CS 231A Project Report

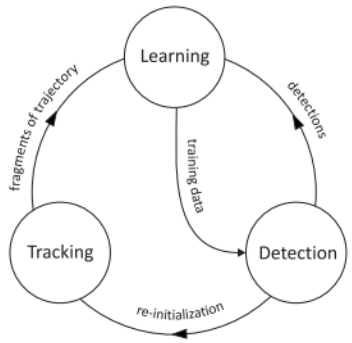
Title: Implementation of TLD Long-term tracking framework.
Authors: Jamie Ray, Jason Jong, Paul Chen

Abstract:
Long term tracking strives to locate designated objects in a video stream, while the object may change in scale, rotation, illumination, or become occluded. In this project, we implemented the TLD framework as proposed in 2010 by Kalal et al. [1] This novel approach combines the outputs of a tracker and a detector in order to produce a more robust system. The tracker's output is used to correct the detector if necessary, and also allows the detector to learn additional viewpoints of the object. Our results appear similar to that of Kalal.

Section 1 - Introduction:

We are interested in the long term tracking problem in computer vision in which an object in a video stream is tracked over time [11]. The object may change in scale, rotation, illumination, or even be partially occluded; where the goal is to recognize and track the object when it is present in the video stream. Preferably, a detector can process the video in real time and reliably track the object for an indefinite amount of time.

A working solution for the long-term tracking problem has many interesting applications. If an object of interest appears several times in a video, one could avoid having to manually scan the entire video and instead generate a timeline of when it can be found so the user can focus on those 'interesting' times. A wildlife camera looking for pandas could be processed to flag or report panda sightings for panda fans. We might imagine a multiple-tracking system that can tag each object so that users could easily find their favorite actor, player, car in a video, etc. Other potential applications include aerial surveillance by autonomous drones, video object stabilization, or automated scientific data collection.



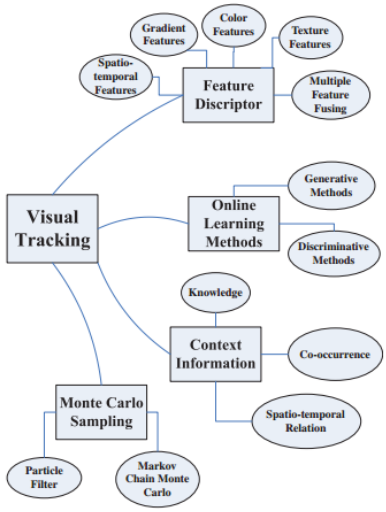
In 2010, Kalal et al. proposed the Tracking-Learning-Detection (TLD) framework [1] that synthesizes recent advances in both trajectory trackers and object detectors in a novel semi-supervised learning framework. In essence, given an initial bounding box of the object to track, TLD continually augments the detector with the new information from the incoming video with the help of the tracker.

The TLD framework achieved impressive results, but still has several limitations such as out-of-plane rotations, articulated objects, and single object tracking. We attempted to reproduce TLD's results in this project, and in the process gained further insights for potential improvements. There exists an open source implementation, but by building as much as we can ourselves we can learn about methods in computer vision, using the open source version [3] as a benchmark.

Section 2 - Review of previous work:

Object tracking is a decades-old computer vision problem with a variety of subtasks to be addressed [11]. It is naturally suited to use both geometry and appearance information. Recently, however, advances in computing power have allowed much more sophisticated algorithms to track at or near frame rates. One can categorize this variety of approaches in several ways, although the latest trackers tend to combine several concepts or techniques. All must choose an appearance model to represent the object and background - the type of features used to describe the object, and distinguish it from background patches. This aspect of the problem is similar to object detection in static images, and trackers have used correspondingly more complex descriptors (HOG, LBP, Gabor filters) as they have become successful in object detection. Others have realized that even richer features can describe the additional temporal information based on optical flow (HOG/HOF), background subtraction (eigenbackground), or explicit 3D modeling (HOG3D). There are also a variety of motion models that impose assumptions about physically allowed changes - smooth motion, constant velocity or acceleration, etc. Finally, the

challenge of object tracking is in the range of changes over time that a tracker should be able to handle - occlusion, intensity change, motion across the background, rotations, and more. Thus, successful methods attempt to combine descriptions and learning methods to optimally distinguish object from background in a general way (that is, robust to these imaging variations). This usually involves online learning for adaptability to new views.



