Adaptive Depth Computational Policies for Efficient Visual Tracking

Chris Ying, Katerina Fragkiadaki
Machine Learning Department
Carnegie Mellon University University
Pittsburgh, PA 15213
{cying,katef}@cs.cmu.edu

Abstract

Though convolutional neural networks have made significant improvements to the task of video tracking, they come at the cost of being extremely computationally expensive. In this work, we make the observation that different frames in a video can require different levels of network complexity in order to track with high accuracy. To exploit this, we propose a fully convolutional Siamese network that performs video tracking at multiple network depths. We learn an adaptive policy, in the form of gating functions, to control how deep to evaluate the network during runtime. Training proceeds in two phases, a finetuning phase where we train the convolutional weights to extract meaningful features for metric learning and a gating phase where we train the gate weights to balance accuracy and computational cost. Our results show that our network can achieve accuracy that is competitive with the state-of-the-art on the VOT2016 benchmark. Furthermore, we show that using conditional computation with the adaptive policy, we achieve higher accuracy than fixed-depth policies with less computational cost. The framework we present extends to other tasks that use convolutional neural networks and enables trading speed for accuracy at runtime.

1 Introduction

Convolutional neural networks have made significant progress on a wide range of computer vision tasks, including image classification [11], semantic segmentation [12], and optical flow [5]. This success has come from both improvements to network architecture, such as fully convolutional networks, and to the increasing usage of GPUs to speed up training and evaluation. However, even with high-end GPUs, it is extremely computationally expensive and time consuming to use convolutional neural networks. This makes many deep learning techniques unfeasible to use on less power devices (e.g. mobile phones) and for online tasks that require low latency output (e.g. self-driving cars).

Consider the task of video tracking, which is to take a labeled object in one frame of a video (hereinafter referred to as the key frame) and to label the location of the object in the subsequent frames (hereinafter referred to as the search frames). Many current approaches to this task use region proposal and classification [19] or finetune an object detector on the key frame via back-propagation before running on the search frames [18]. These approaches require specialized training for different objects and suffer from slow test-time performance (around 3 fps for both approaches).

In this work, we propose a fully convolutional neural network based on Siamese networks [8] that performs visual tracking via cross-correlation. We make the observation that in many search frames of a video, the tracked object is very similar to the object in the key frame (as in figure 1) and it would be computationally wasteful to use the entire network to make a simple prediction. To address

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

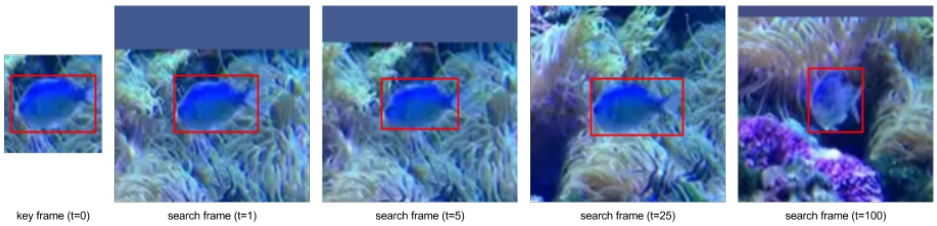

Figure 1: Preprocessed key and search frames from `fish1` video in VOT2016. The red box is the ground truth bounding box and is added for visualization purposes. Each training search frame is centered on the ground truth box of the previous frame. Search frames that extend beyond the original video border and are padded with the mean RGB.

this, we learn an adaptive policy that uses conditional computation in the form of gating functions to dynamically determine how deep we should to evaluate the network before outputting the tracking prediction. Combining these ideas, we demonstrate that we can perform video tracking with close to state-of-the-art accuracy at real-time speeds on the challenging VOT2016 dataset [10].

2 Background

2.1 Fully convolutional networks

Classical convolutional neural networks such as AlexNet [11] and VGG [15] use many convolutional layers followed by a small number of fully connected layers to perform image classification. Fully convolutional networks are networks with no fully connected layers. A consequence is that fully convolutional networks are entirely translation invariant and can handle different resolution inputs without modifying the architecture or weights. The latter is particularly useful for this paper because the key and search frames can be different sizes. In addition, the output of a fully convolutional network is a feature map, which allows pixel-wise prediction. Recent work has shown that fully convolutional networks can achieve state-of-the-art results on semantic segmentation [12] and visual tracking, [18].

