A Reimplementation of Online Tracking by Learning Discriminative Saliency Map with Convolutional Neural Network

Aaron Gonzales
gonza647@msu.edu

Steven MW Hoffman
hoffm470@msu.edu

1. INTRODUCTION

We chose to implement our machine learning project with a *deep learning thrust*. We are basing our project on the work of Hong, et al. [1], and our goal is to reproduce their work through the creation of a viable demo. Hong, et al. propose a novel online tracking scheme for use in applications where tracking an object through frames of a video is desired. Although there are a large range of potential applications for object tracking software, it is still a difficult problem due to challenges of occlusion, pose variations, illumination changes, fast motion, and background clutter [1], and any potential tracking solution must have robust methods to overcome these challenges. This paper proposes to solve these problems using the combined utility of both a convolutional neural network (CNN) and support vector machine (SVM), wherein a discriminative saliency map is produced and used to calculate the posterior probability of the location of the target in the image. The object tracking algortihgm by Hong, et al. is described below and is portrayed in Figure 1.

The tracking algorithm proposed by [1] begins by first generating a set of sample images, each of which is drawn from candidate bounding boxes near where the target was located in the previous frame. Each of these sample images is passed through a pre-trained CNN [2]. A CNN is used because CNNs have been shown to be very successful at creating image representations useful for object discrimination. Additionally, CNNs have shown promise in overcoming challenges from many of the current difficulties presented in object tracking, including pose variations, illumination changes, and background clutter. For each image, the output from *the first fully-connected layer* of the network is extracted and is used as the feature vector describing that image sample. The image sample feature vector is then given to an SVM which will classify it as either a positive sample, including the object we are tracking, or a negative sample, which does not include the object we are tracking. In contrast to the CNN, which is learned offline on generic image data not specific to the target, the SVM is learned online using the samples it has seen up to the previous video frame. This allows the SVM to adapt to different types of objects which the user would like to track. For each positive sample, the target-specific features are extracted by using those features which corresponded to positive weights in the SVM, setting all other feature values to zero. The positive weights of the SVM are chosen because they are the weights which correspond to positively identifying a target. These target-specific features are then backpropagated through the CNN, producing an image containing a saliency map. A saliency map is created for every positive sample, and these are combined to make a final target-specific saliency map where larger values in the map indicate a larger likelihood that the target is located at that pixel. Through these means, the target can be segmented out of the image at a near pixel level. A generative model is then computed to refine the likelihood estimate based on what has been seen in previous frames. Finally, the posterior probability of each original sample containing the target is calculated, and the bounding box containing the highest posterior is selected as the target location. With the target successfully found, the algorithm begins anew in the next frame, creating candidate bounding boxes around where it just found the target in the preceding frame.

2. RELATED WORK

The problem of object tracking in video is a large domain, so we will restrict our discussion here to a few works which also attempted to use CNNs to perform tracking, as these are most relevant to the paper we have chosen by Hong, et al. [1]. We also highlight how the approach proposed by Hong, et al. differs from these approaches, making it a novel work.

[3] utilizes a CNN for tracking; however they use an offline trained CNN. They also require a separate class-specific network to track various other objects. Hong, et al [1], in contrast, proposes using a pre-trained CNN used for large scale image classification which is trained on generic image data. An online trained SVM is then used in conjunction with the CNN by Hong, et al. to learn the target specific information.

[4] also uses a pre-trained network where a stacked denoising autoencoder is trained using a large number of images to learn generic image features. However, as this network is trained on small grey images, its representation power is limited.

[5] proposed a target-specific CNN for tracking, where the CNN is trained online. However, this network is shallow in comparison with the deep CNN proposed by [1], and as such does not take advantage of the rich information a deep CNN provides.

In addition to the novelties described above, the tracking method proposed by [1] differs from all three of the above papers in a few important ways. First, it uses an online trained SVM with the offline trained CNN in order to adapt the tracking to whatever type of object the algorithm happens to be presented with. Secondly, it uses saliency maps

