Learning to Recognize Objects in Images

Huimin Li[*] and Matthew Zahr[†]

December 13, 2012

1 Introduction

The goal of our project is to quickly and reliably classify objects in an image. The envisioned application is an aid for the visually-impaired in a real-time situation, i.e. an algorithm that can be run on a mobile device, with real-time environmental data being collected via the device's camera. Ideally, the camera would capture an image of the user's path and the proposed method would determine if any obstructions pose a threat. Due to time limitations of the quarter, we did not carry this project through the envisioned application; instead we laid the foundational steps, primarily algorithm development and training.

Our application to assisting the visually-impaired in real-time has a very natural separation of the training and testing phase. In the training phase, it is assumed that computational time and resources are essentially unlimited. This phase can be thought of as product development where a large amount of computational effort is devoted to learning an algorithm. Once the algorithm has been sufficiently trained, it is deployed on a mobile device and must be able to rapidly detect obstructions. In practical situations, training time is an important consideration since development cost is proportional to development time. This issue is addressed in our approach.

The approach used incorporates four computer vision and machine learning concepts: sliding windows to extract sub-images from the image, feature extraction to get meaningful data from the sub-images, Support Vector Machines (SVMs) to classify the objects in sub-image, and Principle Component Analysis (PCA) to improve efficiency. As a model problem for the motivating application, we focused on the problem of recognizing objects in images, in particular, soccer balls and sunflowers. For this algorithm to be useful as a real-time aid to the visually-impaired, it would have to be enhanced to distinguish between "close" and "far" objects, as well as provide information about relative distance between the user and the object, etc. We do not consider these complications in this project; we focus on the core machine learning issues of object recognition. The training and testing of the proposed algorithm was done using data sets [3, 4].

2 Algorithm

In this section, we present the proposed algorithm for object recognition and clearly distinguish between the training and testing phases. Let us first introduce some notation. Define the set of training images as $\mathcal{I}_{train} = \{X_i\}_{i=1}^{p}$. At this point, we have not chosen a mathematical representation of the image, and therefore the notation used is abstract. We are taking a supervised approach to machine learning so we break this set into the positive training set $\mathcal{I}_{train}^{+} = \{X \in \mathcal{I}_{train} \mid X \text{ contains object of interest}\}$ and the negative training set $\mathcal{I}_{train}^{-} = \{X \in \mathcal{I}_{train} \mid X \text{ does not contain the object of interest}\}$. Similarly, the set of testing images $\mathcal{I}_{test} = \{Y_i\}_{i=1}^{q}$ is decomposed as the positive $\mathcal{I}_{test}^{+}$ and negative $\mathcal{I}_{test}^{-}$ examples. As mentioned in Section 1, many sub-images will be extracted from each *test* image and we denote sub-image j of test image i as Y_i^j.

2.1 Training

At a conceptual level, the training phase of our approach is broken into four stages: (1) filter the background out of the image (or equivalently, locate the objects in the image), (2) extract pertinent features from the image (usually high-dimensional), (3) construct a subspace on which to approximate the feature vector (optional), and (4) train the model based on the reduced representation of the feature vector.

[*]Graduate Student, Department of Computer Science
[†]Graduate Student, Institute for Computational and Mathematical Engineering

The background filtering or object extraction step is achieved manually by centering the object in the image and cropping to 256×256 pixels. For the proposed algorithm, it is important that all training images are the same size because the software package we are using to extract features will generate feature vectors of different dimensions for images of different sizes. Without additional information, data vectors of different dimensions cannot be compared properly using standard classification algorithms.

The feature extraction phase uses the standard computer vision software VLFeat [6] to extract a Histogram of Object Gradients (HOG) from a given image. This software takes our images X_i as input, and returns a vector of HOG feature data $\mathbf{x}_i \in \mathbb{R}^n$. The images in our data set (after cropping) were 256×256 pixels, which resulted in HOG vectors of dimension $n = 7168$.

