Co-Tracking with a Manifold Anchor

Mason F. Matthews

December 8, 2005

Abstract

In its original form, co-training is a process by which multiple learners can boost classification accuracy using unlabeled data points. While co-training has been applied to problems such as visual detection [20] and web page identification [6], this paper describes its use in an application that tracks heads in video. The inherent problems with this approach are discussed, and a geometric solution is proposed. By creating a manifold representing possible poses and using the distance from this manifold as a regularization term, tracking accuracy can be improved. A series of nearest neighbor algorithms are presented, and empirical results using this manifold anchor are discussed.

1 Introduction

In many computer vision applications, we are given a sequence of frames of video and the goal is to track a moving object as its location changes, both in the real world and in the image plane. If two assumptions hold, this problem is relatively simple: first, the object must be accurately identifiable in some initial frame; second, the contents of the image window containing the object does not change as it undergoes translations. In this situation, simple convolutions can be performed on neighboring windows to determine the new location of the object in the image plane.

The first assumption, that an initial position can be found, may be framed as a detection or a recognition problem. These problems have been studied in great depth, and proposed solutions vary greatly from object to object. Examples of face and skin detectors can be found in [30] and [16]; more will be discussed in the background section. In the general case, it is assumed that a satisfactory method exists.

The second assumption, invariance of appearance over time, is somewhat more flexible. Consider the problem of tracking a person's head as it moves across a camera's field of view. Following the frame in which the head is detected, the image window that contains it may have moved. At first glance, this problem

appears trivial: simply perform a convolution of a head template or a window from the previous frame with every window in the new image and choose the one that matches best. However, consider the complications added in real video. Even if the head undergoes no rotation and its motion is parallel to the image plane, the contents of the window can vary: lighting conditions may be different, background pixels will have changed, and a change in viewing angle can make portions of the face disappear. If the motion is not parallel to the image plane, then the head may additionally appear to contract or expand, and the size of the window will have to increase or decrease to accommodate the change. While this does add some complexity to the problem, a few additional convolutions can detect these shifts, and in some applications the tracking can continue at the new scale using simple scaling factors.

However, if a rotating head is being tracked, the window containing the head can go through an enormous variety of transformations. A face pointed directly at the camera can evolve to a profile, the top of a head, or even the bottom of a jaw. One solution to this problem was proposed by Stan Birchfield [4]; by monitoring both a color histogram and an oval-shaped gradient, a head can be tracked through 360 degrees of rotation, though certain lighting restrictions have to be satisfied.

While it may not be immediately apparent, co-training can be used to accurately track an evolving object. In co-training [6], unlabeled data, typically less expensive to gather than labeled data, can be used to boost a learner's accuracy. Initially, at least two independent classifiers are trained on the same labeled data. After the training phase, unlabeled data is given to the system, and if one classifier makes a classification outside a specified margin from the decision boundary (that is, with a sufficient degree of confidence), then that label is matched with the observation, and the pair is passed to the other classifiers as a training point. Using this method, the effective number of labeled training samples can be increased dramatically.

In the tracking setting, evolution may be captured by a series of classifiers that each respond to a single allowable configuration. However, this brute-force method is impractical unless the number of potential configurations is small. Instead, consider a smaller set of classifiers which are each sensitive to independent components of evolution. When tracking heads, for example, one classifier can be sensitive to rotations around a vertical axis and a second can be sensitive to rotations around a horizontal axis. Co-training can then be employed. During horizontal rotation, the certainty of one classifier can be used to correct the uncertainty of the other. In essence, labels are being provided over time from unlabeled images. Through this interplay, the concept of "head" can evolve along with the image and allow for tracking during arbitrary rotations.

Since there exists no known classifier which can identify a head with perfect accuracy, it is possible for a co-training system to treat an incorrect label as truth. Unfortunately, this means that the tracker described

above can get lost and begin treating a non-head as the object to be tracked. While this is a problem that plagues all visual tracking systems, we seek to minimimize the effect.

One solution to this problem involves a manifold anchor. Since head rotations occur around two axes, the space of all head images makes up a low-dimensional manifold in high-dimensional feature space. This manifold can be approximated with a large set of images in known poses (and their corresponding adjacency matrix), and if we assume reasonable limits of the speed of rotation, motion in the manifold space from frame to frame can be limited to a local neighborhood. By regularizing (anchoring) the most likely windows with their distances from the manifold. This concept of a manifold anchor is explored in this paper.