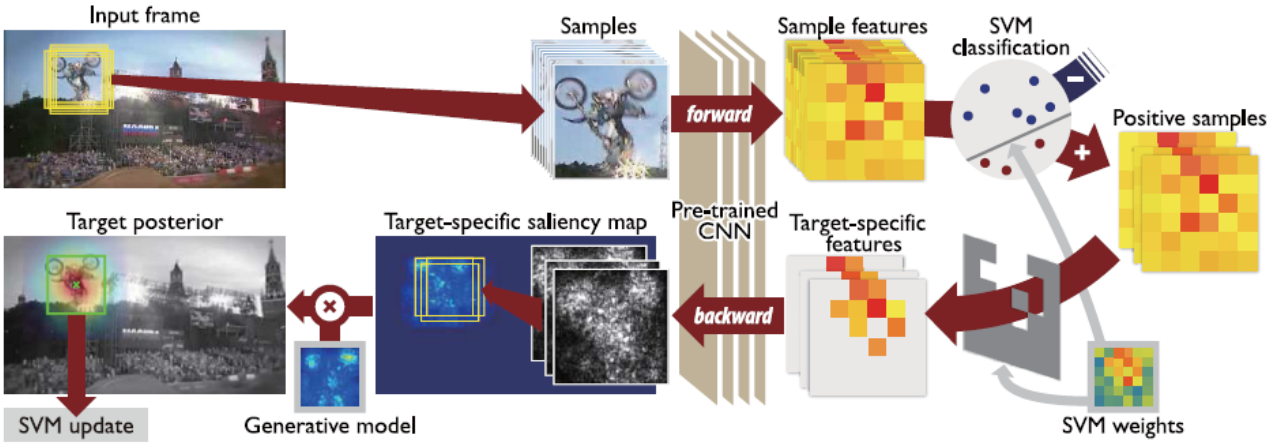


Figure 1: A pictoral description of the algorithm described by Hong, et al. in [1]. Candidate frames are passed through a pre-trained convolutional neural network (CNN) and a feature vector is extracted for each sample from the first fully-connected layer of the CNN. These feature vectors are passed through an SVM, retaining only samples which the SVM believes to contain the target object. The model weights of the SVM are used to determine the target-specific features of each positive sample, which are then back-propagated through the CNN to retrieve a target-specific saliency map. A generative model, produced from previous frames, is convolved with the target-specific saliency map, and the posterior probability that each of the original samples contained the target object is computed.

to find the precise location of the tracked object.

3. DATASET DESCRIPTION

Following the form of Hong, et al. [1], we tested the tracking algorithm against video sequences from the Tracker Benchmark v1.0 dataset [6]. This dataset contains 50 testing sequences, i.e. videos containing a specific object to track, all of which vary in video length. These sequences present a variety of tracking challenges such as illumination variation, deformation, motion blur, background clutter, etc, which can be seen in Table 1. In addition to these visual challenges, objects of interest vary from humans, to vehicles, to animals, and to various inanimate objects. Each of these sequences contain ground truth text files which contain bounding box information for objects of interest at each frame of the video. Some of the videos contain multiple objects of interest and as a result come with multiple ground truth files. We will analyze our tracking results on these videos qualitatively. Video frames from a selection of these testing sequences is shown in Figure 2.

4. TRACKING ALGORITHM

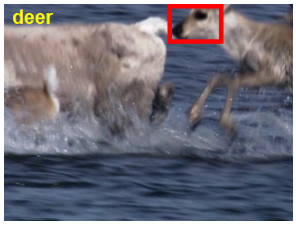
In the introduction, we provided a brief overview of the tracking algorithm proposed by Hong, et al. [1], which we have implemented for our project. In this section, we will go into more detail describing this algorithm and how we have implemented it. All of the code for this project was written in Matlab and was executed on both Matlab 2014b and Matlab 2015b. Note that Figure 1 provides a nice pictorial overview of the algorithm.

4.1 Generating Candidate Sample Patches

In order to search for the target object in the current frame, candidate sample patches need to be generated, i.e. small portions of the image frame cropped to the size of the target object's initial ground-truth bounding box. The

Table 1: Various video attributes contained within the Tracker Benchmark v1.0 dataset which make objects harder to track in the video. Each of the 50 test videos is annotated with the attributes which that video contains [6].

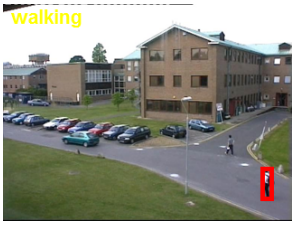
Video Attribute	Description
IV	*Illumination Variation*: the illumination in the target region is significantly changed.
SV	*Scale Variation*: the ratio of the bounding boxes of the first frame and,the current frame is out of the range [1/ts, ts], ts >1 (ts=2).
OCC	*Occlusion*: the target is partially or fully occluded.
DEF	*Deformation*: non-rigid object deformation.
MB	*Motion Blur*: the target region is blurred due to the motion of target or camera.
FM	*Fast Motion*: the motion of the ground truth is larger than tm pixels (tm=20).
IPR	*In-Plane Rotation*: the target rotates in the image plane.
OPR	*Out-of-Plane Rotation*: the target rotates out of the image plane.
OV	*Out-of-View*: some portion of the target leaves the view.
BC	*Background Clutters*: the background near the target has the similar color or texture as the target.
LR	*Low Resolution*: the number of pixels inside the ground-truth bounding box is less than tr (tr =400).



(a) Deer: MB, FM, IPR, BC, LR

(b) MountainBike: IPR, OPR, BC

(c) Tiger1: IV, OCC, DEF, MB, FM, IPR, OPR



(d) Walking: SV, OCC, DEF

Figure 2: Video frames from sample sequences in the Tracker Benchmark v1.0 dataset along with bounding boxes drawn around the target object [6]. The video attributes are also listed for each shown video.

rest of the tracking algorithm will then be focused on choosing the candidate sample patch, which we will also call a sample or a bounding box, that has the maximum posterior probability of containing the target. To generate candidate samples, we first let the bounding box containing the target in the previous frame be a candidate sample for the current frame. Note that our algorithm will always use the ground truth bounding box provided with the dataset for at least the first frame, beginning tracking with the next frame; this ensures there will always be a bounding box from a previous frame for our algorithm to draw on and it allows us to easily specify which object the algorithm should track in a video by placing the initial bounding box over that object. We then generate 120 additional samples by drawing from a normal distribution defined by $x_i \sim \mathcal{N}(x^*_{t-1}, \sqrt{wh}/2)$, where w and h denote the width and height of the target bounding box in the initial frame, x^*_{t-1} is the location of the target in the previous frame, and x_i is the ith sample generated at frame t, the current frame. This is the same procedure as used by [1], and it is based on the assumption that the most likely locations for the object are distributed around the location of the object in the previous frame. An example of some of these samples is shown in Figure 3.

