A System for Rapid Interactive Training of Object Detectors

No Author Given

No Institute Given

Abstract. Machine learning approaches have become the de-facto standard for creating object detectors (such as face and pedestrian detectors) which are robust to lighting, viewpoint, and pose. Generating sufficiently large labeled data sets to support accurate training is often the most challenging problem. To address this, the active learning paradigm suggests interactive user input, creating an initial classifier based on a few samples and refining that classifier by identifying errors and re-training. In this paper we seek to maximize the efficiency of the user input; minimizing the number of labels the user must provide and minimizing the accuracy with which the user must identify the object. We propose, implement, and test a system that allows an untrained user to create high-quality classifiers in minutes for many different types of objects in arbitrary scenes.

1 Introduction

For many vision applications, it is important to have detectors for specific object classes. In recent years, the simultaneous development of large data sets and advances in machine learning algorithms have made it possible to make visual object detectors quite robust, for example, face detection is efficient and robust enough to be built into many modern digital cameras. In fact, machine learning techniques are often more accurate than hand-coded algorithms for many object detection problems, if sufficient training data is available. However as pointed out in [1], the hand annotation of training data is often the most time consuming part of the process of creating an accurate detector.

We believe there are two key challenges to overcome to make interactive training a near real time process:

1. The feature set and learning algorithms must be efficient to compute so that a user can complete the classifier training in one sitting.
2. User labeling should be "simple," ideally requiring just one click (rather than outlining an object), and it should accommodate several pixel errors in the click.

Our contributions in this paper are the integration tools to address each of these challenges. We describe a workflow and system that makes it possible to generate an effective classifier in just a few minutes. This comprises three contributions, first the integration of histogram of oriented Gaussian (HoG) features

and an incrementally trained SVM (Support Vector Machine) classifier. Second, a method to locally correct the positions of an object designated by a user click, and an experimental analysis of classifier performance with and without the click correction. Third, a characterization of the reduction in number of labeled samples required to train a classifier using the iterative approach versus standard single-shot classifier training. Finally, we present empirical results from classifiers that our system has developed for video data from YouTube, and archived webcam data from the "Archive of Many Outdoor Scenes" data set [2].

2 Related Work

The recent enthusiasm for the use of machine learning algorithms for object detection is perhaps spearheaded by Viola-Jones face and pedestrian detectors [3]. This work considers a very large feature space of Haar-like features, and uses AdaBoost to select a small set of discriminative features. The training set size is typically quite large with tens to hundreds of thousands of examples of the object and the same or more negative examples. There have been many papers exploring object detection with a variety of feature spaces, learning algorithms, and objects to be detected; a few examples showing the variation of approaches to pedestrian detection include using histogram of oriented gradients as a feature and linear SVM as the classifier [4], using Haar features with support vector machines [5] and local receptive fields similar to Gabor filters with neural networks as a classifier [6].

Previous research has directly considered iterative algorithms for incrementally training object detectors, creating incremental algorithms for boosting [7], and integrating this with a tracking algorithm, using the tracking results to maintain an up-to-date object classifier despite appearance changes [8].

Another approach uses stereo imagery from an aerial platform to detect false positives (whose depth is not consistent with targets of interest) which are feed back into the classifier as negative updates, substantially improving performance [9].

Other current work focuses on users cooperating with an algorithm to perform tasks. One exemplar of this cooperation is in the "Livewire" or "Intelligent Scissors" segmentation algorithms [10, 11]. In these, a user clicks on a point to define a graph minimization problem to easily define complex boundaries. These algorithms are successful *because* there is immediate feedback allowing the user to click a few extra points in order to create the desired boundary. Another field where iterative feedback between users and algorithm parameters are applied to computer vision is in Content Based Image Retrieval, where a few iterations of user queries and refinements continues to give state of the art results (a comprehensive review is [12]).

To the best of out knowledge the only work done on including people in an online training loop is the Seville system, created by Abramson and Freund [1]. Their iterative approach focuses the human's efforts on labeling data that are initially mis-classified. Such data points are likely to be near the final decision

boundary and therefore are most useful in training. However, this promising system still required 4 hours of human time in hand-labeling data, and a wall-clock time of 30 hours (mostly taken by iterations of the boosted learning algorithm) to train a pedestrian classifier to be effective on video data from a moving car. While this dramatically improves upon the weeks or months of time that have gone into making standard training data sets, it does not allow a user to quickly make a new detector.