Section 2 provides an overview of head tracking, co-training, and some of the work on manifold reconstruction. Section 3 brings these notions together to describe a novel approach to tracking. Geometric algorithms for improving efficiency are presented in Section 4, experimental results are presented in Section 5, and a discussion of these results can be found in Section 6.

2 Background and Related Work

2.1 Face Detection and Head Tracking

Research into face detection has been extensive, and some thorough survey papers exist (i.e. [13, 32]). Some methods attempt to encode human knowledge [31], some look for structural features [27], and others choose the best features for classification based on the samples in the training set. Viola and Jones [30] use this last method for face detection, and also develop a cascade of classifiers to remove previous false positives. Semi-supervised learning has also been applied to face detection and expression recognition [7].

These methods are effective for detecting faces when they are pointed towards the camera, but performance degrades as offset angle increases. Jones and Rehg [16] developed a color-based skin detection scheme that is not as susceptible to this problem, and Schneiderman and Kanade [28] extended their detectors to locate faces in any of three orientations. Even though this work was more robust than frontal face detectors, these methods could not follow heads through full 360 degree rotations. As mentioned earlier, Birchfield [4] developed a head-tracking algorithm which searched for windows containing oval-shaped intensity gradients and certain color histograms. This system could accurately track a head rotating around a vertical axis in many scenarios, but would occasionally find ovals of incorrect scale or lose track of the head under changes in lighting. Unfortunately, work such as this can also not be extended to track objects other than heads; shape and color assumptions will only hold for a certain class of objects.

In addition to these methods, a large number of applications employ particle filters for tracking. Despite

its simplicity and empirical success, the particle filter does suffer from the curse of dimensionality. More sophisticated variations have been developed which address this, including the Annealed Particle Filter [**?**] and Variational Sequential Estimation [29].

2.2 Co-Training

In the standard classification setting, each data instance x_i is drawn at random from a probability distribution D on the instance space X. The concept function h to be learned is a member of hypothesis space H, and the result of $h(x_i)$ is a label y_i. x_i is a positive sample if $y_i = 1$, and a negative sample if $y_i = 0$. Suppose that we are given a learning algorithm, and we train it with a particular training set of instances $(x_1, y_1), (x_2, y_2), ...(x_m, y_m)$ drawn from D. The goal of the algorithm is to find an estimate $\hat{h}$ and use it to generate classifications for a test set. If the algorithm generalizes well, an increase in the number of training points should increase the accuracy of the resulting function. For some applications, however, large numbers of labeled observations are not readily available.

In the *semi-supervised learning* framework, unlabeled data, typically available in abundance, can be used to boost a learner's accuracy. These algorithms can take a variety of approaches (for a thorough survey, see [33]). Manifold learning techniques use the unlabeled data to define a lower-dimensional manifold in the feature space; classification can then take place on the manifold [3]. Self-training allows a trained classifier to classify the most "obvious" unlabeled data points according to a given selection metric and add them to its training set [25, 26]. In addition to these, there has also been recent interest in using graphs to exploit the value of unlabeled samples [5, 15].

The Expectation-Maximization (EM) algorithm is also a semi-supervised learning technique. Given labeled and unlabeled data along with an assumption of how the data was generated (i.e. using a Gaussian distribution), EM will locally maximize the likelihood of the distribution parameters and estimate the missing labels. The algorithm iterates through two steps: the E-step calculates the most likely labels for each unlabeled data instance, and then the M-step recomputes the classifier given these labels. These steps are repeated until the classifier converges.

Co-training is a semi-supervised paradigm originally proposed by Blum and Mitchell [6]. In co-training, it is assumed that the instance space X can be broken into at least two independent views, f_1, f_2, and so forth. Training sets are generated for each view and are used to train different classifiers: classifier 1 will have access to the set $\{(f_1(x_1), y_1), (f_1(x_2), y_2), ...(f_1(x_m), y_m)\}$, and will not be given any of the remaining information in $x_1, x_2, ...x_m$. For instance, given the problem of classifying images of apples and bananas, one classifier may only discriminate based on color histograms of the image while the other discriminates based

on shape (potentially determined by image gradients).