The deep feature map at the end of a fully convolutional network is essentially an embedding of the image onto a high dimensional space of image features of increasing complexity (e.g. edges, shapes, patterns, etc...). A key observation we make in this paper is that the feature maps at intermediate layers of the fully convolutional network are also embeddings of the image, albeit with shallower features than the deepest layer. These shallower features can be used for metric learning, which uses the distance between features rather than the features themselves. GoogLeNet [17] uses the idea of intermediate layers to introduce auxiliary classifiers in order to address the vanishing gradients problem.

2.2 Metric learning for video tracking

Metric learning refers to learning a distance function over a space of objects. For image similarity, one approach to learning this distance function is through a Siamese neural network [8] which uses the same neural network to extract features from a pair of images before using a single fully connected layer to predict the distance. [6] goes further and introduces a triplet network to simultaneously minimize the distance between instances of the same object while maximizing the distance between different objects.

In the context of video tracking, metric learning is useful for calculating the distance between the tracked object in the key frame at each spatial location in the search frame. [4] uses a convolutional neural network as the feature extractor and uses 2D cross-correlation to implement the distance layer between deep feature maps, as in figure 2. This implicitly implements a triplet network and efficiently calculates the distance function over all spatial locations simultaneously. Since cross-correlation requires no additional learned parameters, any deep feature extractor can be used, including a pretrained model for some other computer vision task like image classification. The resulting network

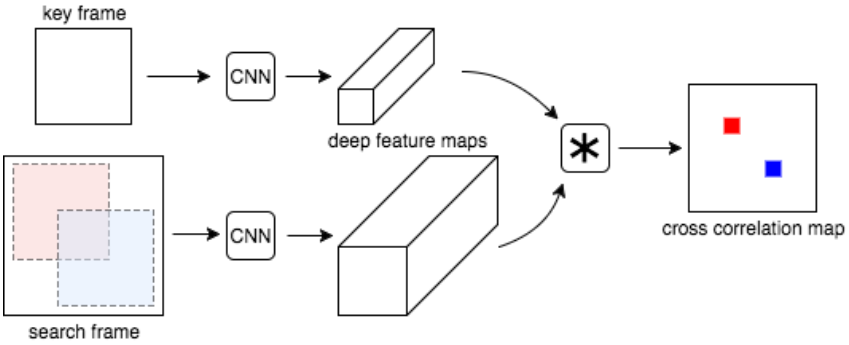

Figure 2: Siamese network with 2D cross-correlation for key-search frame pairs. The deep feature maps for the key and search frames are extracted by the same convolutional neural network. $*$ denotes 2D cross-correlation. The scores for the light red and light blue squares in the search frame are represented by single values at spatial locations in the output cross-correlation map.

requires no extra training during test time and can track with a single frame as input (also known as one-shot learning).

2.3 Conditional computation via reinforcement learning

Computational cost is an important factor in designing convolutional architectures. Conditional computation refers to activating different network components depending on the input and serves as a promising way to reduce computational cost without sacrificing representational power. [2] implements conditional computation by selectively activating different weights in each layer and frames the training as a reinforcement learning problem. Rather than using different weights in the same network layers, [14] uses a sparse gating function to determine which sub-networks (each of which are "experts" for different input) to execute and shows that it is possible to train the gating and network weights jointly via back-propagation.

In this work, we observe that deciding the depth of the network to evaluate can be formulated as a reinforcement learning task:

- The state is the current depth of evaluation in the network and the observations are the feature maps at the depth.
- The action is whether to continue computation to the next depth or to stop and output the current prediction. This is controlled by a gating function.
- The reward is accuracy of the of the bounding box prediction made at the determined depth. A penalty is added to the reward which increases based on the depth of evaluation.

Theoretically, we could learn the weights to the gating function via stochastic neurons [3] and policy gradients [16]. However, in this work we introduce a surrogate loss and optimize via back-propagation, much like [14], which gives more balanced updates to all the gates and more meaningful gradients.

