, and any task where real-time edge detection can be used in the algorithm pipeline.

References


Figure 4: Bird

Figure 5: Axon

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Method</td>
<td>0.1</td>
<td>0.86</td>
<td>0.17</td>
</tr>
<tr>
<td>Canny</td>
<td>0.04</td>
<td>0.36</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 2: Surfer - threshold = 0.4

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Method</td>
<td>0.1</td>
<td>0.43</td>
<td>0.16</td>
</tr>
<tr>
<td>Canny</td>
<td>0.06</td>
<td>0.22</td>
<td>0.09</td>
</tr>
</tbody>
</table>

Table 4: Axon

Table 3: Bird - threshold = 0.35

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canny</td>
<td>0.36</td>
<td>0.42</td>
<td>0.39</td>
</tr>
<tr>
<td>Our Method</td>
<td>0.21</td>
<td>0.44</td>
<td>0.29</td>
</tr>
</tbody>
</table>