After the training phase, unlabeled data is given to the system, often in one large set, and co-learning proceeds in stages. In each stage, the classifiers generate labels for the entire unlabeled set, and those which are most confidently labeled are added to the training set. The classifiers are then retrained and the process repeats. Alternatively, it can be assumed that unlabeled data are input one instance at a time. If one classifier makes a classification outside a specified margin (with a certain degree of confidence) then that label is matched with the observation, and the pair is passed to the other classifiers as a training point. As mentioned earlier, the effective number of training samples can be increased dramatically. However, the training set runs the risk of accumulating incorrect classifications whenever labels deemed "confident" are in fact wrong. This incorrect data will shift classification boundaries toward inclusion of the new points, making new errors even more likely.

Blum and Mitchell [6] applied co-training to a web site classification problem where one classifier viewed web page content and the other viewed the hyperlinks to the web page. However, they made the assumption that the classification function could be learned by either of the views alone, and that the responses in the two views are not correlated; in many applications, one or both of these assumptions do not hold. Multiple research groups have used co-training for natural language processing [8, 21, 24], both spoken and written. Extensions to co-training have been proposed by some of these researchers: Collins and Singer [8] used a boosted version of co-training called CoBoost, while Pierce and Cardie [24] employed corrected co-training, a method whereby a human annotator could prevent incorrect classifications from reaching the training set.

While the concept of co-training is intuitive enough, it is not immediately clear when the paradigm is effective. Nigam and Ghani [23] studied the classifier "independence" proposed by Blum and Mitchell, and showed that co-training does perform best when the two views are conditionally independent. However, they also compared the algorithm with self-training, EM, and co-EM, and found that co-training was effective even when an independent split of the view was not apparent. Balcan, et. al. [2] proposed that the important feature of the split is not the independence of the views, but how well the split satisfies the expansion assumption. This concept is similar to the graph-theoretic notion of expansion, which measures how well a weighted graph can be separated into strong clusters with weak connections between them.

2.3 Co-Training Errors

As mentioned above, it is possible for incorrect classifications to reach the training set. This can occur when there is noise in the data or when the classification function cannot be learned by each classifier. As would be expected, both of these situations are ubiquitous in computer vision applications.

Some research has been performed on this phenomenon, and the accumulation of incorrect labels can be bounded with high probability [10, 24]. These studies make a few assuptions that are reasonable in standard classification, but are not reasonable in tracking. The most important assumption is hardly ever mentioned: that classification labels of 0 or 1 are either correct or incorrect.

Unfortunately, this is not true in tracking. Consider the two bounding boxes given in Figure 1. Even though the rightmost box is not perfectly centered on the head, this window does give a good estimate of head location. Accepted metrics for tracking accuracy use overlap measurements (discussed in Section 5), which blur the line between "correct" and "incorrect."

Figure 1: The leftmost image shows the window that exactly fits the head. The window in the rightmost image is somewhat shifted, but is not an "incorrect" choice of head locations.

When tracking with co-training in the presence of noise, a window with, say, 96% overlap may be repeatedly added to the training set, and drift can occur as shown in Figure 2. In this setting, bounds on the correctness of the training set do not apply.

Figure 2: A series of images demonstrating the drift that can take place during tracking.

2.4 Manifold Representation

If an $n \times m$ image is flattened into a vector, then this image can be thought of as a point in nm-dimensional image space. Even though the set of all images fills this space, the set of all images of faces does not; it fills a lower-dimensional manifold embedded within image space [19]. Intuitively, we can think of the space of all faces as covering a two-dimensional manifold parameterized by the two angles of rotation. This is not entirely true in practice due to changes in expression and differences between individuals, but the true surface can be approximated in two dimensions.

Manifolds have been used extensively in phoneme classification [3] (by defining a classification function on the manifold) and face recognition [18, 19] (by matching a set of images to the closest individual manifold). However, their use in visual tracking applications is limited.