3 Methodology

3.1 Data

Experiments in this paper use the Visual Object Tracking dataset VOT2016 [10]. The dataset consists of 60 videos extracted from various other computer vision datasets. There are a total of 21455 frames of various resolutions. Each frame is labelled with the (x, y) coordinates of a bounding box that surrounds the a single object being tracked in the video. The dataset is designed to be difficult for tracking so the videos have noisy backgrounds, the object can change shape or orientation, and there is occlusion in some frames.

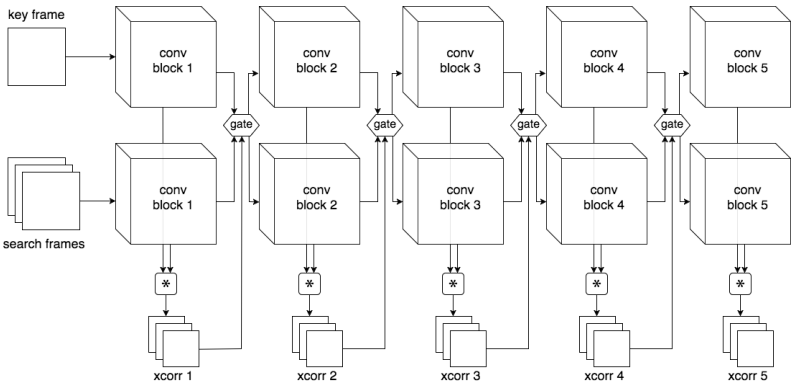

Figure 3: Base architecture for tracker. Each conv block includes 2-4 convolutions with ReLU activation. All blocks except 5 end with a max-pool layer. The feature maps of the key and search frames between each conv block and at the end are cross-correlated to yield 5 sets of cross-correlation (xcorr) maps. A gating function is added between each convolutional block which controls whether the network stops or continues computation during test time.

Since the VOT2016 dataset does not include a train-validation split, we choose approximately 25% of the data to hold out as the test set.

3.2 Preprocessing

Before using the dataset for training and evaluation, we first preprocess the videos by selecting a key frame every 10 frames and the subsequent up-to-100 frames as the search frames. We resize and crop the key frames to 128×128 centered at the tracked object such that there is at least 25% padding around the bounding box. Each of the search frames are resized with the same scale and cropped to 256×256 such that the frame is centered at the object at the previous frame. If the cropped search frame extends beyond the edge of the image, we pad the extra pixels with the mean RGB value. See figure 1 for an example of preprocessing on some frames from the `fish1` video category.

3.3 Network architecture

The base architecture we use for the feature extractor is the 19-layer VGG architecture [15]. We remove the fully connected layers of the architecture and treat the remaining convolutional and max-pool layers as the feature extractor. The VGG architecture is divided into 5 blocks of convolutions with 2-4 convolutional layers each, each ending in a max-pool layer. We remove the last max-pool layer in order to keep the deep feature maps as large as possible (i.e. 16×16 in the last layer).

After each convolutional block, we have a cross-correlation layer which performs 2D cross-correlation between the key and search frame feature maps. To improve training, we normalize the key and search feature maps via batch normalization and rescale the output cross-correlation map to $[0, 1]$.

This yields a total of 5 cross-correlation maps. To keep the computational cost low and to ensure that each cross-correlation map is the same resolution, we first downsample the feature maps via max-pool to the same resolution as the deepest feature map before performing cross-correlation.

We implement conditional computation via gating functions between convolutional blocks. From each cross-correlation map, we extract a feature vector that captures the "quality" of the map (intuitively, a cross-correlation map is better when it is "peakier", i.e. has a single definitive peak). Since this feature feature vector will be used for gating, it must be lightweight, which rules out a convolutional neural network. Instead we use hand-designed features which measure various statistics of the cross-correlation map including entropy, kurtosis, depth, max peak values, peak offset, etc... (we do not use bounding box information because that is not available during testing). On top of this feature vector v_i at each of the 5 depths, we learn a linear predictor (ϕ_i) with sigmoid activation (σ) and a single output. The confidence value c_i for the i-th cross-correlation map is calculated in equation 1

which introduces a "total budget" of 1.0 for all the confidence values. This scheme gives the desired behavior that a higher confidence in a shallower depth corresponds to less need for deeper layers.

$$c_i = \begin{cases} (1.0 - \sum_{n=1}^{i-1} c_n)\sigma(\varphi_i(v_i)) & i \in [1, 4] \\ 1.0 - \sum_{n=1}^{4} c_n & i = 5 \end{cases} \tag{1}$$

For soft-gating, we take the weighted average of the cross-correlation maps with confidence as weight for the final prediction. Soft-gating does not save any computation and is mainly for comparison. For hard-gating, we take the shallowest cross-correlation map with confidence greater than some tune-able threshold.