Early trackers include Kalman filter approaches (a probabilistic model which assumes a Gaussian distribution in state space, which in this case is the position/motion) and point correspondence with constraints like the KLT (Kanade-Lucas-Tomasi) tracker (finding interest points using structure matrix and searching for similar points in the next frame). The detection component often used variants of simple sliding-window cross-correlation. A variety of more complex approaches have found success more recently. Most (including TLD) use multiple features to describe the image. Learning systems include online SVMs or random forests, as well as Monte Carlo-based methods like particle filtering. Clearly, the space of tracking algorithms is quite large, with a diversity of representations and strategies that attempt to optimize for different parts of the complex tracking problem.

However, some of the closest competitors to TLD include PROST [8], ALIEN [9], and Compressive tracking [10]. PROST (Parallel Robust Online Simple Tracking) is quite similar in spirit to TLD, and tries to address the ‘dilemma’ that adaptability creates - namely, without enough adaptation the system will fail when the object’s appearance changes, but with too much it can overfit to a variety of noise sources, degrading performance or causing drift [8]. As the degree of adaptability is difficult to parametrize and may depend on the application, PROST combines components with increasing amounts of adaptation: cross-correlation with the initial object view (no adaptability), online random forest detector, and optical flow tracking. The only significant difference of TLD is in the middle component. Rather than using Haar features for base learners, TLD uses raw pixel values. Furthermore, it updates the initial object view with new patches to be used in cross-correlation (injecting further adaptability), pre-filters using a sliding window variance threshold, and trains the trees differently. PROST learns tree structure online, while TLD chooses 13-deep trees using splits on random feature values, and only learns the leaf probabilities online. It seems that these moderate changes, which increase adaptation of the cross-correlation filter while reducing that of the random forest classifier, produce better performance, probably because they can improve the quality of sampled training data and use it more effectively.

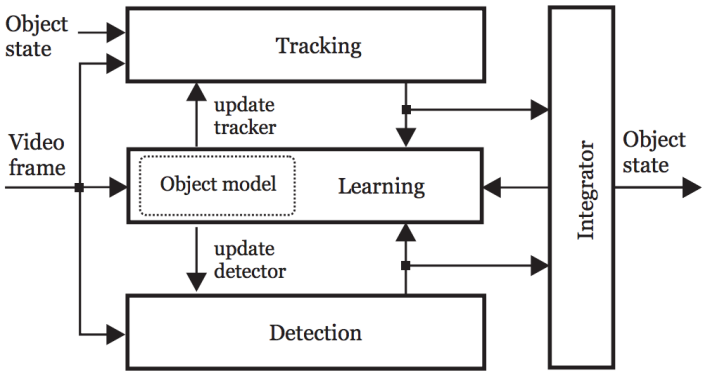
While PROST targets the adaptability problem, ALIEN (Appearance Learning In Evidential Nuisance) focuses on dealing with non-invertible noise for which invariant features can’t be constructed (occlusion is the dominant example) [9]. They note some poorly motivated choices of many trackers. Boosted versions [2] often assume iid data and good labeling, which is far from the truth of semi-supervised online learning. In addition, the object bounding box often includes background pixels, which if used can degrade the matching (and movement across the background can substantially change the ‘object’ representation). To address these concerns, the authors store oversampled SIFT features in an attempt to describe local regions of the object across multiple views, which can be matched to query patches in a RANSAC-like way to estimate geometric transformation between frames. They omit a motion estimation component, and instead restrict the motion by trying to detect the object anywhere within a radius of the prior location. In essence, this is a feature matching approach (using SIFT features and the standard ratio test) which restricts good matches using a shape model, compares matches to object and ‘context’ (background) features to detect occlusion, and updates by trying to find discriminative features (appearing only in the object or the context, but not both). This seems to achieve success similar to TLD with a very different approach, and it would be interesting to investigate whether strengths of the two can be combined.

