Adaptive Depth Computational Policies for Visual Tracking

Chris Ying and Katerina Fragkiadaki

Machine Learning Department, Carnegie Mellon University

Objectives

- Design and train a **fully convolutional neural network** to perform metric learning at multiple network depths
- Learn an adaptive policy to control the depth of evaluation at runtime to **balance accuracy and computational cost**

Background

Video tracking.

Input: Labelled **key frame** with object bounding box, unlabelled subsequent **search frames**
Output: Bounding box of object in search frames

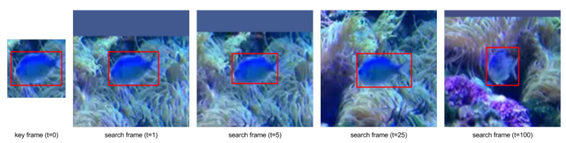

Figure 1: Preprocessed key and search frames extracted from VOT2016. Bounding boxes added for visualization.

Observation: some frames are "harder" to track than others (e.g. due to occlusion or change in shape)

Fully convolutional Siamese network.

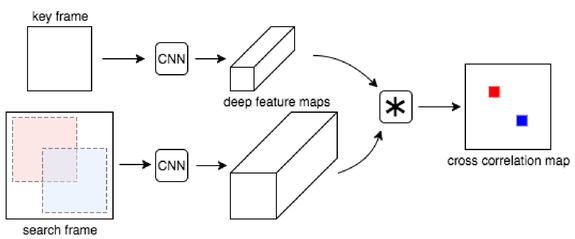

Figure 2: Deep feature maps are extracted from key and search frame using the same fully convolutional network. Metric learning implemented via 2D cross-correlation[1] ($*$). Output is a spatial map with similarity score at each coordinate.

Observation: CNNs are feature extractors for images, the intermediate activations are also feature maps. Deeper feature maps encode higher-level information but also take more computation.

Key idea: learn to use different network depths for different video frames depending on how difficult it is to perform tracking.

[1] 2D cross-correlation can be efficiently implemented efficiently on GPU via 2D convolution by treating the key frame deep feature map as a convolutional filter with 1 output channel.

Methods

Target map G is a 2D Gaussian centered at the ground truth object offset. Loss of i-th cross-correlation map C_i is softmax cross-entropy loss:

$$C_i'(x,y) = \frac{\exp(C_i(x,y))}{\sum_{x,y}\exp(C_i(x,y))} \quad (1)$$

$$L_i(C_i, G) = -\frac{1}{|C|}\sum_{x,y} C_i'(x,y)\log(G(x,y)) \quad (2)$$

Train phase 1: training convolutional weights:

$$L_{\text{finetune(deepest)}} = L_5 \qquad L_{\text{finetune(all)}} = \sum_{i=1}^{5} L_i \quad (3)$$

Gating function at i-th depth is a linear predictor φ_i on top of hand-designed features v_i (incl. kurtosis, entropy) extracted from each cross-correlation map with sigmoid activation. Output values measure "confidence" of the current layer's map (designed to sum to 1.0):

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

Train phase 2: training gate function weights:

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

Prediction policies:

fixed depth: xcorr1, xcorr2, xcorr3, xcorr4, xcorr5
soft-gating: c_i-weighted average of xcorr layers
hard-gating: shallowest layer with $c_i > T$

Results

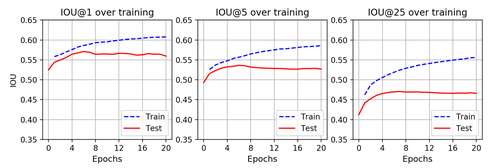

Figure 3: Train and test curves over epochs trained. Intersection-over-union (IOU) averaged over up-to 1, 5, and 25 frames ahead of the key frame. Current state-of-the-art trackers perform at $0.5 \sim 0.6$ IOU.