3 Near Real Time Interactive Object Detector Training

In order to further speed up the process of object detector training, this work focuses on integrating faster learning algorithms, with tools that make the user interaction more effective. In particular, the goal is to minimize the time that the user spends waiting for the detector to be retrained, and to allow the user to specify an object with a single click that need not be accurate at the pixel scale. To that end, we have created a system that meets these goals. In this section we describe the algorithmic choices that we have made to support such a workflow.

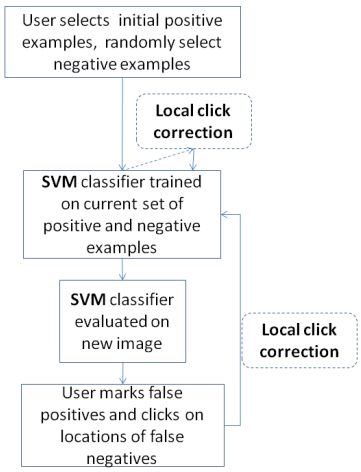

Fig. 1. The workflow for interactive training of an object classifier. The bold terms and dotted lines show our contributions beyond previous work [1], using the (much) faster to train SVM classifier, and a "click correction" allowing users to click approximate locations reduces the total time to create a classifier from 30 hours to several minutes.

Figure 1 illustrates the workflow for the interactive object detector. This workflow begins with a user selecting positive examples of the object he or she wishes to detect, by clicking on them. The remainder of the image is then randomly sampled 1500 times to initialize a negative example set. Once this is finished the system advances some number of frames in the input video and begins the training and object detection phase, which is described in detail in Section 3.1. After the objects in the new video frame are detected, the system enters the interactive phase. In this phase the user corrects mistakes made by the detector, by indicating incorrect or missing detections (each requiring a single click). Figure 2 shows a screenshot of the GUI after the user has completed this process. False negatives are then added to the positive example set, and false positives are added to the negative example set. When finished with the interactive phase, the user clicks a button on the GUI and the video is once again advanced, the training and object detection phase occurs, and finally

Fig. 2. A screenshot of the user interface after one iteration of training on the peachtree traffic video. The user has marked some detections as false positives (red). The user marked two false negatives (green). The correct detections are marked in blue.

the system returns to the interactive phase. In this way the user can iteratively guide the training of the detector and refine its decision surface, until satisfied with the detector's performance.

3.1 Object Detection

Unlike offline learning, interactive training has a human in the loop. This necessitates that the features and training algorithm used in the detector must be extremely fast.

For detection we use a Histogram of Oriented Gradients (HoG) feature descriptor. To compute the feature an image box is divided into 4×4 blocks, and for each block an image gradient direction histogram was created with 10 angular bins. For the car detection data featured in the results section, the image box is 24×24 pixels. Since these features are used in a scene-specific detector, it is unnecessary to normalize the feature vector. An advantage of the HoG feature descriptor is that it divides an image patch into axis-aligned rectangular blocks. This allows gradient magnitudes in blocks to be summed in constant time by precomputing the integral of the image gradient for the entire video frame. Note that while we used the HoG feature descriptor in our system, any reasonable method of creating a feature descriptor vector from a image box could be substituted into the system in place of HoG.

As a classifier we use a linear SVM, trained with the Sequential Minimal Optimization (SMO) algorithm, implemented in MATLAB [13].

Detection is performed by sliding a window over the current frame of the video. For each image window a HoG feature is calculated and evaluated by the SVM, which outputs a single detector score for each image window. This process can be greatly sped up by precomputing the image gradient bins for the entire image and by processing these windows in parallel. Boxes indicating the location of detections are superimposed on the image for all windows that have a score above a threshold selected by the user.

Slightly offset boxes from an image are likely to have very similar HoG features. In order to avoid overwhelming the user with many detections of a single object, we used non-maxima suppression on the detections.