The optional step of dimension reduction is achieved using Principle Component Analysis (PCA). The procedure to apply PCA to this problem is to collect the HOG feature vectors into a matrix and compute its Singular Value Decomposition (SVD): $\begin{bmatrix} \mathbf{x}_1 - \boldsymbol{\mu} & \mathbf{x}_2 - \boldsymbol{\mu} & \cdots & \mathbf{x}_p - \boldsymbol{\mu} \end{bmatrix} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$, where $\boldsymbol{\mu} = \frac{1}{p}\sum_{i=1}^{p} \mathbf{x}_i$. A low-dimensional representation for the feature vectors can be obtained by approximating them as a linear combination of the first k columns of $\mathbf{U}$: $\mathbf{x}_i - \boldsymbol{\mu} \approx \boldsymbol{\Phi}\tilde{\mathbf{x}}_i$, where $\boldsymbol{\Phi} \in \mathbb{R}^{n \times k}$ is the matrix formed from the first k columns of $\mathbf{U}$ and $\tilde{\mathbf{x}}_i \in \mathbb{R}^k$. For this to be useful, we require $k \ll n$. Since $\boldsymbol{\Phi}$ is an orthogonal matrix, we can recover the reduced vector by simply projecting the feature vector onto the basis: $\tilde{\mathbf{x}}_i = \boldsymbol{\Phi}^T(\mathbf{x}_i - \boldsymbol{\mu})$. The SVD is generally considered an expensive operation, but there exist efficient algorithms for computing only a few dominant left singular vectors [1]. The purpose of using PCA is to gain efficiency in the *training* phase; as we show later, the online (testing) cost is independent of whether PCA is used. If there are many training images, which will be the case in practical situations, the cost of training the SVM will be high because there are terms in the optimization formulation that involve sums over all training data (where each data point is a HOG vector). Therefore, the incorporation of PCA will reduce the training cost if: there are many training images, only a few dominant singular vectors are required, and these dominant singular vectors can be computed quickly. As mentioned in the introduction, the motivation for reducing offline cost is to minimize product development cost while maintaining accuracy.

The next step of the proposed approach consists of training a Support Vector Machine (SVM) using the HOG feature vectors, $\mathbf{x}_i$, or their reduced representation, $\tilde{\mathbf{x}}_i$, along with their known classification as training data. Our implementation leverages the SVM library LIBLINEAR [2] to perform this step. The output of the SVM is a vector defining the normal to the linear classifying hyperplane and a scalar defining the offset. These quantities are denoted $\mathbf{w} \in \mathbb{R}^n$ and b, respectively if the full HOG vectors are passed to the SVM or $\tilde{\mathbf{w}} \in \mathbb{R}^k$ and $\tilde{b}$, if the reduced HOG vector representation is passed to the SVM. At this point, we also define the vector $\bar{\mathbf{w}} = \boldsymbol{\Phi}\tilde{\mathbf{w}}$, which will be used in the testing phase (if PCA used).

2.2 Testing

The time-critical testing phase involves four steps: (1) extract sub-images from the original image, (2) extract HOG feature vector from each sub-image, (3) determine the reduced representation of the feature vectors (optional), and (4) classify the image by using the trained SVM on the sub-images.

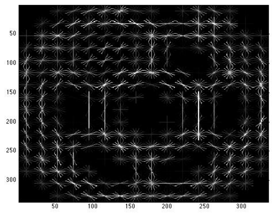
The first step is achieved by considering a fixed-size bounding box (window) that slides over the entire image. Every time the bounding box moves, the portion of the image within its domain is extracted as a sub-image. To generate a sufficiently large number of sub-images for appropriate identification of an object, the bounding box begins in the upper-left position of the image and moves 8 pixels horizontally at each step; once the edge of the image is reached, the window is moved down 8 pixels and begins sliding to the left in 8 pixel increments. The sliding window approach allows the SVM built in the training phase to be robust with respect to the position of the object in an image and background noise. By using bounding boxes of several different sizes, the algorithm becomes more robust with respect to scaling of the object. Every time a sub-image, Y_i^j is extracted, its HOG vector $\mathbf{y}_i^j$ is determined using [6].