4.2 Computing Sample Features with a CNN

For the next step of the algorithm, a CNN is used to transform each sample patch into a sample feature vector. The CNN used should be pre-trained on a dataset for generic object detection in images; this is because CNNs trained for such a task have been known to create transformed representations of images useful for object discrimination [7]. Each candidate sample is passed through the CNN, and the output of the CNN's first fully-connected layer is used as the feature vector representing that sample.

The original algorithm by Hong, et al. [1] used the Caffe

Figure 3: The location of the target from the previous frame (red box) and the normally sampled candidate patches (yellow boxes) for a biker from the video tracking dataset [6].

Model Zoo implementation of the pre-trained *R-CNN* created by Girshick, et al. [8] to create their sample feature vectors. However, [1] specified in their paper that CNN models other than Girshick's R-CNN may also be used for similar results. Thus, to circumvent the large amount of time needed to install Caffe, we decided to instead use MatConvNet [9], a CNN toolkit for Matlab with a simple installation process. MatConvNet provides several pre-trained CNNs, and we decided to use *VGG-F*, a CNN that has achieved state-of-the-art performance on the ImageNet object recognition database [10].

4.3 Using an SVM to Find Target-Specific Features from Sample Features

Online SVM code has been acquired from Cauwenberghs, et al. [11]. For this step of our procedure we utilize those sample features from the output of the CNN's first fully-connected layer and use them as feature inputs for our online SVM. Those samples which are positively classified (those representing the object of interest) will be used to generate target-specific features, all negative samples are discarded. Rather than use all of the features in each positive sample to generate a saliency map, we want to focus on those features which helped best discriminate our target against the background. We obtain the target-specific features as

$$\phi^+_k(x_i) = \begin{cases} w_k \phi_k(x_i), & \text{if } w_k > 0 \\ 0, & \text{otherwise} \end{cases}$$

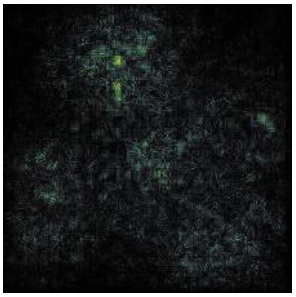
where $\phi_k(x_i)$ denotes a the k-th entry of our positive samples $\phi(x_i)$, and w_k denotes the k-th entry of our weight vector. In this way, we are able to stress those pixels which offer the best discriminatory information of our target and disregard the rest.

4.4 Computing Target-Specific Saliency Maps

Once the target-specific features are found for the positive samples, a class-specific saliency map needs to be generated for each of these samples, using the process detailed in [12]. These class-specific saliency maps will then be used to generate a target-specific saliency map, which will be used to find the likelihood of each pixel containing the target. To compute the class-specific saliency map from the target-specific features of a sample, we first have to insert the target-specific features into the first fully-connected layer of the pre-trained

CNN. After doing this, the features are back-propagated through the CNN back to the input layer of the network. The data retrieved from the input layer of the net is of the same size as the original samples, that is of size H x W x C, where H = the height of the target bounding box in the initial frame, W = the width of the target bounding box in the initial frame, and C = the number of color channels in the frames of the video. If C $\neq$ 1, then the resultant data is compressed to a single channel by taking the maximum absolute value across the C channels at every pixel. This produces a class-specific saliency map for each one of the positive samples. An example class-specific saliency map is shown in Figure 4.

(a) Positive Sample



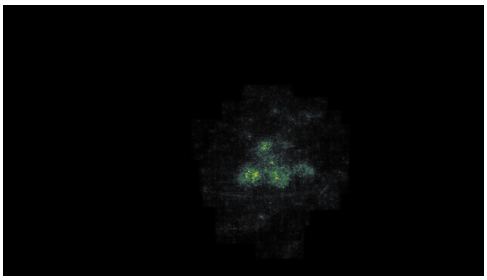
(b) Class-Specific Saliency Map

Figure 4: The class-specific saliency map produced from a positive candidate sample patch. Note that the regions in the saliency map that have the highest magnitude correspond to pixel locations that contain the target object.

Once a class-specific saliency map is generated for each positive sample, their data is aggregated into a single target-specific saliency map. This is done by first zero-padding each class-specific saliency map to the size of the original video frame, taking into account the location from which its corresponding candidate sample patch was cropped from the video frame. A target-specific saliency map is then generated by taking the maximum value across all the zero-padded class-specific saliency maps at each pixel location. This produces a single target-specific saliency map that has the same width and height as the original video frame. An example target-specific saliency map can be seen in Figure 5.