3.4 Training

Training is performed in batches of 1 key frame paired with 25 search frames, where the goal is to find the objected labelled in the key frame in each of the 25 search frames. This efficiently implements 25 pairwise search tasks simultaneously, each with the same key frame. The ground truth cross-correlation map, G, is generated via a 2D Gaussian distribution centered at the true offset (t_x, t_y) of the tracked object, as in equations 2 and 3.

$$G'(i, j) = A \exp\left(-\left(\frac{(i - t_x)^2}{2\sigma^2} + \frac{(j - t_y)^2}{2\sigma^2}\right)\right) \tag{2}$$

$$G(i, j) = \frac{G'(i, j)}{\sum_{i,j} G'(i, j)} \tag{3}$$

The loss per feature map is calculated via spatial softmax and cross-entropy as in equations 4 and 5. $C_i(x, y)$ refers to the (x, y) coordinate of the i-th cross-correlation map.

$$C_i'(x, y) = \frac{\exp(C_i(x, y))}{\sum_{x,y} \exp(C_i(x, y))} \tag{4}$$

$$L_i(C_i', G) = -\frac{1}{|C|} \sum_{x,y} C_i'(x, y) \log(G(x, y)) \tag{5}$$

Since we cannot back-propagate through the gating functions (the features are non-differentiable), we perform training in 2 phases: the finetuning phase which involves tuning the weights in the convolutional layers, and the gating phase, which involves tuning the linear predictors in the gating functions.

During the finetuning phase, the loss L_{finetune} is the sum of the losses at the 5 depths of cross-correlation layers. This encourages the network to extract features that are both good for image similarity and good for the next convolutional block for the intermediate convolutional layers.

$$L_{\text{finetune}} = \sum_{i=1}^{5} L_i \tag{6}$$

During the gating phase, instead of using the hard-gating prediction, we train using a weighted sum of the losses L_{pred}, weighted by the confidence outputs c_i. Furthermore, to encourage the network to select shallower cross-correlation maps to save computation, we introduce a computational loss term L_{comp} which penalizes selecting deeper cross-correlation maps. The final loss that is use for training L_{gating} is a sum of the two.

$$L_{\text{gating}} = \underbrace{\sum_{i=1}^{5} c_i L_i}_{L_{\text{pred}}} + \lambda \underbrace{\sum_{i=1}^{5} p^{i-1} c_i}_{L_{\text{comp}}} \tag{7}$$

p_i are hyperparameters that control the increase in penalty for going deeper in the network and λ is a hyperparameter that controls the weight of the computation loss. Optimization is performed using Adam [7] for both phases.

3.5 Implementation

The network was implemented in Tensorflow v1.0.0 [1] using pretrained VGG network weights on trained on ImageNet [13] for image classification. All training and evaluation was performed on a single NVIDIA TITAN X GPU, and Intel Xeon E5-2630 v3 CPU, and 16 GB of RAM. The code can be found on Github [1].

Since training is performed via batches of 1 key frame and B search frames, the dimensions of the key and search feature maps are $1 \times W_k \times H_k \times C$ and $B \times W_s \times H_s \times C$, respectively. Cross-correlation is efficiently implemented on GPU by performing 2D convolution on the search feature maps with the key feature maps as the filter, treating the C dimension as the input channel size (the output channel size is 1).

To efficiently implement hard-gating, we use Tensorflow's control flow operators (`tf.cond`). Hard-gating is only fully efficient when the batch size of the search frames is 1 since the computation is bottlenecked by the deepest cross-correlation map that is required by a sample in a batch. In practice, this is not as much of an issue since consecutive frames tend to use the similar depth maps for prediction.