The probabilistic appearance manifold [18] has been used as a successful tool for face recognition. This manifold is modeled as a collection of pose manifolds and the probabilities of transitions between poses. Submanifolds are created by clustering similar images from short video clips and building a manifold for each cluster. Transition probabilities between submanifolds are estimated from the transitions that take place in the video clips.

We intend to use a similar manifold here as an anchor for our tracking application. We construct the manifold ahead of time and assure that the tracker does not wander too far from the surface of the manifold using a regularization parameter. If the manifold contains sufficient detail regarding the set of possible configurations, then the true head cannot lie at a substantial distance from the surface, and it will be more easily detected.

3 The Approach

For the sake of computational efficiency, the algorithm is not run in the full image space. Even if images of heads are downsampled to 19x19 pixels (the smallest commonly used in research [18, 19]), image space is made up of 361 dimensions. Instead, each co-training classifier works in a 10 dimensional space where a point's coordinates represent the responses of ten Haar-like filters.

Haar-like filters are rectangular filters with positive and negative regions. The response of one of these filters is given by

$$r_i = \sum_{p \in P} I(p) - \sum_{n \in N} I(n)$$

where I is the image intensity matrix, P is the set of points in the positive regions and N is the set of points in the negative regions. These filters were developed by Viola and Jones [30], and are very quick to compute using the integral image. They also demonstrated that faces could be accurately detected by a small combination of these filters, which makes the ideal for this application. Examples of Haar filters are given in Figure 3 [30].

This reduction in dimension may appear drastic, but is necessary for tracking. In the typical classification setting, high dimension can be overcome with large amounts of data (although the Curse of Dimensionality can still be a problem). However, since our system has to adapt to quickly changing configurations as the head rotates, our training set must be small, and because of this, working in high dimension is infeasible.

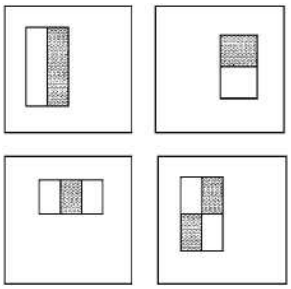

Figure 3: Four examples of Haar-like filters used to classify image windows as heads.

The algorithm for tracking with co-training and a manifold anchor assumes that faces must start in a frontal position. It proceeds as follows:

Before test data set is input:

1. **Find the optimal Haar filters for head tracking.** Since co-training requires two classifiers with independent views of the images, we must first find these views. Haar filter responses are ideal for this due to the built-in independence: vertically-aligned filters (Figure 3 upper-left) and horizontally-aligned filters (Figure 3 upper-right) give different information about the image. Also, vertically-aligned filters are naturally invariant to vertical motion. If a set of dark pixels move upwards (within certain limits), they will still fall within the same region of the filter and the response will be unchanged. The corresponding reasoning holds for horizontally-aligned filters.

   Each set of filters is composed of those that could best classify 1200 labeled images. Filters in the space of vertical filters had to discriminate between faces in vertical orientations (shown in Figure 4) and non-faces (shown in Figure 6), while horizontal filters had to discriminate between faces in horizontal orientations (Figure 5) and non-faces (Figure 6).

   Within a set, filters could only be selected if they had a small amount of overlap with each other.

Figure 4: Examples of positive images used to choose vertical-invariant filters.

2. **Construct the manifold anchor.** Given a set of images that (a) exactly frame the heads and (b)

Figure 5: Examples of positive images used to choose horizontal-invariant filters.

Figure 6: Examples of negative images used to choose most discriminative filters.

contain heads in regularly-spaced intervals. Unlike the standard probabilistic appearance manifold, we are injecting a large amount of a priori knowledge into the system. Not only are we picking the images which are representative of the manifold, but we define the adjacency relationships based on our knowledge of the data aquisition process.

For each frame during execution:

1. **Determine appropriate manifold neighborhood.** Assuming limits to the speed of rotation, the head in the current frame should be somewhat similar to the head in the previous frame. In the previous frame, say that the head was closest to manifold point p_{t-1}. In the current frame, we will only consider image windows which are similar to points in the manifold neighborhood of p_{t-1}.

   The size of the manifold neighborhood is a parameter of the application. The optimal value is selected empirically, and is discussed in the Results section.