3.2 User Interaction

In our approach, the training of the SVM classifier is done iteratively, starting with just a few labeled examples, and allowing the user to correct mistakes. In this section we detail the user interaction itself, including an automatic way to "correct" a user click with respect to the current SVM classifier, which further increases the efficiency of the human interaction.

A user corrects the detection results with two tools: mark false positive and mark false negative. To mark a false positive the user selects that tool and then clicks inside of a detection rectangle. The rectangle is recolored red to indicate that it has been marked. To mark a false negative the user clicks on the object missed by the detector. A green detection rectangle is drawn around the object. The system will make slight corrections to the location of the user's click. This process is described in Section 3.3.

3.3 User Click Correction

It is very unlikely that a user will click the exact same place on cars when marking missed detections. Therefore if the system were to simply add a positive example to the training set based only on the point where the user clicked, it is likely that the decision surface of the SVM will move to accept many different offsets of image boxes relative to true car locations. To correct this in our system, when the user clicks a false negative, the SVM score is computed for all boxes within 2 pixels of the original click, and the box with the maximum score is added to the positive training set. In this way, the SVM self-reinforces a correct click position.

4 Experimental Analysis

In this section we characterize the performance of an interactively trained classifier. We measure classification performance using a performance curve, and we explore performance as a function of the number of user-clicked points, accuracy of point clicking, and various choices of what data should be incrementally added to the SVM training set.

4.1 Quantitative Evaluation of Active Learning

In this experiment, we aim to quantify the improvement of the classifier when using incrementally chosen examples versus random examples. We compare the incrementally trained classifier to a standard classifier over many trials to characterize the performance distributions over different random samples. This experiment is performed using positive examples from the MIT pedestrian data set [14] and negative examples drawn randomly from natural images known not to contain humans.

The training data includes 200 positive examples of pedestrians, and a very large set of non-pedestrians. To create the baseline classifier, we randomly sample 9 positive examples and 18 negative examples, and compute the classification error over a fixed testing set of 419 positive examples and 725 negative examples, choosing a threshold that minimizes total mis-classifications. Performing 500 trials of this form gives the distribution of total mis-classifications shown in blue in Figure 3.

In comparison, the incremental classifier was trained as follows:

1. Randomly select 3 pedestrian images and 6 non-pedestrian images as initial training data, and train the SVM classifier.
2. Classify all remaining *training*-set images, randomly select 3 mis-classified positive examples and 6 mis-classified negative examples, and retrain the classifier.
3. Again classify all remaining *training* images, randomly select 3 mis-classified positive examples and 6 mis-classified negative examples, and retrain the classifier.
4. Using this SVM classifier, compute classification results on the same test set as used by the baseline classifier.

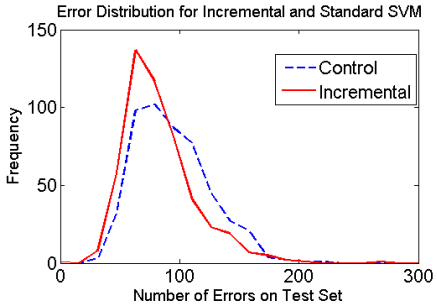

Fig. 3. Even with small sample sizes, iterative training of an SVM classifier noticeably outperforms batch training on the MIT pedestrian dataset.

This classifier is evaluated over 500 random trials (encompassing different initial images and different random choices from the mis-classified images). Figure 3 shows the distribution of total mis-classifications in red, illustrating that even at low sample sizes, there is a noticeable improvement in the performance of the incrementally trained classifier.

4.2 Experimental Setup with Real World Data

The testing data for the traffic video is taken from the Peachtree dataset (available at http://ngsim.fhwa.dot.gov/). A screenshot of this video is shown in Fig-

ure 2. The testing data includes 50 frames of video, each 10 seconds apart, taken from a section of the video that does not overlap with the portion used for the incremental training. Ground truth was created by hand. Cars that were mostly occluded and other types of vehicles such as trucks and buses were not labeled as cars. The testing data contained 601 labeled cars.

The training was performed by a user utilizing our system on the training portion of the peachtree video and was stopped after 20 iterations.