A third successful tracking approach (called Compressive Tracking, or CT) is similar in that it tries to perform detection within a local region around the prior location [10]. It is based on the theory of compressed sensing, citing theoretical properties of compressing patch descriptors using random sparse matrices and the

ability to discriminate between the compressed descriptors. As with ALIEN, the motion model is quite simple - just look in the area around the last location and pick the highest-scoring patch. They use a large set of Haar-like features, whose localization can give some robustness to occlusion, and compress to only 50 dimensions. The random projection also guarantees (with high probability) independent gaussian distributions of the elements, allowing the authors to train a Naive Bayes classifier on the compressed vectors online by simply estimating means and variances of the individual gaussians.

These three state of the art trackers demonstrate some of the variability in terms of tracking theory and approaches to object representation and detection. They focus on different sub-parts of the tracking problem, and it would be nice to investigate whether their solutions can be integrated into the TLD framework. It is difficult to judge which tracker is the 'best,' given their different strengths and the fact that they were tested on different datasets (probably playing to these strengths). However, we found TLD to be a compelling solution with value as a learning experience.

Section 3.1 - Technical Summary:



We developed the system using MATLAB for its convenience in prototyping and visualization (although one strength of the algorithm is a complexity footprint small enough to work in real time and/or on mobile devices). The authors model the object as a set of size-normalized patches of constant aspect ratio. Some of these patches truly represent the object, while others are patches known to be background. By comparing patches in a new frame with this model, the system can predict locations for the object and compare them with tracking predictions. It is easy to update using good or bad patches found in future frames.

The tracker performs the task of estimating the motion of an object between two frames. The authors choose to estimate motion of points within the object's bounding box using a Lucas-Kanade tracker and then choose the median translation vector as their prediction (this is known as Median Flow), we followed suit. Then, while each point's motion is estimated independently, we have to use all these predictions to generate a transformed bounding box in the new frame. In this approach, the old box is essentially translated by the median of the per-point motion vectors (unless this motion is too large, in which case the tracker output that it has failed).

The learning component is defined by two distinct goals of the detector (which are addressed by separate 'agents') - to correctly localize all appearances of the desired object, and to avoid predicting a location at which it doesn't appear (recall and precision). The PN learning framework leverages structure in the video stream to harvest examples that can help with each of these goals. Namely, this structure is the spatiotemporal continuity of object locations under conditions on the reliability of a track. New positive (P) examples come from the motion of the object through frames, which is estimated by the tracker and can thus find views that may not be classified correctly by the detector. Similarly, new negatives (N) come from the notion that the object is unique and can't appear in a drastically different position if the track is reliable, so patches far from the track location are taken as negatives. Thus, it is important to note that the PN agents may help the detector discriminate a unique object from other objects of the same class that appear elsewhere in the frame. A track is termed 'reliable' if it has ever contained a patch that was well-matched (at some threshold) with an early part (e.g. first half) of the object model (set of positive patches collected over time). Thus, it leverages the output of the tracker to choose potential patches for training the detector, and the output of the detector to decide whether a track is 'reliable' enough to learn from. It might be interesting to try to modify the learners. For example, we could try to help the agents realize when other objects of the same class appear and flag them as positive instead of negative. We could add different ways to measure the reliability, or have a parametrized tracker that could be updated via detector output.

The detector takes a cascaded approach to quickly eliminate the most unlikely swaths of the frame. The

first stage examines patches of the same aspect ratio as the object in a pyramid of locations and scales throughout the image. It computes their variances, and discards any patch whose grayscale variance is significantly different from the model patches (computing the variance with integral images). The next stage is an ensemble classifier that averages predictions of base classifiers. Each base classifier is built of a set of binary pixel comparisons within the patch (15x15 patch, 13 comparisons per classifier in the paper), and the bit vector encoded by the comparisons is used to index into a posterior distribution of detection probability. The ensemble score is thresholded, and high scores are passed to the final stage, a nearest-neighbor approach using the model patches. The distance is measured by normalized correlation coefficient between the query patch and the model patches, and the output score is a 'relative similarity' which compares the correlation with the best-matching positive patch to the best-matching negative patch. The patch with highest relative similarity is then the detector's prediction of the new object location.