If PCA is not used, the third step is skipped and classification is performed on the full HOG vectors $\mathbf{y}_i^j$ using $\mathbf{w}$ and b. We classify an image Y_i as a positive example (i.e. contains object of interest) if $\mathbf{w}^T\mathbf{y}_i^j + b > 0$ *for at least 3 values of* j; otherwise, it is classified as a negative example. This is interpreted as claiming that image Y_i contains the object of interest if at least three of its sub-images contain it. If PCA is used to generate low-dimensional representations of the HOG vectors, the HOG feature $\mathbf{y}_i^j$ must be projected onto the subspace $\boldsymbol{\Phi}$: $\tilde{\mathbf{y}}_i^j = \boldsymbol{\Phi}^T(\mathbf{y}_i^j - \boldsymbol{\mu})$. Then, an image Y_i is classified as positive if $\tilde{\mathbf{w}}^T\tilde{\mathbf{y}}_i^j + \tilde{b} > 0$ for at least three values of j.

At first glance, it appears that PCA actually has a more expensive testing phase than does the algorithm without PCA. This can be seen from the fact that the non-PCA algorithm requires a single inner product of n-dimensional vectors ($\mathbf{w}^T\mathbf{y}_i^j$) *for each* j to classify Y_i, but the algorithm with PCA first requires k inner products of n-dimensional vectors ($\boldsymbol{\Phi}^T(\mathbf{y}_i^j - \boldsymbol{\mu})$) and a single inner product of k-dimensional vectors ($\tilde{\mathbf{w}}^T\tilde{\mathbf{y}}_i^j$) for each j. However, our introduction of $\bar{\mathbf{w}} = \boldsymbol{\Phi}\tilde{\mathbf{w}}$ in the previous section, reduces the testing cost of the PCA algorithm to a single inner product of n-

dimensional vectors for each j: $\tilde{\mathbf{w}}^T\tilde{\mathbf{y}}_i^j = \tilde{\mathbf{w}}^T\mathbf{\Phi}^T(\mathbf{y}_i^j - \boldsymbol{\mu}) = (\mathbf{\Phi}\tilde{\mathbf{w}})^T(\mathbf{y}_i^j - \boldsymbol{\mu}) = \bar{\mathbf{w}}^T(\mathbf{y}_i^j - \boldsymbol{\mu})$. Therefore, the online cost is the same regardless of whether PCA is used. Also notice that pre-computing $\bar{\mathbf{w}}$, eliminates the need to store reduced basis $\mathbf{\Phi} \in \mathbb{R}^{n\times k}$ during the testing phase. This is a significant observation if the training phase of our algorithm is performed on mobile devices, with limited memory.

Figure 1: Soccer Ball Image and visualization of its Feature Vector



3 Results

In this section, we present the results of applying the algorithm in Section 2 to determining if an image contains a soccer ball or sunflower. Our investigation of the study is performed by progressively improving the robustness of the algorithm by adding complexity to the algorithm. Initially, instead of using the sliding window approach, we manually pre-process each test image. This initial approach is tested with and without the incorporation of PCA. The next step incorporates the sliding window approach.

3.1 Study 1: Test Images Manually Pre-Processed (No Sliding Window)

In this section, the algorithm in Section 2 is applied to recognizing a soccer ball in an image without the added complexity of the sliding window; all test images are manually pre-processed by centering the object and cropping the image to the appropriate size. We trained our algorithm with 50 positive examples and 50 negative examples. The algorithm was tested on a set containing 28 positive examples and 51 negative examples. The quality of our algorithm is assessed by the number of mis-classified test examples, which we present in terms of false positive rate, false negative rate, and total error rate in Table 1. It can be seen that the error rate is quite low and incorporating PCA with dimension 20 introduces *no additional error*. This initial version of the algorithm will fail if the object is not manually centered or the background is not removed; it relies on the image pre-processing. This algorithm has relatively low errors because we are explicitly telling the algorithm the location of the object by manually cropping. Therefore, the errors in Table 1 essentially serve as a lower bound for the errors we expect from the sliding window implementation. If using sliding windows, we lose a bit of accuracy, but we gain robustness with respect to object position and scaling, and the online algorithm is entirely automatic.