As noted in [1], back-propagating the features through the CNN is equivalent to taking the derivative of the feature vector in respect to the change in the image. This means that large values in the saliency maps (produced from back-propagation) indicate those pixel locations where changing the pixel value will most drastically affect the object classification of the CNN (which is determined by the feature vector). Intuitively, foreground pixels are those that change the CNN object classification the most because the object classification should ideally not be affected by the background pixels. Thus, the saliency maps capture the likelihood of a pixel being part of the foreground. Furthermore, because the target-specific features are only produced for the samples which the SVM determined contained the target, this means it is likely that these foreground pixels will be pixels containing the target object. Removing the features corresponding to non-positive weights further helps to ensure this. Thus the saliency maps can be used to provide a pixel-wise segmentation of the target object in the frame, as can be seen in Figure 4 and Figure 5.

(a) Original Video Frame



(b) Target-Specific Saliency Map

(c) Target-Specific Saliency Map Overlayed onto Video Frame

Figure 5: The target-specific saliency map produced from all the positive candidate sample patches. Note that the regions in the saliency map that have the highest magnitude correspond to pixel locations that contain the target object.

4.5 Localizing the Target with Target-Specific Saliency Map

Given the target-specific saliency map, we wish to locate the target in the video frame. Hong, et al accomplish this in their algorithm through a process they refer to as *sequential Bayesian filtering* [1]. Let M_t represent the target-specific saliency map at the current frame, frame t, and allow x_t to represent one of the candidate sample patches from this frame. Sequential Bayesian filtering then attempts to compute the *posterior probability* $p(x_t|M_{1:t})$ of each candidate x_t containing the target, given by

$$p(x_t|M_{1:t}) \propto p(M_t|x_t)p(x_t|M_{1:t-1}) \quad (1)$$

where $p(M_t|x_t)$ represents the likelihood of the sample and $p(x_t|M_{1:t-1})$ represents the prior of the sample. The candidate sample patch with the highest posterior is then selected

as the bounding box that contains the target in the current frame x_t^*. The algorithm will then continue by moving to the next frame, repeating the algorithm until all frames have been analyzed.

4.5.1 Computing the Likelihood with the Generative Model

In order to determine the likelihood of a candidate sample, a generative model H_t is first computed. This generative model is computed using the target-specific saliency maps of the m preceding frames where m is a constant, scalar value, set to $m = 30$ in [1]. Let x_k^* indicate the bounding box of the located target in frame k and $M_k(x_k^*)$ denote the target-specific saliency map at frame k cropped to the bounding box x_k^*. Hence, $M_k(x_k^*)$ shows how the target appeared in the saliency map at frame k. The generative model is then computed by simply finding the average appearance of the target in the last m frames:

$$H_t = \frac{1}{m} \sum_{k=t-m}^{t-1} M_k(x_k^*) \tag{2}$$

The likelihood of a candidate patch x_t is then found by convolving the generative model H_t with the current saliency map cropped to the candidate patch $M_t(x_t)$, as follows:

$$p(M_t|x_t) \propto H_t \otimes M_t(x_t) \tag{3}$$

where $\otimes$ represents the convolution operator. The likelihood of each candidate sample patch is then found in this way, using the same generative model for all patches.

4.5.2 Computing the Prior

To find the prior $p(x_t|M_{1:t-1})$ of a candidate patch x_t, both the location of this bounding box and the posterior distribution from the previous frame is considered. The prior is computed through the following formula:

$$p(x_t|M_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|M_{1:t-1})dx_{t-1} \tag{4}$$

This equation shows that a candidate's prior is the sum over all the candidates from the previous frame x_{t-1} of a location-based probability $p(x_t|x_{t-1})$ weighted by the previous frame's candidates' posterior probabilities $p(x_t|M_{1:t-1})$. The posterior probabilities were already calculated during the last iteration of the algorithm; the location-based probability, however, needs to be calculated. To do this, the mean μ_t and covariance Σ_t of the locations of the positive samples from the current frame are calculated. Then, the amount d_t that the target object has moved between frames $t-1$ and t is estimated using the following formula:

$$d_t = \mu_t - x_{t-1}^* \tag{5}$$

where x_{t-1}^* is the estimated location of the target object in the previous frame. It is then assumed that the the probability of the target object moving to a location is normally distributed around a mean of the distance vector d_t with covariance Σ_t. Thus the location-based probability of a candidate sample in relation to a candidate sample from the previous frame is found by:

$$p(x_t|x_{t-1}) = \mathcal{N}(x_t - x_{t-1}; d_t; \Sigma_t) \tag{6}$$

Plugging this into Equation 4 allows the prior for each candidate sample in the current frame to be computed.

4.5.3 Computing the Target Posterior

After computing the likelihoods and priors, the target posterior $p(x_t|M_{1:t})$ for a candidate sample patch can then be computed by using Equation 1. The location of the target in the current frame x_t^* is then set as the candidate patch with the highest posterior according to the following equation:

$$x_t^* = \underset{x_t}{\operatorname{argmax}}\, p(x_t|M_{1:t}) \tag{7}$$

Having found the target in the current frame, the algorithm then has the option to update the training of the SVM, as detailed Section 4.6. After this, the algorithm begins its next iteration by searching for the object in the next frame, $t+1$, starting by generating new candidate sample patches around x_t^*.