Section 3.2 - Technical Details:

Tracker

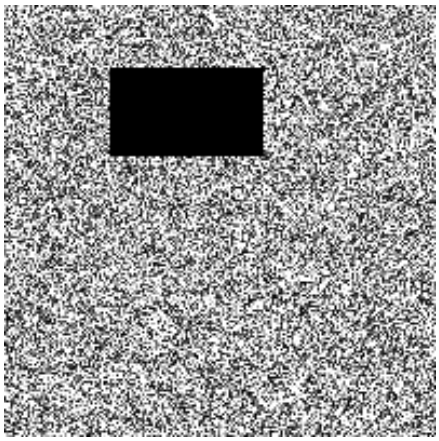
We are using the Lucas-Kanade tracker with image pyramid representations based on an implementation by Edward Wiggin [4]. The tracker is given a seed bounding box from the previous frame as output by the learner. Next we pick points within the bounding box to track. We first did this by uniformly sampling points within the box, and also by corner points found by the Shi-Tomasi corner detection method; then the flow vector for each point is estimated. We then calculate the median optical flow; we found the most success with the median of the central fifty percentile. We implemented failure detection by setting a threshold for how large the median flow vector can be. If the flow vector is too large, it means the tracker is not going to be reliable, so we indicate that in the output. This is to account for occlusions or when the object moves off screen. Finally, if the tracker did not fail, we shift the position of the previous bounding box by the median flow vector to estimate the next bounding box. In the image below, the green bounding box is the previous frame, and the blue points are the corners found by the Shi-Tomasi method. The red bounding box is the position of next frame as estimated by the Lucas Kanade algorithm. As we can see, the motorcyclist is moving upwards in the image.

Detector

Our priority for the detector is a faithful reproduction of the pipeline described in the TLD report without using external libraries or toolboxes. This approach yielded better understanding of the detector's performance and some insight into failure modes, as well as a variety of ideas for simple extensions to the methods used. The detection component uses a multi-scale sliding window approach to try to find patches (of constant aspect ratio) in the image that closely match prior views of the object. The scale is sampled in factors of 1.2, and the horizontal and vertical position in steps of .1 * patch size in that dimension. Thus, even for a small frame, we may need to examine of order 10^5 patches, making frame-rate computation challenging. To address this complexity issue, we use a cascaded approach that aims to filter out (reject) a large number of candidate patches in the early stages, which are less computationally intensive, before spending more operations on the most likely patches.

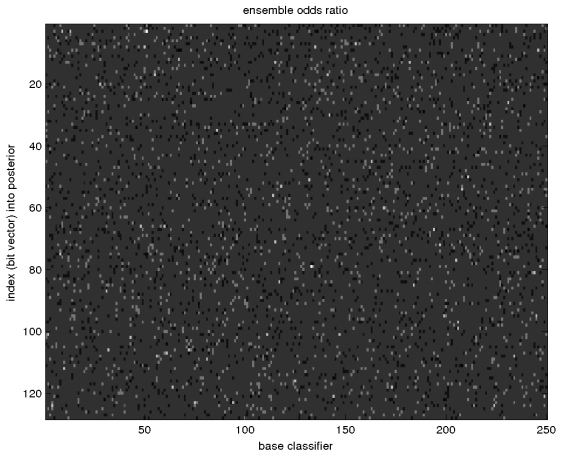
The first stage is a simple variance-based filter. In the TLD paper, this stage requires patches to have variance at least half that of the initial patch. As this piece is responsible for reducing the candidate set to a manageable number, it must spend very little time per frame. After building a naïve (and obviously much too slow)

version, we switched to using integral images over the normalized grayscale pixel intensities (and squared intensities). These allow efficient computation of the expected intensity and squared intensity over any patch (using one additions and two subtractions each). We later optimized further, noticing that patches are tiled horizontally and vertically over the image. This means that a group of patches with only horizontal offsets covers a set of rows in the image, such that variance of these patches only depends on intensities in that set of rows. Thus, for computing patch variance in this group, we need only the integral of those rows, and can compute mean values with a single subtraction.