2. **Evaluate Haar filters over a subset of image windows.** An image window is defined as any contiguous rectangular subimage. Since the total number of windows is quadratic in the number of image pixels, we do not consider all possible windows for the location of the head. In our 720x480 images, we evaluate the Haar filters at 86,000 images using the equation given above.

3. **Minimize regularization function.** For the set of all evaluated windows W, we minimize

   $$\arg\min_{w \in W} \left(||w - h_{t-1}||^2 + \alpha * min_{p \in P} ||w - p||^2 \right)$$

   where h_{t-1} is the window containing the head in the previous frame and P is the set of all points in the appropriate manifold neighborhood. Note that all operations are performed in the space of filter responses.

4. **Perform co-training on the optimal window.** Using linear regression in the two feature spaces, determine how certain each of the classifiers are about the new window. If one is certain of the positive classification outside the co-training margin, update the training set with the new point. If the classifiers are not certain, begin the search for a new head.

5. **Set parameters for next iteration.** Find the linear regression parameters for the new training set and record p_t, the point in the manifold most similar to the new head.

4 Nearest Neighbor

One potential place for optimization lies in the regularization function. Evaluating this function requires finding the distance from the candidate window to the manifold, which amounts to a nearest neighbor calculation.

4.1 Naive Search

Given the set of points P in a manifold neighborhood and a target point w, our goal is to find the point in P that is closest to w. The simplest algorithm for finding such a point is a simple search: calculate the distance between w and each point in P. Maintain a variable containing the minimum distance discovered so far, and return the index of that minimum point when all points have been examined.

This algorithm requires constant space and $O(|P|)$ time, as each point in the neighborhood needs to be examined. If we are willing to sacrifice space, there are algorithms that can perform faster asymptotically. Two of them are discussed below.

4.2 kd-Tree Search

The problem of range search and nearest neighbor search are closely related. For instance, a kd-tree can be used to efficiently find a target point w's nearest neighbor in P, but it requires linear space. See Figure 7 for an example of this process [22].

Given the two-dimensional kd-tree constructed in the figure, find a first approximation for the nearest neighbor by searching for the leaf node which would contain the target point. Use the point p_i contained therein as this approximation. p_i may not be the nearest neighbor, but we do know that the nearest neighbor must lie at least as close, and is therefore within the circle defined by w and p_i. We next back up to the parent of the current node. We then check to see whether a closer point could exist in the parent node's

other child. If so, proceed down the tree and see if the next point is closer. Recompute the circle if necessary. If a closer point could not exist in p_i's sibling, move up the tree and repeat.

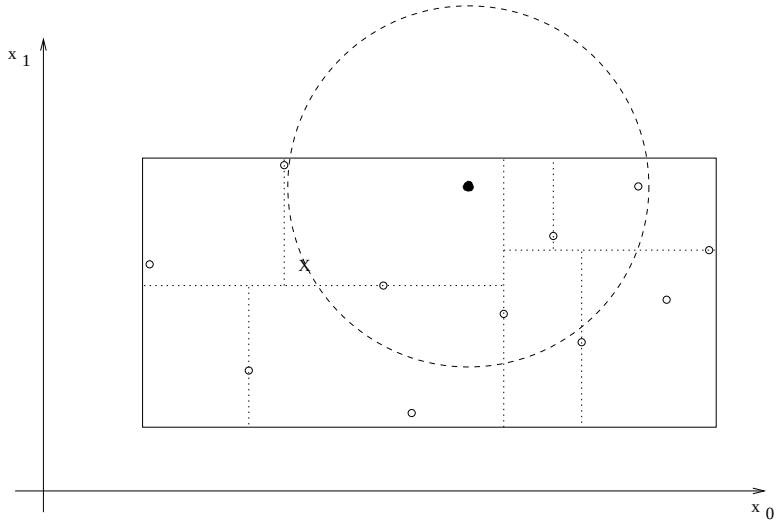

Figure 7: An illustration of one step in the nearest neighbor search within a 2D kd-tree. The X represents the target point, and the black point represents its closest neighbor in the target region.

It is clear from this description that at least $\Omega(\lg |P|)$ nodes will be inspected by this algorithm; at least one leaf node will be examined. The number of nodes examined can also not exceed $|P|$. As it turns out, under certain assumptions the expected time for finding the nearest neighbor is $O(\lg |P|)$. Calculation of the expected number of inspections is difficult because it depends on the expected distribution of the points in the kd-tree, and in the distribution of the target points. The analysis can be found in [11].

