Clockwork Convnets for Video Semantic Segmentation

Evan Shelhamer* Kate Rakelly* Judy Hoffman* Trevor Darrell
{shelhamer,rakelly,jhoffman,trevor}@cs.berkeley.edu

UC Berkeley

Abstract. Recent years have seen tremendous progress in still-image segmentation; however the naïve application of these state-of-the-art algorithms to every video frame requires considerable computation and ignores the temporal continuity inherent in video. We propose a video recognition framework that relies on two key observations: 1) while pixels may change rapidly from frame to frame, the semantic content of a scene evolves more slowly, and 2) execution can be viewed as an aspect of architecture, yielding purpose-fit computation schedules for networks. We define a novel family of "clockwork" convnets driven by fixed or adaptive clock signals that schedule the processing of different layers at different update rates according to their semantic stability. We design a pipeline schedule to reduce latency for real-time recognition and a fixed-rate schedule to reduce overall computation. Finally, we extend clockwork scheduling to adaptive video processing by incorporating data-driven clocks that can be tuned on unlabeled video. The accuracy and efficiency of clockwork convnets are evaluated on the Youtube-Objects, NYUD, and Cityscapes video datasets.

1 Introduction

Semantic segmentation is a central visual recognition task. End-to-end convolutional network approaches have made progress on the accuracy and execution time of still-image semantic segmentation, but video semantic segmentation has received less attention. Potential applications include UAV navigation, autonomous driving, archival footage recognition, and wearable computing. The computational demands of video processing are a challenge to the simple application of image methods on every frame, while the temporal continuity of video offers an opportunity to reduce this computation.

Fully convolutional networks (FCNs) [1–3] have been shown to obtain remarkable results, but the execution time of repeated per-frame processing limits application to video. Adapting these networks to make use of the temporal continuity of video reduces inference computation while suffering minimal loss in recognition accuracy. The temporal rate of change of features, or feature "velocity", across frames varies from layer to layer. In particular, deeper layers in the feature hierarchy change more slowly than shallower layers over video sequences. We propose that network execution can be viewed as an aspect of architecture and define the "clockwork" FCN (c.f. clockwork recurrent networks [4]). Combining these two insights, we group the layers of the network into stages, and set separate update rates for these levels of representation. The

* Authors contributed equally.

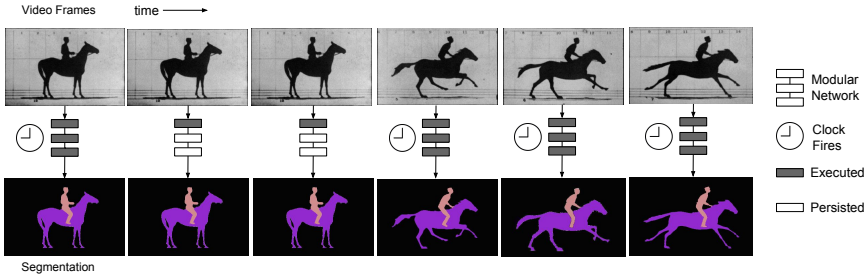


Fig. 1: Our adaptive clockwork method illustrated with the famous *The Horse in Motion* [9], captured by Eadweard Muybridge in 1878 at the Palo Alto racetrack. The clock controls network execution: past the first stage, computation is scheduled only at the time points indicated by the clock symbol. During static scenes cached representations persist, while during dynamic scenes new computations are scheduled and output is combined with cached representations.

execution of a stage on a given frame is determined by either a fixed clock rate ("fixed-rate") or data-driven ("adaptive"). The prediction for the current frame is then the fusion (via the skip layer architecture of the FCN) of these computations on multiple frames, thus exploiting the lower resolution and slower rate-of-change of deeper layers to share information across frames.

We demonstrate the efficacy of the architecture for both fixed and adaptive schedules. We show results on multiple datasets for a pipelining schedule designed to reduce latency for real-time recognition as well as a fixed-rate schedule designed to reduce computation and hence time and power. Next we learn the clock-rate adaptively from the data, and demonstrate computational savings when little motion occurs in the video without sacrificing recognition accuracy during dynamic scenes. We verify our approach on synthetic frame sequences made from PASCAL VOC [5] and evaluate on videos from the NYUDv2 [6], YouTube-Objects [7], and Cityscapes [8] datasets.