While the variance filter as described can effectively prune a large number of candidate patches (usually > 90% in testing, although highly dependent on the the complexity of the object to be tracked relative to the image), some simple changes could help with performance. One option is to also set an upper limit on the patch variance. The toy example illustrates the value of this approach – suppose we want to track a dark rectangle moving across a background of white noise. Then we are really searching for a patch of very low variance, while the background has high variance. The initial approach cannot filter any patches in this scenario, leading to considerable slowdown in the later stages. More realistic examples of this phenomenon can be imagined, such as a uniform-intensity boat on choppy water, or a hand moving across wallpaper. A second problem is that this stage doesn't seem to be able to learn from new appearances, at least as described. Thus, it will have difficulty tracking changes in the statistics of the object's intensity over time, which might be caused by varying illumination, non-Lambertian reflections, rotation out of plane, or occlusion. The last is quite difficult to address, but it would be interesting to explore a more sophisticated variance filter that can also learn the variances of non-object patches to better recognize occlusion. However, the first three are all variations we would like to be able to deal with, and simply recording variances of new training patches can help. We thought about several methods to learn from new patches, and decided to take a weighted mean of their variance with the prior variance estimate. However, later work will explore a better system for training this stage that will find a lower and upper variance threshold that best separate the object from the background in a particular frame, then average these with previous thresholds. Finally, we note that there are still many simple object-background pairs that this method fails to discriminate, resulting in too many candidates passed to the next part of the cascade. One is a mostly-dark object with a bright spot (maybe a car with headlights) on a textured background, and and another is an object with the same intensity as the background but different color. It is expensive to use positional information, which is reserved for later stages, but these problems can be targeted by a more sophisticated description of the pixel distribution in a patch. Color histograms are a fairly popular tracking method [5-7], so we built a follow-on stage that compares coarse color histograms to do additional filtering. The current version uses Bhattacharya and chi squared distances between weighted histograms of pixel hues from the HSV color space, with the weight related to saturation so that hues with low saturation don't contribute as much [7]. Once adequate labeled testing data is acquired, we plan to quantify the ability of these enhancements to improve tracking performance.

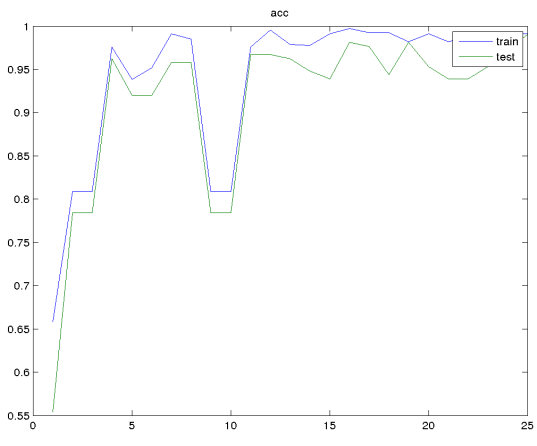
The second stage of the cascade is the 'ensemble classifier,' which aggregates the results from a set of base classifiers, each of which performs binary comparison of several horizontal or vertical pairs of pixels in the query patch. The result of these comparisons yields a bit vector (one bit per pair) that can index into a probability table describing the posterior probability that the patch contains the object given the observed bit vector. These posteriors are based on counts of that bit vector (which can be considered



as a descriptor for the patch) occurring in positive and negative training patches coming from the PN agents. The number of such pixel pairs is 13, although the reason for this choice isn't clear, and leads to a table with 2^13 = 8192 entries and therefore considerable sparsity. Nevertheless, it seems to perform reasonably, although varying this parameter is something we'd also like to test on labeled data. We originally chose to use 1000 base classifiers with 13 pairs each, but requiring 13000 comparisons per candidate patch. However, the system seems to perform equally well with a smaller number (250x8 = 2000 pairs), which seems to be a good choice as the cascade's speed is limited by this stage (taking > 80% of the time). We can also motivate this choice by realizing that each training patch only updates the counts at a single index, so until the number of training patches is significantly larger than 2000 (which would take quite some time) the counts will be fairly sparse - we show this effect by plotting the odds ratio and noting that the non-unity entries are sparse. The TLD paper doesn't specify the number of base classifiers, although we could assume that it uses each possible pixel comparison once, yielding around (w*(w-1)/2 * h) + (h*(h-1)/2 * w) pairs, or 5250 for a normalized patch size of 15x15. Obviously, an ensemble with a single base classifier may not extract enough information to filter well – it is worth examining different tracking situations and quantifying the number of base classifiers at which performance saturates (as well as the number of pairs per classifier). In the initial implementation, the probabilities from each base classifier are averaged to obtain an overall 'probability' of the object being in the patch, which is thresholded to reject patches with low probability. Training the ensemble is as simple as incrementing the counts of descriptors found in any training patch for which it predicts the incorrect label. In this way, the stage attempts to learn only from the difficult examples.