In the higher-dimensional case where the dimension is denoted by d, the efficiency of kd-trees decreases. As the dimension of space increases, the ratio of the volume of a hypersphere to the volume of its enclosing hypercube decreases, implying that the search must cover extra space. A kd-tree region also has $2d$ sides, so as the dimension increases, the number of neighbors to be searched also increases.

In our problem setting, the kd-tree is operating in twenty-dimensional space. A kd-tree nearest neighbor search was implemented in our system, but on average the method took approximately 60% more time than the linear search. The slowdown was exacerbated by the fact that the manifold neighborhood changed at every timestep, requiring tree reconstruction.

4.3 Approximate Nearest Neighbor

Another approach to the nearest neighbor problem is to settle for an approximate solution. The problem appears in such a wide variety of domains that many researchers have studied it, and multiple algorithms have been developed. Some employ random projections into lower-dimensional spaces [17], some use locality-sensitive hashing [14], and others build divide-and-conquer structures such as kd-trees or bd-trees [1].

One implementation of an approximate nearest neighbor algorithm is ANN, provided by David M. Mount and Sunil Arya of the University of Maryland [1]. Unfortunately, when given the data from our tracking problem's twenty-dimensional space, the implementation required more time than the linear search. Even when approximation was allowed, the time still exceeded that of the naive method. Based on these preliminary results, the implementation was not integrated with the co-training application.

Despite the best of intentions, it appears that advanced data structures do not empirically improve the efficiency of this search. Given all of this, simple linear search was implemented in our algorithm.

5 Experimental Results

Experiments were performed on one individual (myself) due to the time requirements of data collection. The manifold was constructed from carefully collected poses of this individual's head, and the test and training data were captured from the same individual at different time periods.

To have a source of ground truth, the test data was labeled by hand. On a select set of frames (referred to as "keyframes"), the true window containing the head was recorded. When a keyframe occurs during execution, the application compares the true window with the window selected by the tracker and records the results.

The performance for a given regularization parameter can be measured by the following formula [29]:

$$S = \frac{1}{|M|} \sum_{m \in M} \frac{1}{|L_m|} \sum_{t \in L_m} \left(\frac{2A_{overlap,t}}{A_{true,t} + A_{estimate,t}} \right) \in [0, 1]$$

where $A_{true,t}$ is the area of the head's true window at timestep t, $A_{estimate,t}$ is the area of the estimated head window at timestep t, and $A_{overlap,t}$ is the area of the overlap between the two windows. M is the set of independent runs of the tracker, and L_m is the set of keyframes in a run m.

For this project, a series of ten independent runs on difficult video segments were performed for each parameter setting. The runs contained head translations and rotations in a variety of directions, and they contained 55 keyframes.

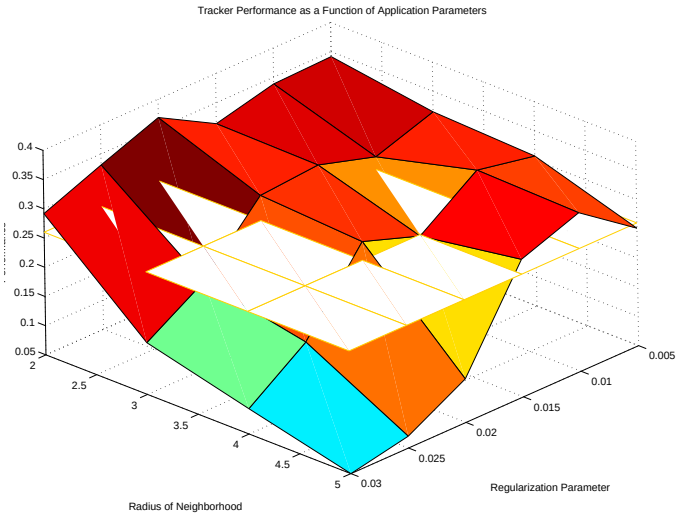

Figure 8: Results of procedure used to find optimal value of manifold anchor parameters.