2 Related Work

We extend fully convolutional networks for image semantic segmentation to video semantic segmentation. Convnets have been applied to video to learn spatiotemporal representations for classification and detection but rarely for dense pixelwise, frame-by-frame inference. Practicality requires network acceleration, but generic techniques do not exploit the structure of video. There is a large body of work on video segmentation, but the focus has not been on *semantic* segmentation, nor are methods computationally feasible beyond short video shots.

Fully Convolutional Networks A fully convolutional network (FCN) is a model designed for pixelwise prediction [1]. Every layer in an FCN computes a local operation, such as convolution or pooling, on relative spatial coordinates. This locality makes the network capable of handling inputs of any size while producing output of corresponding dimensions. Efficiency is preserved by computing single, dense forward

inference and backward learning passes. Current classification architectures – AlexNet [10], GoogLeNet [11], and VGG [12] – can be cast into corresponding fully convolutional forms. These networks are learned end-to-end, are fast at inference and learning time, and can be generalized with respect to different image-to-image tasks. FCNs yield state-of-the-art results for semantic segmentation [1], boundary prediction [13], and monocular depth estimation [2]. While these tasks process each image in isolation, FCNs extend to video. As more and more visual data is captured as video, the baseline efficiency of fully convolutional computation will not suffice.

Video Networks and Frame Selection Time can be incorporated into a network by spatiotemporal filtering or recurrence. Spatiotemporal filtering, i.e. 3D convolution, can capture motion for activity recognition [14, 15]. For video classification, networks can integrate over time by early, late, or slow fusion of frame features [15]. Recurrence can capture long-term dynamics and propagate state across time, as in the popular long short-term memory (LSTM) [16]. Joint convolutional-recurrent networks filter within frames and recur across frames: the long-term recurrent convolutional network [17] fuses frame features by LSTM for activity recognition and captioning. Frame selection reduces computation by focusing computational resources on important frames identified by the model: space-time interest points [18] are video keypoints engineered for sparsity, and a whole frame selection and recognition policy can be learned end-to-end for activity detection [19]. These video recognition approaches do not address frame-by-frame, pixelwise output. For optical flow, an intrinsically temporal task, a cross-frame FCN is state-of-the-art among fast methods [3].

Network Acceleration Although FCNs are fast, video demands computation that is faster still, particularly for real-time inference. The spatially dense operation of the FCN amortizes the computation of overlapping receptive fields common to contemporary architectures. However, the standard FCN does nothing to temporally amortize the computation of sequential inputs. Computational concerns can drive architectural choices. For instance, GoogLeNet requires less computation and memory than VGG, although its segmentation accuracy is worse [1]. Careful but time-consuming model search can improve networks within a fixed computational budget [20]. Methods to reduce computation and memory include reduced precision by weight quantization [21], low-rank approximations with clustering, [22], low-rank approximations with end-to-end tuning [23], and kernel approximation methods like the fast food transformation [24]. None of these generic acceleration techniques harness the frame-to-frame structure of video. The proposed clockwork speed-up is orthogonal and compounds any reductions in absolute inference time. Our clockwork insight holds for all layered architectures whatever the speed/quality operating point chosen.

Semantic Segmentation Much work has been done to address the problem of segmentation in video. However, the focus has not been on semantic segmentation. Instead research has addressed spatio-temporal "supervoxels" [25, 26], unsupervised and motion-driven object segmentation [27–29], or weakly supervising the segmentation of tagged videos [30–32]. These methods are not suitable for real-time or the complex multi-class, multi-object scenes encountered in semantic segmentation settings. Fast Object Segmentation in Unconstrained Videos [28] infers only figure-ground segmentation at 0.5s/frame with offline computed optical flow and superpixels. Although

its proposals have high recall, even when perfectly parallelized [29] this method takes $>$ 15s/frame and a separate recognition step is needed for semantic segmentation. In contrast the standard FCN computes a full semantic segmentation in 0.1s/frame.

3 Fast Frames and Slow Semantics

Our approach is inspired by observing the time course of learned, hierarchical features over video sequences. Drawing on the local-to-global idea of skip connections for fusing global, deep layers with local, shallow layers, we reason that the semantic representation of deep layers is relevant across frames whereas the shallow layers vary with more local, volatile details. Persisting these features over time can be seen as a temporal skip connection.