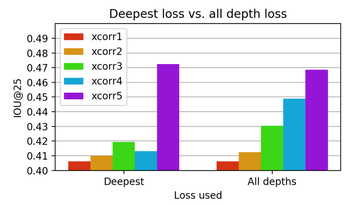

Figure 4: Comparison of finetuning with loss on deepest layer only vs. all layers. Training with deepest layer yields slightly higher accuracy at xcorr5 at the cost of lower accuracy at shallower depths.

Table 1: Theoretical floating point operations (FLOPs) per key-search pair for prediction policies

Pred. 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

Model Architecture

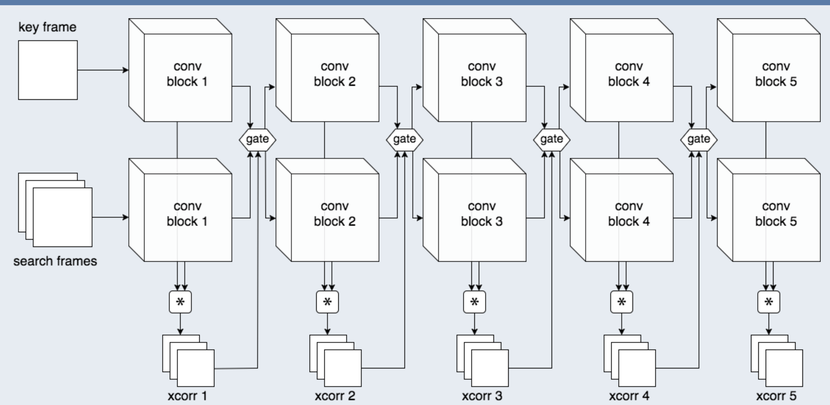

Figure 5: End-to-end architecture for visual tracking. Convolutional blocks contain 2-4 convolutional layers with ReLU activation followed by max pool. Layers are adapted from 19-layer VGG architecture using pre-trained weights on ImageNet for classification (fully connected layers removed). Implemented in TensorFlow. Runs at 57 FPS (xcorr1) and 38 FPS (xcorr5) on NVIDIA TITAN X and Intel Xeon E5-2630 v3.

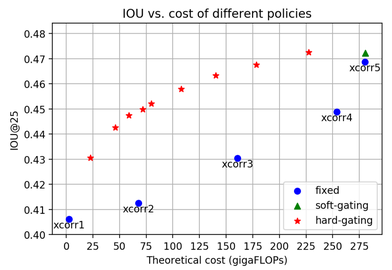

Figure 6: Comparison of accuracy and cost of different prediction policies. Hard-gating is hyperparameter sensitive so values at various settings are reported.

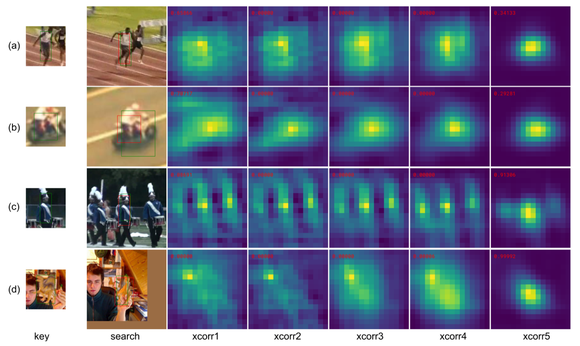

Figure 7: Select tracking outputs (green box is ground truth, red box is prediction, red numbers are confidence weights). In (a) and (b), the tracker learns that xcorr1 is sufficient for tracking. In (c) and (d), tracking is more difficult and the tracker learns that it needs to compute xcorr5 in order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

Conclusions

- Fully convolutional Siamese networks work reasonably well for object tracking without needing to train on the object being tracked during test time
- Using conditional computation via gating can **save computation without losing commensurate representational power**

Future Work

- Improve accuracy by increasing resolution of cross-correlation map localized on coarse bounding box prediction
- Perform test time training on key and search frames to improve tracking for subsequent frames