Again, understanding the detail of the ensemble classifier yields some insight into potential improvements. We note above that the ensemble score is taken to be a probability, but weights the votes of each base classifier equally. However, imagine we have two base classifiers – one which has observed many instances of its corresponding descriptor, and one that hasn't. It seems likely that we can get a better detection by trusting the more experienced base classifiers – i.e. the ones that have a larger number of counts in the corresponding entry of their probability table. Thus, we switched to a weighted average of the base probabilities, with the weight given by the number of counts. It is easy to see that this amounts to simply aggregating the positive and negative counts for a set of patches, so we can see the adjusted 'ensemble' as a single classifier that compute into potential improvements. Such a change of approach is probabilistically motivated if we can assume that a larger number of counts can give us greater confidence, which may be at fault if the object versus background appearance changes significantly. In that case a large number of counts could correspond to the old appearance, whereas a smaller number might signal a descriptor (bit vector) of the object's new appearance, thus making it more trustworthy. As for the variances above, we may take a weighted sum of previous and incoming counts to compute 'pseudo-counts' that can facilitate adjustment to new views of the object. Finally, the cascaded nature of the detector, use of binary comparison, and calculation of integral images in the previous stage immediately brings to mind a Haar cascade classifier (Viola Jones method). All the pieces were in place to compute binary descriptors using pixels in the integral images rather than the raw pixel values, and it was trivial to change the code to compare means over rectangular patches. This extension was recently implemented, and it would be interesting to do a more complete comparison with the raw pixel version, different options for quantizing the Haar features (the current binary quantization checks whether the computed value is positive or negative, but it might be valuable to use more thresholds), a full boosted Viola Jones style detector, or even a random projection of a larger feature space as done in the compressive tracker.

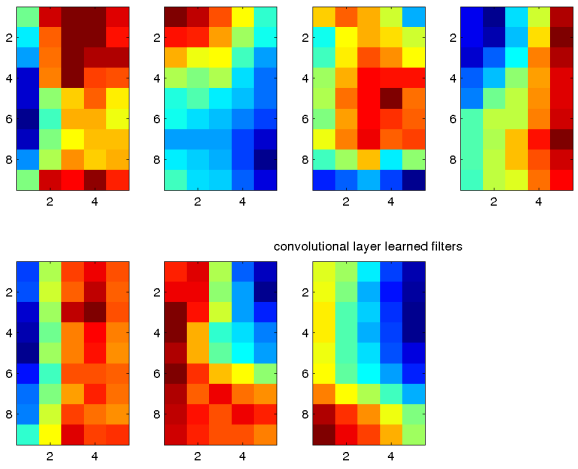
The third stage is designed to be both the most selective and computationally intensive, because it should be working to discriminate between a small group of difficult candidate patches. The TLD paper implements this stage as a modified nearest-neighbor classifier, computing normalized correlation coefficients between the query patch and all 'model' patches. The model consists of a set of patches (normalized to 15x15 pixels) that have been chosen by the PN experts as object (positive) or background



(negative). In essence, this stage finds the closest-matching positive and negative patches that have been previously observed, and compares the correlations. If the best correlation with a positive patch is much higher than the negative, the query is more likely to contain the object. The actual score used is termed 'relative similarity,' and defined as the ratio of the best positive correlation with the sum of the best positive and negative correlations (so that it ranges from 0 to 1). Any candidate patches with relative similarity greater than 0.6 are taken to be true observations of the object (from the detector's point of view) and have passed the cascade. Training is again fairly simple, and amounts to adding new patches to the set of model patches if they are incorrectly classified (or within some small margin of the classification boundary). If there are too many patches in the model, some are randomly removed or 'forgotten' to save memory and time.