Measuring the relative difference of features across frames confirms the temporal coherence of deeper layers. Consider a given score layer (a linear predictor of pixel class from features), ℓ, with outputs $S_\ell \in [K \times H \times W]$, where K is the number of categories and H, W is the output dimensions for layer ℓ. We can compute the difference at time t with a score map distance function d_{sm}, chosen to be the hamming distance of one hot encodings.

$$d_{\text{sm}}(S_\ell^t, S_\ell^{t-1}) = d_{\text{hamming}}(\phi(S_\ell^t), \phi(S_\ell^{t-1}))$$

Table 1 reports the average of these temporal differences for the score layers, as computed over all videos in the YouTube-Objects dataset [7]. It is perhaps unsurprising that the deepest score layer changes an order of magnitude less than the shallower layers on average. We therefore hypothesize that caching deeper layer scores from past frames can inform the inference of the current frame with relatively little reduction in accuracy.

The slower rate of change of deep layers can be attributed to architectural and learned invariances. More pooling affords more robustness to translation and noise, and learned features may be tuned to the supervised classes instead of general appearance.

score layer	temporal difference	depth	semantic accuracy
`pixels`	$.26 \pm .18$	0	-
`pool3`	$.11 \pm .06$	9	9.6%
`pool4`	$.11 \pm .06$	13	20.7%
`fc7`	$.02 \pm .02$	19	65.5%

Table 1: The average temporal difference over all YouTube-Objects videos of the respective pixelwise class score outputs from a spectrum of network layers. The deeper layers are more stable across frames – that is, we observe supervised convnet features to be "slow" features [33]. The temporal difference is measured as the proportion of label changes in the output. The layer depth counts the distance from the input in the number of parametric and non-linear layers. Semantic accuracy is the intersection-over-union metric on PASCAL VOC of our frame processing network fine-tuned for separate output predictions (Section 5).

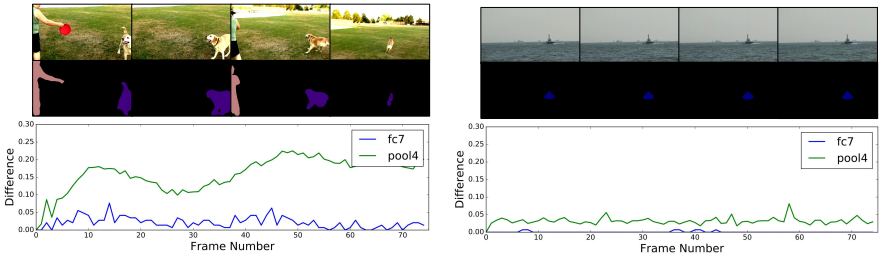


Fig. 2: The proportional difference between adjacent frames of semantic predictions from a mid-level layer (`pool4`, green) and the deepest layer (`fc7`, blue) are shown for the first 75 frames of two videos. We see that for a video with lots of motion (left) the difference values are large while for a relatively static video (right) the difference values are small. In both cases, the differences of the deeper `fc7` are smaller than the differences of the shallower `pool4`. The "velocity" of deep features is slow relative to shallow features and most of all the input. At the same time, the differences between shallow and deep layers are dependent since the features are compositional; this motivates our adaptive clock updates in Section 4.3

While deeper layers are more stable than shallower layers, for videos with enough motion the score maps throughout the network may change substantially. For example, in Figure 2 we show the differences for the first 75 frames of a video with large motion (left) and with small motion (right). We would like our network to adaptively update only when the deepest, most semantic layer (`fc7`) score map is likely to change. We notice that though the intermediate layer (`pool4`) difference is always larger than the deepest layer difference for any given frame, the `pool4` differences are much larger for the video with large motion than for the video with relatively small motion. This observation forms the motivation for using the intermediate differences as an indicator to determine the firing of an adaptive clock.

4 A Clockwork Network

We adapt the fully convolutional network (FCN) approach for image-to-image mapping [1] to video frame processing. While it is straightforward to perform inference with a still-image segmentation network on every video frame, this naïve computation is inefficient. Furthermore, disregarding the sequential nature of the input not only sacrifices efficiency but discards potential temporal recognition cues. The temporal coherence of video suggests the persistence of visual features from prior frames to inform inference on the current frame. To this end we define the clockwork FCN, inspired by the clockwork recurrent network [4], to carry temporal information across frames. A generalized notion of clockwork relates both of these networks.

We consider both throughput and latency in the execution of deep networks across video sequences. The inference time of the regular FCN-8s at $\sim$ 100ms per frame of size 500×500 on a standard GPU can be too slow for video. We first define fixed clocks then extend to adaptive and potentially learned clockwork to drive network processing. Whatever the task, any video network can be accelerated by our clockwork technique.