4 Analysis

4.1 Metrics

The goal of the analysis is to show that our system can approach state-of-the-art metrics in video tracking and also save computational power without sacrificing "too much" tracking accuracy. The standard metric for measuring tracking accuracy is intersection-over-union (IOU) (equation 8) between the predicted object box P and the ground truth box G. The predicted object box is found using the position of the maximum value in the cross-correlation map as the offset and the bounding box dimensions are the same as the key frame reference box.

$$\text{IOU}(P, G) = \frac{|P \cap G|}{|P \cup G|} \tag{8}$$

To measure how well the system works at different frame gaps, we measure the IOU at up-to 1, 5, and 25 frames ahead of the key frame (e.g. for IOU@25, we take the average IOU of the tracker with key frame t and search frames $t+1, t+2, ..., t+25$).

To measure computational cost, we compute the theoretical floating point operations (FLOPs). Experimentally, we find that FLOPs is a good proxy for true computational cost as measured in frames-per-second (FPS). The reason FLOPs is the preferable metric is that FPS is heavily tied to hardware and software constraints, which may prevent the architecture from achieving the theoretical speedup.

4.2 Experimental results

4.2.1 Finetuning performance over time

We want to demonstrate that the network can successfully perform metric learning for video tracking. We measured the IOU metrics of the network during the finetuning phase of training in figure 4. While the train IOU continues to improve with diminishing returns as training continues, the testing IOU levels off at around 8 epochs and decreases afterwards. The results are more stable as for longer frame windows and the IOU drops as the frame gap increases as expected since the object may deform or change size over time. Since the starting weights are pretrained, even without finetuning, the network can perform tracking with over 0.40 IOU.

[1] `https://github.com/chrisying/vgg-tracker`

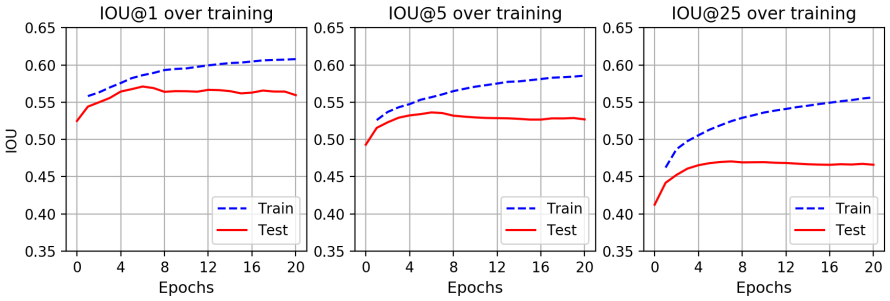

Figure 4: Training and testing curves over epochs trained during finetuning. From left to right, plots show the IOU of the system averaged over 1, 5, and 25 frames ahead of the key frame.

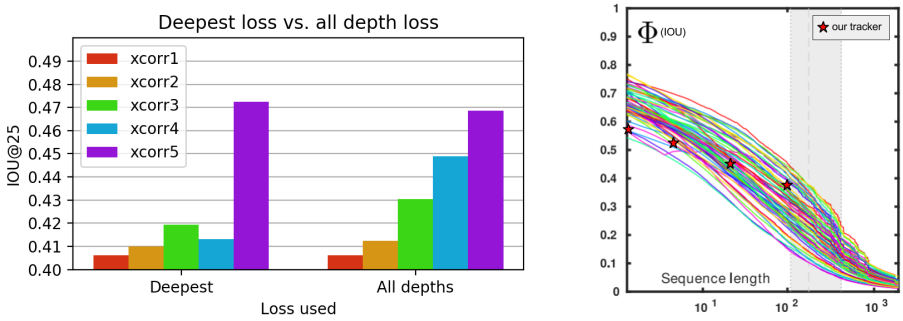

Figure 5: (left) Comparing accuracy when training with all depths loss vs. only deepest loss. (right) Comparison of our tracker against top submissions to VOT2016. Each curve is a different submitted tracker. The stars represent our tracker's accuracy at select sequence lengths.

4.2.2 Finetuning deepest vs. all layers

In the previous experiment, we used the loss at all depths during optimization, which ensures that all cross-correlation layers can be used for metric learning. If we instead use only the loss at the deepest layer, we can improve the accuracy at the deepest cross-correlation map (xcorr5) at the cost of accuracy at other depths (xcorr1 - xcorr4). As demonstrated in figure 5 (left), training with all the losses yields increasing accuracy as depth increases while training with only the deepest yields a big jump in accuracy at depth 5 but lower accuracy at shallower depths. If computational cost is not a factor, using only the deepest loss gives around 0.01 IOU benefit over using all depths. Since our goal is to potentially use shallower layers for evaluation, for further experiments we choose to use the loss at all depths.