As we learned in class, methods that operate on raw pixel data tend to be less robust to varying image conditions than those using transformed representations like SIFT. It seems, then, that the final stage is ripe to be upgraded to features or techniques with more power than what is only a small step up from a sliding-window cross-correlation search. One natural choice might be to generate e.g. HOG descriptors for each patch and compare those. However, given that the patches are already normalized to a constant and fairly small size, we thought it would be interesting (and computationally tractable) to apply a convolutional neural network as the final-stage classifier. This provided a great opportunity to learn more about CNN techniques, which have recently been very successful but saw little coverage in the course other than our guest lecture. It is also a good candidate for online learning, which is difficult for many learning methods but necessary to work with changing object views in the tracking problem. Ideally, the CNN can discriminate as successfully, and offer potentially better generalization with less memory overhead because the patches can be learned from without storing them. The approach we take is similar to the educational example of digit classification found at [12]. However, all of the code is written from scratch in an object-oriented way, rather than using the provided starter code. It includes neuronal layers capable of convolutional filtering, nonlinear pointwise activation, max-or-mean pooling, fully-connected information propagation, and a variety of loss functions for the output layer. Moreover, based on difficulty with initialization and slow training, we replaced the standard sigmoid with a 'softsign' activation and accompanying weight protocols as suggested by [13]. The current version of the CNN consists of a convolutional layer with 7 9x5 learned filters (the same aspect ratio as the patch), a softsign nonlinearity, a mean pooling layer over 5x3 patches, a fully-connected layer with softsign nonlinearity, and a logistic loss output. It is trained via minibatch SGD with momentum, this time implemented using the starter code from UFLDL [12]. We hypothesize that a well-tuned CNN should be able to generalize better to new views and learn the relevant features distinguishing object from background, and thus may outperform the normalized correlation approach in testing. However, it is difficult to design the layer architecture and tune parameters, especially for the online learning. We show the training and test accuracy for this network (using the initial training patches), noting that it converges to high accuracy. The nearest-neighbor approach achieves 100% accuracy, but the difference may not be significant. We also display some of the learned filters to verify that they can capably describe interesting features of the object - in this case, they appear to compare (red vs blue) blobs of different sizes and orientations.

Unfortunately, while the simplicity of the detection cascade provides a great starting point for more sophisticated enhancements, a few of which have been explored above, the performance in the videos doesn't seem to be limited by the detector output. This can be observed on our primary video, where the initial detector is able to find the object when it is in frame and localizes better than the full TLD pipeline. For this example, performance seems more affected by significant motion, which hurts the tracker, but not the detector unless blur changes the appearance. Furthermore, with a lack of labeled data it is difficult to characterize the potential benefits of the extensions described above. Thus, the quantitative comparison of different detection methods will be carried out in future work. In each



case, we take what is a very simple filter and modify the type of features used or the way the decision is made while preserving the general approach of that stage – these include filtering using patch statistics (intensity but no position information), binary comparisons (position-dependent, but only weakly encoding intensities), and finally a high-fidelity classification that relies on all pixels in the patch (both position and intensity). We believe that each is well-motivated by the material covered in class or other successful methods in computer vision, coupled with an understanding of the detection internals gained by faithful reimplementation.

Learner

The learning component of the TLD tracking framework relies on the integration of the tracker and detecting boxes. Once the tracker and detector outputs their best bounding boxes, we send these boxes to an integrator. This integrator finds the detector box with the largest "overlap" with the current tracking box. To find this overlap, we use something similar to Jaccard distance: we divide the intersection of the areas of these two bounding boxes by the total area that these two boxes encompass. If the "overlap" of these two boxes exceeds a certain threshold, we declare the integrated box as sufficiently detected and tracked, and we use the patch overlaying this box to help train the detector for better and more recent chances to the image.

As an example why this is necessary, the object we track will seldom maintain the same shape or colors as its original image. Thus, it is necessary to make constant updates to the detecting framework so that the new patches used to train the detector will be of an object relatively similar to one of the previous frame, but because of rotations, deformations, and scaling to the object, this new detector will just to take care of these modifications.

Once we have the best bounding box returned by the integrator and have been approved for learning, we generate multiple bounding boxes around the best bounding box. These boxes undergo rotation, translation, and scaling in order to generate a level of noise. We then take these set of bounding boxes and send them through experts: the P-expert and the N-expert. The P-expert focuses on the temporal structure of the image. The P-expert independently examines the randomly generated hypotheses and determines ones with enough overlap with the tracker. It assumes the tracker as truth and marks these as positive examples to be placed as a positive example in the detector. The N-expert focuses on spatial structure. Independent of the P-expert, the N-expert will take the patches that have been marked as positive by the detector and marks all other patches (those that should be positive, but were not matched) as negative. Thus, the P-expert focuses on false negatives while the N-expert focuses on false positives. With the combination of the two, the detector system becomes a flexible framework for detecting a moving and changing object.