Note that in practice, we found that if the target object moved too much between a pair of frames, the prior probability, which gives preference to bounding boxes that have not moved very much, would prevent the algorithm from selecting candidate sample containing the target object. Therefore, we found that the algorithm produces more accurate and smooth results when the prior is not used to calculate the posterior, using a modified posterior $p(x_t|M_{1:t})'$ such that

$$p(x_t|M_{1:t})' \propto p(M_t|x_t) \tag{8}$$

Because of this, the majority of our reported results do not use the prior to calculate the posterior. This is discussed further when discussing the results below.

4.6 Training the SVM

In our implementation we update the online SVM every third frame past the initial frames. Our intuition behind this lay in the fact that the representation our target changing dramatically in two frames is highly unlikely give the FPS rate of most modern recording devices. This also helps reduce the computational overhead associated with training new samples at every frame, improving the overall speed of our program. As an additional step in helping reduce the computational demands, we have a strict limit in the number of samples in our training set. While time did not permit for much parameter tuning, a base of 300 samples. Each frame used for training is limited to 25 overall samples (1 positive, 24 negative), the labels of which are calculated as follows

$$y_i = \begin{cases} +1, & \text{if } x_i^{'} = x^*t \\ -1, & \text{if } \frac{\text{BB}(x_t^*) \cap \text{BB}(x_i{}')}{\text{BB}(x_t^*) \cup \text{BB}(x_t{}')} < \delta \end{cases} \tag{9}$$

where BB(x) denotes the bounding box of a given state x, and δ represents an overlap threshold (in our case 0.3). Concretely, the negative examples we use in our training set are obtained by choosing those samples which have less than a 30% overlap with the sample that offered the best posterior probability. We implement a FIFO-like replacement policy on our SVM training set, such that when our set threatens to become larger than the 300 samples we simply replace with oldest set of 25 samples with the most recent. Doing this allows our SVM to train on samples that discriminate between the target and very recent background information - this is helpful in sequences that have rapidly changing backgrounds. After updating the SVM we need to recalculate

the weights. We can do this by calculating the following

$$w = \sum_{i \in E} a_i y_i \phi(x_i) \tag{10}$$

Training the SVM returns a_i which are Lagrange multipliers, and E which are the support vectors lying on or inside the margin. This updated weight vector is used for the target-specific feature generation discussed in section 4.3.

5. RESULTS

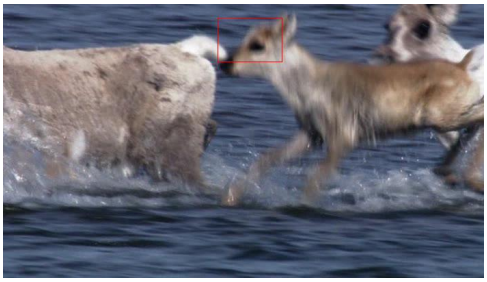
We ran our algorithm on many different videos from the Tracker Benchmark v1.0 dataset [6] and analyzed the resulting videos qualitatively. Some of these results can be found on our GitHub page here: `https://github.com/msu-ml/16spr_cao_gonzales_hoffman/tree/master/results`. In the following sections, we present and analyze some of these results.

5.1 How the Prior and the SVM Affect the Tracking

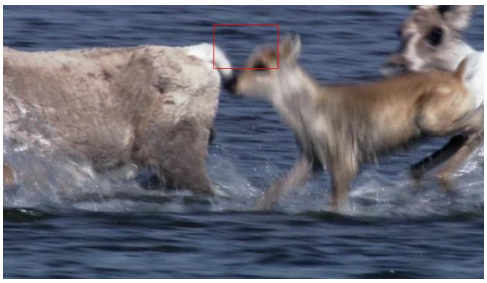
Early on in testing our algorithm, we decided to analyze how the SVM affects the tracking algorithm and also how using the prior probability in determining the posteriors, i.e. using Equation 1, affects the algorithm in comparison to not using the prior, i.e. using Equation 8. This section presents and discusses some of these results. The video sequences referred to in this section can be viewed at the following location on our GitHub page: `https://github.com/msu-ml/16spr_cao_gonzales_hoffman/tree/master/results/Affects-of-Prior-and-SVM-Results`.

The first test we decided to perform was to compare using the prior and not using the prior. As the SVM had not been fully integrated into our algorithm at this point these tests contained no help from an online SVM. The reason we decided to test the affect of the prior is because we noticed that for some video sequences, the target object would move a relatively large distance in between frames, i.e. the object was moving fast, and our tracking algorithm was not following the object when this happened. For example, a few frames from the *Deer* video sequence where this fast object movement occurs are shown in Figure 6.

(a) frame = 2



(b) frame = 3



(c) frame = 4

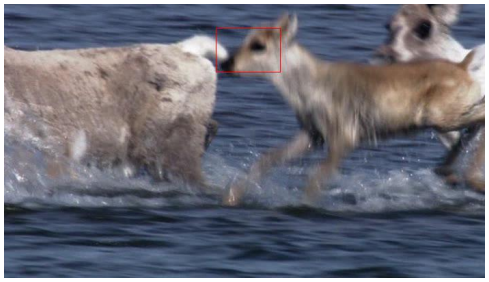
Figure 6: A video tracking sequence showing fast object movement, which caused the algorithm to fail. The prior was used and an SVM was not used in this sequence.