Without a manifold anchor, the co-training tracker performed with an average overlap of 27.67 percent. Parameters were modified along two dimensions (regularization parameter and size of manifold neighborhood), and the results are given in Figure 8. For many parameter settings, the manifold anchor tracker outperformed the standard tracker. The peak performance was 36.87 percent, though it is unclear whether a more thorough search of parameter space (given more time) would yield better performance.

Performance of this method is compared with other methods in Figure 9. Particle filters and annealed particle filters are commonly used for problems of this type, and the graph demonstrates that the proper choice of parameters allows us to exceed their performance. However, some of the newer state-of-the-art trackers outperform our method. The results for Variational Tracking are presented in the graph. These results are drawn from [**?**], have been run on data sets with similar clutter and motion, and assume a sample size of 15. Given more data, these methods can exceed the performance shown here.

6 Conclusions and Future Work

While the results from these experiments do not reach the level of the best head trackers, this does not mean that the line of questioning should be dropped. It is encouraging that the addition of a manifold anchor increases performance, and it is unclear that these gains are the greatest that can be achieved.

Many simple changes could have an impact on performance. For one, a larger set of manifold points may form a better approximation to the true manifold. Fine-tuning the paramaters may also help, as the

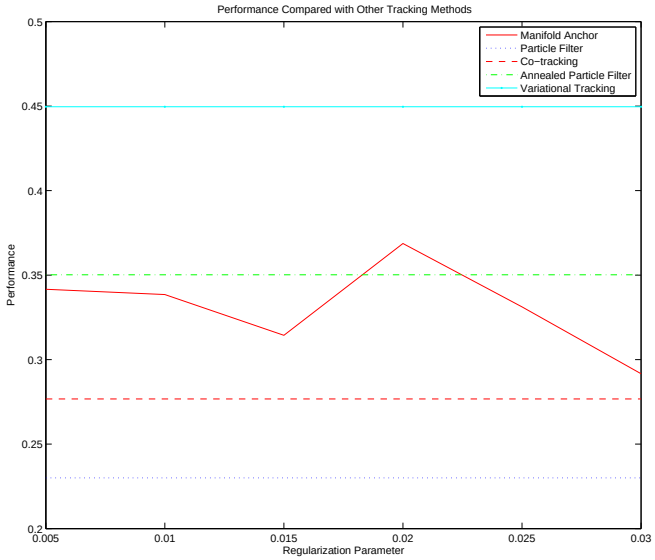

Figure 9: Performance across multiple tracker types. This graph is given for a fixed neighborhood radius of 2.

paramaeter space is quite large. However, spending a significant amount of this would be a waste.

The primary problem is most likely the filter selection. It is questionable whether twenty total Haar filters are sufficient to detect heads in all configurations. Good performance has been shown for faces, but profiles and the backs of heads are significantly different. A larger number may improve performance, but a different type of filter altogether may be necessary.

Thinking optimistically, if performance could be increased further, then the applications of this are widespread. It would even be feasible to construct a system that tracked many individuals; independent research groups have shown that once lighting and albedo effects are removed, there is a general face manifold that does not depend on identity [9, 12]. If such a manifold could be used, then this application would be much more general.

If such a system is to be used in the real world, then speed is also an issue. In such a high dimensional space, it does not appear that advanced nearest-neighbor algorithms offer much improvement. The search for the optimal window and the minimization of the regularization function take the most time; decreasing the resolution of the image or searching over a smaller number of windows may be the best means of reducing this.

It is still unclear how far I will pursue this research, but I feel that this look into the manifold structure of faces has been useful. It is possible that such concepts will be useful in many other Computer Vision

applications.

References

[1] S. Arya and D. M. Mount. Approximate nearest neighbor searching. In *Proc. 4th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 271–280, 1993.

[2] Maria-Florina Balcan, Avrim Blum, and Ke Yang. Co-training and expansion: Towards bridging theory and practice. In *NIPS 2004*, 2004.

[3] Mikhail Belkin and Partha Niyogi. Semi-supervised learning on riemannian manifolds. *Machine Learning*, 56:209–239, 2004.

[4] Stan Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *Proceedings of CVPR*, pages 232–237, 1998.

[5] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of ICML*, 2001.