Section 4 - Experiments:

In the following images, we have an example run of our TLD framework. Each image contains boxes of varying colors. The blue box represents the current tracker box. The magenta box represents the single bounding box outputted from the detector that most closely matched the tracker's box. As in, this is the box that will be used by the integrator to calculate the best bounding box, which is displayed in yellow. In the next frame, the tracker uses the previous best bounding box (previous yellow box) in order to track the next frame.

When the tracker and detector box have enough overlap, we know the resulting yellow best bounding box is sufficent to be used for learning. In these instances, we generate a set of hypotheses from the yellow bounding box and learn from these resulting boxes.

In certain circumstances, the tracker cannot even find the object anymore. In these situations, we assume that the object has gone off screen. In these situations, we specifically indicate that the object has not been detected. The yellow box is simply a representation of the best detector box found, but it is not a good representation in this case of object detection.

However, when the object reappears, we can see that the tracker will be constantly reinitialized with the yellow bounding box. We see here that the tracker reinitializes, and we can now continue to detect the object once again

Section 5 - Conclusion:

In this project, we were able to implement the Tracking Learning Detection framework for the long term tracking problem. Our implementation is able to successfully track objects in a video stream continuously while the object is still on screen. We are also able to detect when the object is occluded and then recover when the object returns on screen. We are also able to increase the robustness of the detector by automatically augmenting it with training data from additional viewpoints of the object as they are show in the video.

Section 6 - References:

1. Tracking-Learning-Detection. Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 6, NO. 1, JANUARY 2010 http://epubs.surrey.ac.uk/713800/1/Kalal-PAMI-2011%281%29.pdf
2. On-line Boosting Trackers. S Stalder, H Grabner, ETH Zurich. http://www.vision.ee.ethz.ch/boostingTrackers/
3. TLD Code and data: http://cmp.felk.cvut.cz/tld
4. Median flow tracker, Edward Wiggin. http://www.mathworks.com/matlabcentral/fileexchange/30822-lucas-kanade-tracker-with-pyramid-and-iteration
5. A Scale Adaptive Tracker Using Hybrid Color Histogram Matching Scheme. Nikhil Naik, Sanmay Patil, and Madhuri Joshi. *ICETET, page 279-284. IEEE Computer Society,* (*2009*) http://web.mit.edu/naik/www/tracking/naik_icetet_09.pdf
6. Color Feature Detection. T. Gevers, J. Weijer, H Stokman. http://staff.science.uva.nl/~gevers/pub/CIP06.pdf
7. Boosting Color Saliency. T. Gevers, J. Weijer, A. Bagdanov. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 2005 http://hal.archives-ouvertes.fr/docs/00/54/86/15/PDF/pami2005b.pdf
8. PROST: Parallel Robust Online Simple Tracking. Jakob Santner, Christian Leistner, Amir Saffari, Thomas Pock, Horst Bischof. *CVPR, page 723-730.IEEE,* (*2010*). http://gpu4vision.icg.tugraz.at/papers/2010/santner_cvpr2010.pdf
9. Object Tracking by Oversampling Local Features (ALIEN). Federico Pernici, Alberto Del Bimbo. http://www.micc.unifi.it/pernici/index_files/ALIEN_final.pdf
10. Realtime Compressive Tracking. Kaihua Zhang, Lei Zhang, and Ming-Hsuan Yang. ECCV 2012, Part III, LNCS 7574, pp. 866–879, 2012 http://www4.comp.polyu.edu.hk/~cslzhang/CT/eccv_ct_camera.pdf
11. Recent advances and trends in visual tracking: A review. Hanxuan Yang, Ling Shao, Feng Zheng, Liang Wang, and Zhan Song. *Neurocomputing 74(18):3823-3831*(*2011*) http://lshao.staff.shef.ac.uk/pub/Tracking_NeuCom2011.pdf
12. Convolutional Neural Network Exercise. Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen. http://ufldl.stanford.edu/tutorial/index.php/Exercise:_Convolutional_Neural_Network
13. Understanding the difficulty of training deep feedforward neural networks. Xavier Glorot, and Yoshua Bengio. *AISTATS, volume 9 of JMLR Proceedings, page 249-256.* http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_GlorotB10.pdf