4.3 Experimental Study of "Click Correction"

In order to evaluate the effect of click correction a user trained a car detector on the peachtree data as described above twice—once with "click correction" enabled, once with it disabled. Figure 4 shows a performance curve comparing the performance of the two detectors after five iterations of training. Note that at a false positives per true positive rate of 0.1 the detector without click correction had a true positive rate of 0.51, while the detector with click correction had a true positive rate of 0.75.

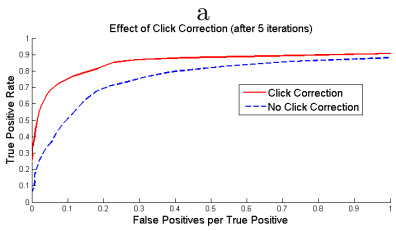

Fig. 4. The click correction substantially improves performance, by removing the variability in the object class due to small position errors.

The effect of click correction diminishes as the number of training iterations increases. However due to the improved performance during early rounds of training, click correction allows a user to train a detector with significantly fewer clicks.

4.4 Overall Results and User Time to Create Classifier

In this section we show the performance of the object detector trained iteratively using our system on the traffic video. We also compare this with the performance of a detector trained non-iteratively. Positive examples for the non-iteratively trained detector were obtained by manually labeling the location of all cars in the same frames of the peachtree video that the iterative training was performed on. Negative examples were obtained by randomly sampling boxes that did not contain cars in these frames. Figure 5 shows the performance of the iteratively trained detector for different numbers of training iterations, as well as the performance of the non-iteratively trained detector.

Figure 5 demonstrates that as more iterations of training are performed the performance of the object detector increases. Note that the iteratively trained detector *outperforms* the non-iterative detector after only 10 training iterations. Also note that the legend shows the number of positive and negative examples included in the training of each of these detectors. With six times as many

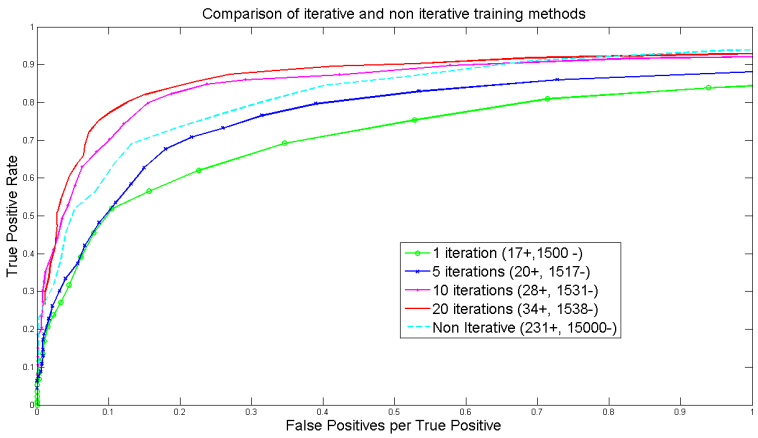

Fig. 5. Creating a detector for cars in a surveillance scenario, the performance of an iterative classifier rapidly matches batch training with much larger training sets.

positive examples and nearly ten times as many negative examples as the 20-iteration detector, the non-iterative detector still fails to perform as well. We hypothesize that this is because the non-iterative detector never received the *right* negative examples—those required to fine tune the decision boundary, even though overall it had many more negative examples.

Training the 20-iteration detector only required a total of 72 clicks by the user (17 initial cars, 17 false negatives, 38 false positives, 1500 of the negative examples are generated randomly). On the other hand, to label all of the cars for the non-iterative detector the user had to click on 231 cars. This saves the user a significant amount of time. It takes a user who is experienced with the system seven minutes of wall-clock time to train the 20-iteration car detector on the traffic video.

5 Discussion and Conclusions

This work aims to explore the premise that interactive training of an object detector requires much less hand-labeling of data than standard batch training. Careful choices of image feature, learning algorithm, and minimal image processing to correct some variance in the user's click location reduces the overall wall-clock time of training a detector to be minutes instead of the 30 hours previously reported. This opens up the possibility of using a trained object detector in a wider variety of arbitrary and niche uses.