Table 1: Soccer Ball Example with Manual Test Image Pre-processing and PCA

	No PCA	**PCA dim 5**	**PCA dim 10**	**PCA dim 15**	**PCA dim 20**
False Positive Rate	4%	10.53%	5%	8%	4%
False Negative Rate	7.41%	18.33%	15.25%	9.26%	7.41%
Total Error Rate	6.33%	16.46%	12.66%	8.86%	6.33%

3.2 Study 2: Sliding Window

In this section, we repeat the experiment from Section 3.1 with the incorporation of the sliding window. We consider four different sliding windows: $32 \times 32, 64 \times 64, 128 \times 128$, and 256×256 pixels, which will be denoted SWI, SWII, SWIII, and SWIV, respectively. Table 3 shows the results of varying the number of sliding windows used. As expected, incorporating different sliding window sizes improves the accuracy of the algorithm by improving robustness with respect to object scale. Table 3 also shows that incorporating the smallest sliding window, SW1, actual increases the error. This is an artifact of the circular shape of soccer balls; once you zoom in on an image (with background noise) enough, it is likely that there will be some objects with circular shape and the algorithm will generate a false positive.

Table 2: Soccer Ball Example with Sliding Window

	SWIII	**SWII & SW III**	**SWII - SWIV**	**SWI - SWIV**
False Positive Rate	26.8%	28.8%	28.4%	36.7%
False Negative Rate	42.0%	32.2%	31.4%	29.0%
Total Error Rate	38.1%	30.6%	30.0%	33.8%

Table 3: Soccer Ball Example with Sliding Window (SWI-SWIV) and PCA

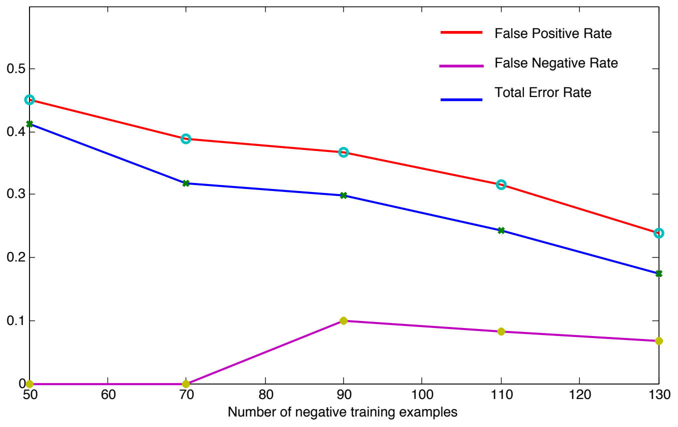
	dim 50	**dim 75**	**dim 100**
False Positive Rate	45.5%	48.1%	43.1%
False Negative Rate	26.9%	0%	35.3%
Total Error Rate	42.5%	46.1%	40.6%

3.3 Study 3: Additional Object - Sunflower

We observed the soccer ball example generated a large number of false positives. This is attributed to the commonality of circular objects in a natural image (i.e. human head or fist, tennis ball, plates, etc. can all be considered roughly circular in a 2D image). To test this hypothesis, we performed an abridged version of Study 2 except we tried to classify images based on whether they contain a sunflower. Sunflowers were chosen because their unique shape will reduce the likelihood of finding similar-shaped objects in an image.

Experimentation with this algorithm along with a literature search emphasized the importance of the negative training set size and quality. Figure 2 illustrates the importance of training set size by plotting the three error rates versus negative training set size. An important feature of this plot is the original negative training set is generated from random, non-sunflower images; however, when the negative training set is enlarged, the additional examples are chosen based on a heuristic relevance criterion. For the smallest negative training set sizes (50 and 70), there is a false negative rate of 0 because the algorithm classified *every* test image as positive (i.e. there were no predicted negative images).