We hypothesized that this failure was due to the use of the prior in calculating the posterior probability in Equation 1. The prior probability algorithm contains a component that functions as a location-based probability (see Equation 6); this means that the prior probability for a sample will be higher for candidate samples that are close to the target object in the previous frame. Thus, when the target object moves a large distance between frames, the candidate sample that corresponds to the moved target will have a *low* prior since it is far from where the target was in the previous frame. Our hypothesis then was that this caused the candidate samples to occasionally have such a low posterior probability that other candidate samples, not containing the target, actually had a larger posterior probability and were selected by our algorithm as containing the target.

To test this hypothesis, we ran the same *Deer* sequence using the modified posterior probability (Equation 8) that does not take into account the prior probabilities. A few resulting frames are shown in Figure 7. As can be seen from these results, the tracking algorithm performed significantly better overall when the prior was not used in comparison to when the prior was used. We also performed this test on a few other sequences, including the *MountainBike* sequence, and found the same pattern to be true. This empirically confirmed our hypothesis that the prior probability was causing errors for fast-moving objects.

Having tested how the prior affects the tracking *without* using an SVM, we decided that our next test would be to see how including the SVM would affect the tracking. We noticed that without the SVM, the tracking would occasionally loose the target, either to track another object or thinking that the background was the target object. This is seen in Figure 8. Including the SVM would allow the algorithm to remember what the target object looks like and, we hypothesized, increase the accuracy. After performing several tests including the SVM, we noted that overall, the SVM did significantly improve the tracking accuracy, as expected. An example of this is seen in Figure 9, where the SVM helped the algorithm to continue tracking the correct deer through a few problem frames. We also performed additional tests that showed that using the SVM *with* the prior was able to perform better in some instances than when neither the

(a) frame = 2

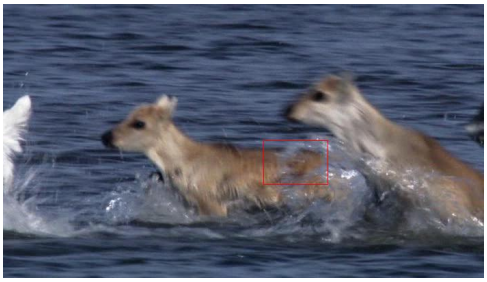


(b) frame = 3

(c) frame = 4

Figure 7: A video tracking sequence showing fast object movement, which the algorithm was able to account for. The prior was not used and an SVM was not used in this sequence. Notice the performance increase compared to when a prior was used in Figure 6.

(a) frame = 43



(b) frame = 44

(c) frame = 45

Figure 8: A video tracking sequence showing the tracking algorithm beginning to track a different object. The prior was not used and an SVM was not used in this sequence.

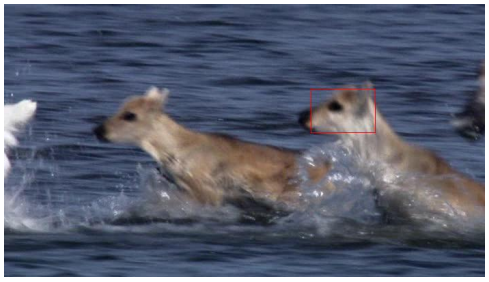
SVM nor the prior were used, and these results are shown on our GitHub page.

After all of these tests, we determined that our tracking algorithm performed best when we used the SVM but did not use the prior. Therefore, the rest of the results shown here use this configuration.

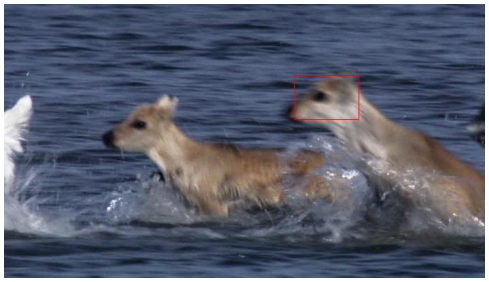
5.2 Modifying the SVM Training to Improve Tracking

Since we implemented a FIFO replacement policy on our SVM training set, we found that some problems occurred. For example, a sample which offered the max posterior probability for a given frame was not guaranteed to contain our object. In the cases where this happened, our SVM would begin adding those samples as positive samples in training. Additionally, it was entirely possible that some of the 'negative' examples which had less than a 30% overlap with this false positive actually contained more of our target than the positive sample! In these cases our SVM began learning that the original background was our tracking object, while our original tracking object became considered background material. Since our oldest samples of the target object had long since been replaced in our training set, there were instances when our SVM training set was full of positive samples that contained none of our original target. To solve this we decided it would be prudent to permanently keep all the positive samples from the initial training frames, as well as the same number of positive samples in the next frames. For example, if we decided to initialize our SVM on ten frames which were guaranteed to contain our target, we would also assume that our algorithm would provide accurate bounding boxes in the subsequent ten frames. These 20 positive samples (one from each frame) would be permanently kept in our training set, as they were samples with a very high confidence of containing our target. After implementing this change we found a noticeable improvement in tracking. Sequences in which the bounding box would lose the initial target before were now much more likely to stay consistently on target throughout.