4.2.3 Comparison to state-of-the-art

We compare our accuracy results to some of the top entries in the VOT2016 competition [9] in figure 5 (right). We did not replicate the testing environment exactly because of preprocessing that we perform beforehand, so this comparison only provides a rough estimate of the performance of our tracker versus the state-of-the-art. Our tracker is not as accurate as the best trackers at small sequence lengths, but is competitive with many trackers at around 100 frames. Note also that the top 4 trackers from VOT2016 run at under 1 FPS while our system runs at over 37 FPS using the deepest layer (xcorr5).

4.2.4 Comparison of different gating policies

The different gating policies that are compared are: fixed depth (always use the cross-correlation map at each of the 5 depths), soft-gating, and hard-gating. Table 1 shows the theoretical FLOPs required to compute the cross-correlation maps for a single key-search batch. For simplicity of calculation, the values only include the floating point multiplication operations, which comprise the bulk of the computation.

Table 1: Theoretical FLOPs for each gating policy.

Gating policy	FLOPs ($\times 10^9$)	Relative to xcorr1
xcorr1	2.78	1.00×
xcorr2	67.70	2.43×
xcorr3	160.75	5.78×
xcorr4	253.79	9.12×
xcorr5	280.37	10.07×
soft-gating	280.53	10.08×
hard-gating	varies	varies

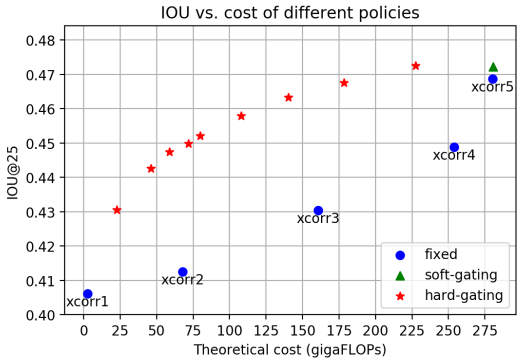

Figure 6: Comparing IOU and computational cost for different policies. Top left is more desirable. For hard-gating, multiple hyperparameter settings are reported.

Figure 6 shows the comparison of the accuracy and the computational cost of the different gating policies. Since hard-gating is hyperparameter sensitive, we report the accuracy and cost values at multiple settings. Both soft and hard-gating can achieve accuracy values that exceed any fixed depth policy and furthermore hard-gating uses significantly less computational cost to achieve the same or better accuracy.

5 Discussion

Our experimental results show that our proposed fully convolutional metric learning network can perform tracking with more than 0.55 IOU at one frame ahead and over 0.45 IOU averaged over 25 frames. Despite limitations discussed in 6, our tracker remains competitive with state-of-the-art trackers submitted to the VOT2016 competition (figure 5 right). Our experiments with different adaptive gating policies show that it is possible to beat the accuracy of fixed depth policies using considerably less computational power. Since the hyperparameters for gating can be varied as desired, our architecture can trade accuracy for computational cost depending on the needs of the task. The cross-correlation maps for select video frames can be viewed in figure 7.

6 Limitations

Our work is limited by a few key factors:

- We finetune purely on VOT2016 data unlike most other submissions which use ImageNet VID, COCO, or other datasets to augment training. VOT2016 dataset is considerably smaller than ImageNet VID and the videos are considered more difficult to track.
- We do not perform bounding box resizing, which makes a very significant difference for videos where the object changes size or orientation. Furthermore, our final cross-correlation map resolution is 16×16 which coarser than most other state-of-the-art trackers.