To illustrate this, we have trained object detectors on objects in several videos from YouTube (after downloading and converting them to AVI). Figure 6 shows results on new frames (not included in the training set) for detecting sailboats ("wall-clock" training time, 1.5 minutes), Sony Aibo robots used in the Robocup

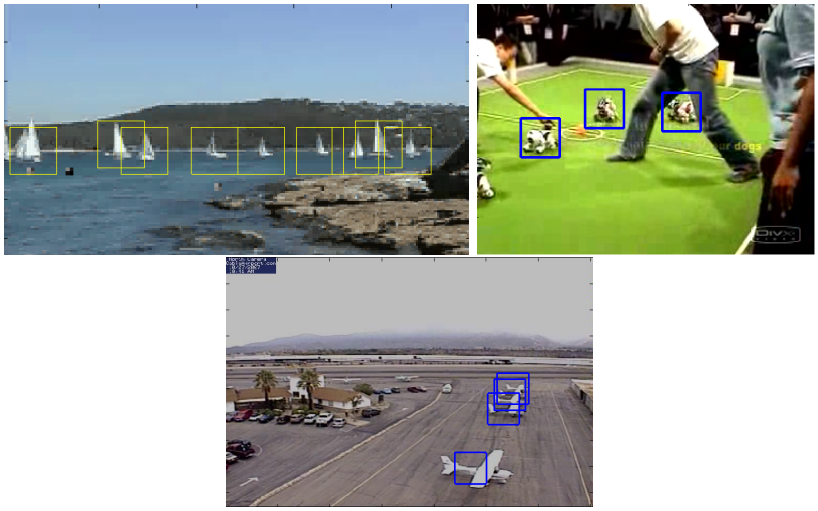

Fig. 6. The overall benefits of this work is the simplicity with which new classifiers can be made for arbitrary applications. This figure shows example results from classifiers trained in under 2 minutes on video clips from YouTube (top) and archived webcam data (bottom).

soccer challenge ("wall-clock" training time, 2 minutes), and airplanes from a general aviation airport webcam archived in the AMOS database [2].

In the past, learning models for object detection required significant human effort in creating sufficient training data, so this was done only for high-impact problem domains (e.g. faces, pedestrians). But the ability to iteratively train a detector with just a few clicks makes it feasible to train special purpose detectors for specific objects (e.g. Sony Aibo), or from individual viewpoints. This may make possible new computer vision applications in surveillance (especially scene specific surveillance algorithms), biological monitoring (detect the ducks on this pond), or microscopy (count the number of a particular cell type) applications.

References

1. Y Abramson and Y Freund. Semi-automatic visual learning (seville): a tutorial on active learning for visual object recognition. *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1, 2005.
2. Nathan Jacobs, Nathaniel Roman, and Robert Pless. Consistent temporal variations in many outdoor scenes. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6, 2007.
3. Paul A. Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 511–518, 2001.

4. N. Dalai and B. Triggs. Histograms of oriented gradients for human detection. *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1:886–893, June 2005.
5. Constantine Papageorgiou and Tomaso Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.
6. Christian Wöhler and Joachim K. Anlauf. Real-time object recognition on image sequences with the adaptable time delay neural network algorithm - applications for autonomous vehicles. *Image and Vision Computing*, 19(9-10):593–618, 2001.
7. Helmut Grabner and Horst Bischof. On-line boosting and vision. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 260–267, 2006.
8. Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *British Machine Vision Conference (BMVC)*, pages 47–56, 2006.
9. S. Kluckner, G. Pacher, H. Grabner, H. Bischof, and J. Bauer. A 3d teacher for car detection in aerial images. In *Proc. IEEE International Conference on Computer Vision*, pages 1–8, 2007.
10. E. Mortensen, B. Morse, W. Barrett, and J. Udupa. Adaptive boundary detection using 'live-wire' two-dimensional dynamic programming. *Proceedings of the IEEE Conference on Computers in Cardiology 1992*, pages 635–638, Oct 1992.
11. Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In *Proc. ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 191–198, 1995.
12. Xiang Sean Zhou and Thomas S. Huang. Relevance feedback in image retrieval: A comprehensive review. *Multimedia Systems*, 8(6):536–544, 2003.
13. G. C. Cawley. MATLAB support vector machine toolbox (v0.55β) [`http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox`]. University of East Anglia, School of Information Systems, Norwich, Norfolk, U.K. NR4 7TJ, 2000.
14. M. Oren, C.P. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 193–99, 1997.