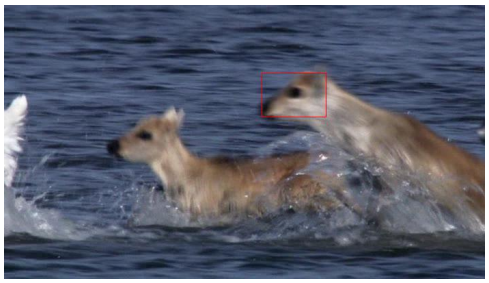
5.3 How Object Scale Affects the Tracking



(a) frame = 43



(b) frame = 44



(c) frame = 45

Figure 9: A video tracking sequence showing the tracking algorithm correctly tracking the target object. The prior was not used and an SVM was used in this sequence. Notice the performance increase compared to when an SVM was not used in Figure 8.

When testing our algorithm, we noticed that the algorithm has trouble tracking an object when the size, or scale, of the object changes between frames. This often happens when the target object comes closer to or goes farther away from the camera. Some examples of our algorithm failing from what we believe are scaling issues are shown in Figure 10 and Figure 11. The complete video sequences can be found on our GitHub page at `https://github.com/msu-ml/16spr_cao_gonzales_hoffman/tree/master/results/Scale-Variation-Results`. Unfortunately, the scaling issue is a problem inherent in the algorithm design by Hong, et al. This is because the algorithm assumes that the target object's bounding box stays constant for all the frames. While the algorithm could be modified to check for growing or shrinking bounding boxes, time constraints prevented us from adding this to our code. This is one task we wish to work on in the future. As it turns out, many of the tracking errors we have had so far can be attributed to variations in scale.

5.4 Miscellaneous Results

Other than those tests already mentioned, we tested our algorithm on several other videos from the testing dataset. Some of these results are shown in the figures below and can also be seen on our GitHub page at `https://github.com/msu-ml/16spr_cao_gonzales_hoffman/tree/master/results`. Note that several figures appear *after* the references section of the report. Overall, we feel that our algorithm was able to perform well on video sequences that did not involve too much scale variation. We believe that many of the primary problems the algorithm currently faces can be fixed through the future work we plan to implement.

6. CONCLUSIONS AND FUTURE WORK

We were able to re-implement the results of Hong, et al. [1] in order to produce an object tracking algorithm for videos that worked surprisingly well overall. This algorithm contributed to the field of object tracking in several ways. The use of a CNN to extract features on the target object allowed us to overcome many of the traditional problems in object tracking, such as some lighting variations and object rotation. The use of an online SVM on top of the CNN then allowed the algorithm to learn the representation of a specific object at runtime, improving tracking performance. We did run into some problems, such as the prior causing the tracker to lose fast moving objects and scale variations causing the tracker to lose objects moving too much towards or away from the camera. However, we were able to circumvent some of these problems (e.g. removing the prior) and came up with plans for future work to solve some of the other problems (e.g. dealing with scale variations). While our algorithm did not perform as well qualitatively as the results provided by Hong, et al., we recognize that the authors of that paper may have not included in their paper all of the optimizations they used for their algorithm, and they likely had a larger amount of time to fine tune the algorithm to the Tracking Benchmark v1.0 dataset. They were also able to use the pre-trained *R-CNN* in their algorithm, which is available in Caffe but not MatConvNet, and this may have caused significant differences in our results.

We learned many important aspects of object tracking while working on this project. First off, we learned how to use a pre-trained CNN to create target-specific saliency maps in order to localize a target. We found this a very interesting, ingenious way to use a CNN in object tracking. This project also gave us experience implementing the results of someone else's paper and learning to interpret/discover the many parts of an algorithm that are not made clear in a paper. During this project, we also had a hands-on experience dealing with many of the challenges involved in video tracking, such as scale variation, occlusions, deformations, fast motion, and object rotation. If not accounted for, these challenges can absolutely devastate a tracking algorithm in only a few frames. While this algorithm was able to handle many of these challenges, it could not handle all of them which led us to think deeper into how these challenges can be overcome.

In the future, we are hoping to continue modifying this algorithm and improving it to work in more situations. One of the most obvious problems inherent in the algorithm that needs to be addressed is that of scale variation. This al-