7 Conclusions

The goal of the work described in this paper is to design an architecture and an adaptive policy that can take advantage of the fact that some frames of a video are easier to track than others. In this paper, we proposed such an architecture and demonstrated that it can track at accuracies competitive with state-of-the-art trackers. Furthermore, we showed that using our learned adaptive policy, we can use conditional computation to reduce the computational cost of visual tracking without losing commensurate representational power of the model. Thus, we can save computation on "easy" frames

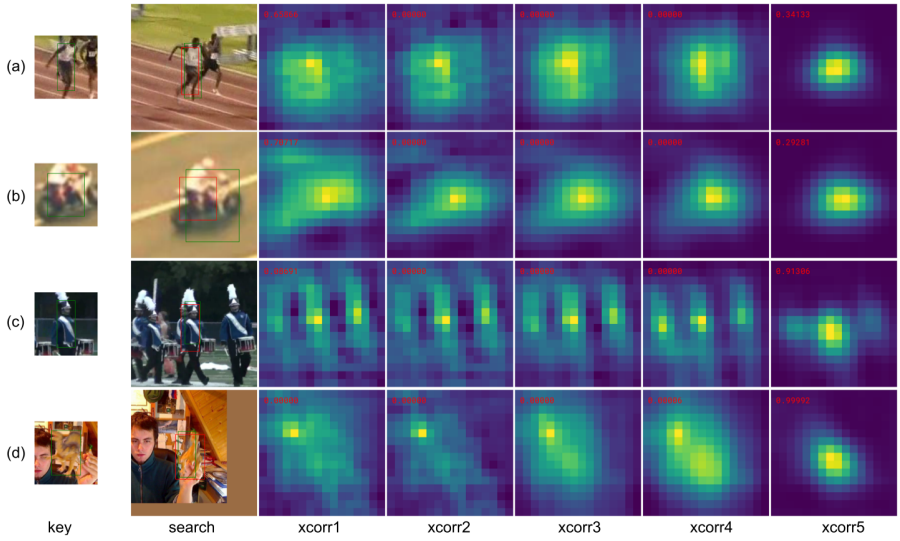

Figure 7: Select tracking outputs (green box is ground truth, red box is prediction, red numbers are confidence weights). In (a) and (b), the tracker learns that xcorr1 is sufficient for tracking. In (c) and (d), tracking is more difficult and the tracker learns that it needs to compute xcorr5 in order to confidently track the object.

without losing representational power of the network on more complex frames that require deeper features to track.

Though the work in this paper specifically addresses video tracking, the ideas can be extended to other computer vision tasks that use convolutional neural networks as feature extractors. For example, in image classification, it is already possible to use intermediate layers for classification (e.g. GoogLeNet). Using gating functions, it may be possible to dynamically determine whether these intermediate classifiers are sufficient for the image being classified.

8 Future work

There are many opportunities for incremental improvements by addressing some of the missing features such as multi-scale tracking and better hand-designed features for gating. Though this work investigates adaptive policies with respect to the depth of the network, there are other promising directions for conditional computation including: spatial location (e.g. use optical flow to focus attention), frame skipping (e.g. use frame features to determine how many frames ahead to evaluate rather than evaluating each frame), and fine-grained depth control (e.g. use intermediate cross-correlation maps to determine which pixels require additional depth).

Acknowledgments

Thank you to Adam Harley, Yijie Wang, and the rest of Katerina's reading group for providing hardware help and valuable ideas for this project. Thank you to Kris Kitani for being on my DAP committee and providing feedback on my paper and presentation.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] E. Bengio, P. Bacon, J. Pineau, and D. Precup. Conditional computation in neural networks for faster models. *CoRR*, abs/1511.06297, 2015.

[3] Y. Bengio, N. Léonard, and A. C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

[4] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision*, pages 850–865. Springer, 2016.

[5] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. *CoRR*, abs/1504.06852, 2015.

[6] E. Hoffer and N. Ailon. Deep metric learning using triplet network. *CoRR*, abs/1412.6622, 2014.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[8] G. Koch. *Siamese neural networks for one-shot image recognition*. PhD thesis, University of Toronto, 2015.

[9] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, L. Čehovin, T. Vojir, G. Häger, A. Lukežič, and G. Fernandez. The visual object tracking vot2016 challenge results. Springer, Oct 2016.

[10] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernández, T. Vojir, G. Hager, G. Nebehay, and R. Pflugfelder. The visual object tracking vot2015 challenge results. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 1–23, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[14] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538, 2017.

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[16] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[18] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[19] G. Zhu, F. Porikli, and H. Li. Robust visual tracking with deep convolutional neural network based object proposals on pets. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2016.