[6] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, New York, NY, USA, 1998. ACM Press.

[7] Ira Cohen, Fabio G. Cozman, Nicu Sebe, Marcelo C. Cirelo, and Thomas S. Huang. Semisupervised learning of classifiers: Theory, algorithms, and their application to human-computer interaction. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 26, pages 1553–1567, 2004.

[8] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[9] Ian Craw, Nicholas Costen, Takashi Kato, and Shigeru Akamatsu. How should we representfaces for automatic recognition? *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(8):725–736, 1999.

[10] Sanjoy Dasgupta, Michael L. Littman, and David A. McAllester. Pac generalization bounds for co-training. In *NIPS 2001*, pages 375–382, 2001.

[11] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. In *ACM Transactions on Mathematical Software*, volume 3(3), pages 209–226, 1977.

[12] Ralph Gross, Iain Matthews, and Simon Baker. Generic vs. person specific active appearance models. *Image and Vision Computing*, 23(11):1080–1093, November 2005.

[13] E. Hjelmas and B. K. Low. Face detection: A survey. *Computer Vision and Image Understanding*, 83:236–274, 2001.

[14] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of 30th STOC*, pages 604–613, 1998.

[15] T. Joachims. Transductive learning via spectral graph partitioning. In *Proceedings of the International Conference on Machine Learning*, 2003.

[16] M. J. Jones and J. M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):8196, 2002.

[17] Jon M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 599–608, New York, NY, USA, 1997. ACM Press.

[18] Kuang-Chih Lee, Jeffrey Ho, Ming-Hsuan Yang, and David Kriegman. Video-based face recognition using probabilistic appearance manifolds. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03)*, volume 1, page 313, 2003.

[19] Kuang-Chih Lee and David Kriegman. Online learning of probabilistic appearance manifolds for video-based recognition and tracking. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 852–859, Washington, DC, USA, 2005. IEEE Computer Society.

[20] Anat Levin, Paul Viola, and Yoav Freund. Unsupervised improvement of visual detectors using co-training. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, 2003.

[21] Beatriz Maeireizo, Diane Litman, and Rebecca Hwa. Co-training for predicting emotions with spoken dialogue data. In *The Companion Volume to the Proceedings of 42nd Annual Meeting of the Association for Computational Linguistics*, pages 202–205, Barcelona, Spain, 2004.

[22] A. Moore. A tutorial on kd-trees. *University of Cambridge Computer Laboratory Technical Report No. 209*, 1991.

[23] Kamal Nigam and Riyad Ghani. Analyzing the effectiveness and applicability of co-training. In *Ninth International Conference on Information and Knowledge Management*, pages 86–93, 2000.

[24] David Pierce and Claire Cardie. Limitations of co-training for natural language learning from large datasets. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 2001.

[25] E. Riloff, J. Wiebe, and T. Wilson. Learning subjective nouns using extraction pattern bootstrapping. In *Proceedings of the Seventh Conference on Natural Language Learning*, 2003.

[26] C. Rosenberg, M. Hebert, and H. Schneiderman. Semi-supervised self-training of object detection models. In *Seventh IEEE Workshop on Applications of Computer Vision*, 2005.

[27] Shin'ichi Satoh, Yuichi Nakamura, and Takeo Kanade. Name-It: Naming and detecting faces in news videos. *IEEE MultiMedia*, 6(1):22–35, 1999.

[28] Henry Schneiderman and Takeo Kanade. A histogram-based method for detection of faces and cars. In *Proceedings of the 2000 International Conference on Image Processing (ICIP '00)*, volume 3, pages 504 – 507, September 2000.

[29] Jaco Vermaak, Neil D. Lawrence, and Patrick Perez. Variational inference for visual tracking. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'03)*, 2003.

[30] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[31] G. Yang and T.S. Huang. Human face detection in a complex background. *Pattern Recognition*, 27(1):53–63, 1994.

[32] Ming-Hsuan Yang, David Kriegman, and Narendra Ahuja. Detecting faces in images: A survey. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 24, pages 34–58, 2002.

[33] Xiaojin Zhu. *Semi-Supervised Learning with Graphs*. PhD thesis, Carnegie Mellon University, 2005. CMU-LTI-05-192.