gorithm currently assumes that the bounding box around an object can stay the same size throughout all the video frames. However, this assumption proves to be false in far too many circumstances. Therefore, we plan to modify our algorithm so that when we generate candidate sample patches, we will generate some that are smaller and some that are larger than the current bounding box size. This would allow the bounding box to grow or shrink as needed. We also noticed that if the algorithm does a poor job at classifying the target for several frames in a row, our current SVM training algorithm will begin training on these mistakes, leading to more mistakes in the future. Occasionally, this causes no positive samples to be found in a frame. Currently, in this situation, we just set the bounding box to be the same as that in the previous frame and then continue to the next frame. In the future, we plan to modify the SVM in a few ways to help account for this. First, if no positive samples are found, we plan to then back-propagate most or all of the samples through the CNN, with the hope that the target object, although not found by the SVM, will be found in some of these samples and will be found by the target posterior. Then we could retrain the SVM including this new target location. Also, we plan to decrease the amount of allowed overlap between the positive sample fed to the SVM and the negative samples fed to the SVM so that there is less likelihood it will be told that a sample including large parts of the target is a negative sample during training. Thirdly, we also wish to modify the SVM so that it is a truly incremental, online SVM; currently, we have to retrain the SVM on *all* the training data every time we wish to train it. Because of this, we had to significantly decrease the amount of training samples we feed the SVM to decrease the amount of time it takes to train. If we could implement an incremental, online SVM, then each time we train it we would only feed it the *new* training samples, i.e. those from the previous frame. This would not only allow it to train on a much larger set of training samples, but also vastly reduce the computational demands per frame which would allow our algorithm to work far faster. This was an issue for us, as we were unable to test our algorithm against some of the larger videos due to time constraints. If properly implemented, we could test our approach against larger video sequences and see how well our algorithm handles changes in the target over time. Other than these updates, we have some other future work planned as well. For instance, we may turn our main program file into a function to make it easier to call, may add an option to track multiple objects, and may investigate tracking objects in spectra other than the visible light spectrum, such as in thermal videos.

7. WORK DISTRIBUTION AMONG AUTHORS

This section outlines how the work was distributed between the two authors of this report.

7.1 Aaron Gonzales

Aaron Gonzales was primarily responsible for generating the candidate sample patches, for using an SVM to find the target-specific features from the sample features, training the online SVM, and generating the demo videos from the final results. He also researched the related work and was the primary contributor towards creating the final project presentation.

7.2 Steven Hoffman

Steven Hoffman was primarily responsible for writing the skeleton code for the overall tracking algorithm, passing samples through the CNN to get the sample features, back-propagating the target-specific features through the CNN to create a target-specific saliency map, computing the generative model, and finding the target posterior. He also was the primary contributor towards creating the final project report.

7.3 Both Authors

Both authors worked together to write up the various project reports, to give the final presentation, and to run the code on various testing sequences (i.e. videos).

8. REFERENCES

[1] Seunghoon Hong, Tackgeun You, Suha Kwak, and Bohyung Han. Online tracking by learning discriminative saliency map with convolutional neural network. *arXiv preprint arXiv:1502.06796*, 2015.

[2] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[3] Jialue Fan, Wei Xu, Ying Wu, and Yihong Gong. Human tracking using convolutional neural networks. *Neural Networks, IEEE Transactions on*, 21(10):1610–1623, 2010.

[4] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 809–817. Curran Associates, Inc., 2013.

[5] H. Li, Y. Li, and F. Porikli. Deeptrack: Learning discriminative feature representations online for robust visual tracking. *Image Processing, IEEE Transactions on*, PP(99):1–1, 2015.

[6] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2411–2418, 2013.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 38(1):142–158, 2016.

[9] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 689–692. ACM, 2015.

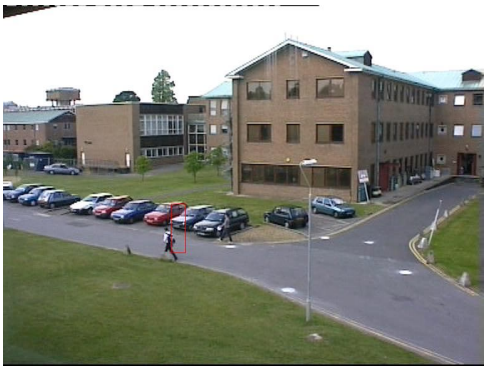
[10] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[11] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, 13:409, 2001.

[12] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

(a) frame = 1

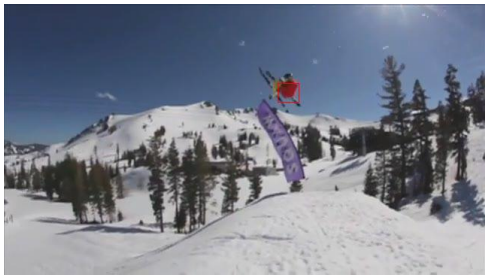
(b) frame = 207



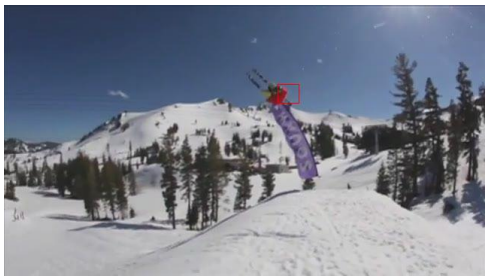
(c) frame = 217

Figure 10: A video tracking sequence showing the tracking algorithm failing to account for changes in scale. Notice that by frame 217, the target object had shrunken so much that the tracking algorithm no longer recognized it.

(a) frame = 1



(b) frame = 31



(c) frame = 32

Figure 11: A video tracking sequence showing the tracking algorithm failing to account for changes in scale. Notice that by frame 32, the target object had grown so much that the tracking algorithm no longer recognized it.