Figure 2: Sunflower Example: Error versus Negative Training Set Size

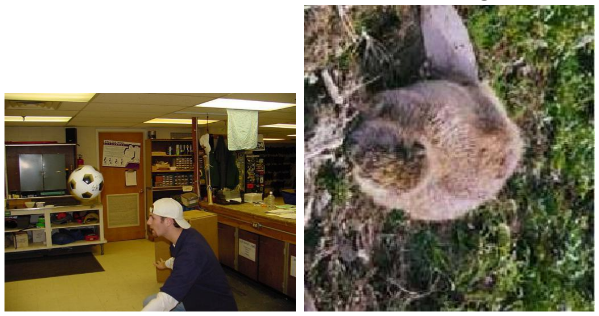


4 Conclusion

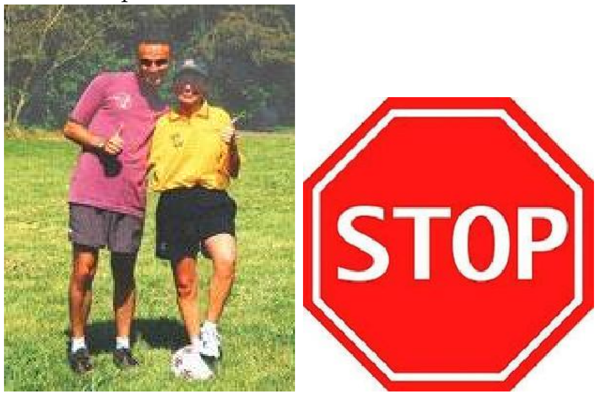
In this project, we used computer vision and machine learning techniques such as sliding windows, HOG, PCA, and SVM to determine whether images contain objects of interest. Particular attention was given to keep training costs low by incorporating PCA. The proposed algorithm had a high success rate on object-centered test images, but had no robustness with regard to object scaling and position. Incorporation of sliding windows increased the error rate but improved robustness.

Future work will focus on improving the robustness of the proposed algorithm to false positives by using an enriched set of negative training examples. Another direction is to improve the performance of PCA with sliding windows by using a local basis for each object in addition to the global basis [5]. A necessary enhancement for our

Figure 3: Soccer Ball Example

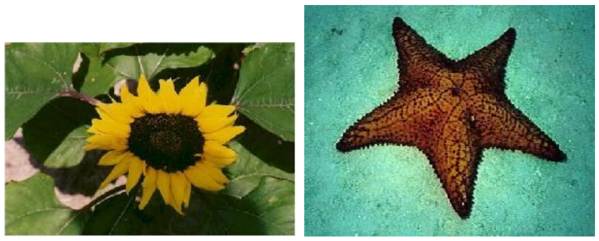


Properly Labeled Positive and Negative Examples

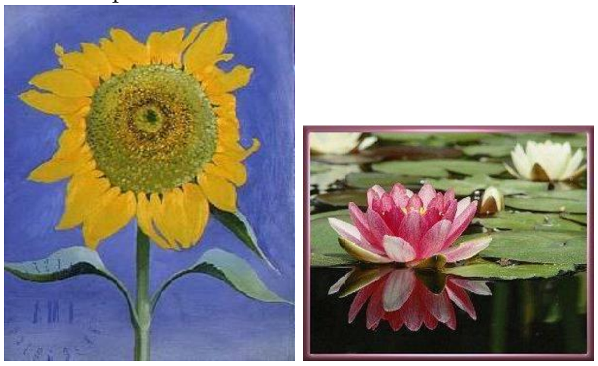


Mislabeled Positive and Negative Examples

Figure 4: Sunflower Example



Properly Labeled Positive and Negative Examples



Mislabeled Positive and Negative Examples

eventual application will be to generalize to multinomial classification for distinguishing multiple objects in a given image.

References

[1] Y. Chahlaoui, K. Gallivan, and P. Van Dooren. Recursive calculation of dominant singular subspaces. *SIAM Journal on Matrix Analysis and Applications*, 25(2):445–463, 2003.

[2] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[3] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. IEEE. CVPR 2004, Workshop on Generative-Model IEEE. CVPR 2004, Workshop on Generative-Model Based Vision, 2004.

[4] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[5] H. Murase and S. K. Nayar. Visual learning and recognition of 3-d objects from appearance. *Internal Journal of Computer Vision*, 14:5–24, 1995.

[6] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. *http://www.vlfeat.org/*, 2008.