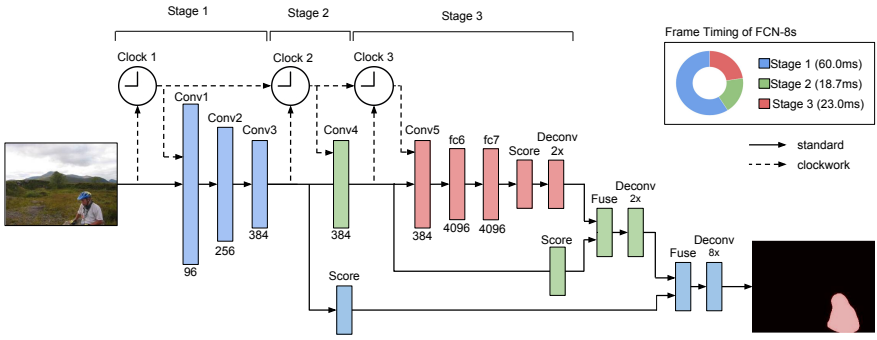


Fig. 3: The clockwork FCN with its stages and corresponding clocks.

A schematic of our clockwork FCN is shown in Figure 3.

There are several choice points in defining a clockwork architecture. We define a novel, generalized clockwork framework, which can purposely schedule deeper layers more slowly than shallower layers. We form our modules by grouping the layers of a convnet to span the feature hierarchy. Our networks persists both state and output across time steps. The clockwork recurrent network of [4], designed for long-term dependency modeling of time series, is an instance of our more general scheme for clockwork computation. The differences in architecture and outputs over time between clockwork recurrence and our clockwork are shown in Figure 4.

While different, these nets can be expressed by generalized clockwork equations

$$y_H^{(t)} = f_T \left(C_H^{(t)} \odot f_H(y_H^{(t-1)}) + C_I^{(t)} \odot f_I(x^{(t)}) \right) \quad (1)$$

$$y_O^{(t)} = f_O \left(C_O^{(t)} \odot f_H(y_H^{(t)}) \right) \quad (2)$$

with the state update defined by Equation 1 and the output defined by Equation 2. The data $x^{(t)}$, hidden state $y_H^{(t)}$ output $y_O^{(t)}$ vary with time t. The functions f_I, f_H, f_O, f_T define input, hidden state, output, and transition operations respectively and are fixed across time. The input, hidden, and output clocks $C_I^{(t)}, C_H^{(t)}, C_O^{(t)}$ modulate network operations by the elementwise product $\odot$ with the corresponding function evaluations. We recover the standard recurrent network (SRN), clockwork recurrent network (clock RN), and our network (clock FCN) in this family of equations. The settings of functions and clocks are collected in Table 2.

Inspired by the clockwork RN, we investigate persisting features and scheduling layers to process video with a semantic segmentation convnet. Recalling the lessened semantic rate of deeper layers observed in Section 3, the skip layers in FCNs originally included to preserve resolution by fusing outputs are repurposed for this staged computation. We cache features and outputs over time at each step to harness the continuity of video. In contrast, the clockwork RN persists state but output is only made according to the clock, and each clockwork RN module is connected to itself and all slower modules across time whereas a module in our network is only connected to itself across time.

network	f_I	f_H	f_O	f_T	C_I	C_H	C_O
SRN	W_I	W_H	TanH	TanH	$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$
clock RN	W_I	W_H	TanH	TanH	C	C	C
clock FCN	$\circ$	I	ReLU	I	C	$\overline{C}$	$\mathbb{1}$

Table 2: The standard recurrent network (SRN), clockwork recurrent network (clock RN), and our network (clock FCN) in generalized clockwork form. The recurrent networks have learned hidden weights W_H and non-linear transition functions f_T, while clock FCN persists state by the identity I. Both recurrent modules are flat with linear input weights W_I, while clock FCN modules have hierarchical features by layer composition $\circ$. The SRN has trivial constant, all-ones $\mathbb{1}$ clocks. The clock RN has a shared input, hidden, and output clock with exponential rates. Our clock FCN has alternating input and hidden clocks $C, \overline{C}$ to compute or cache and has a constant, all-ones $\mathbb{1}$ output clock to fuse output on every frame.

4.1 Execution as Architecture

Clockwork architectures partition a network into modules or stages that are executed according to different schedules. In the standard view, the execution of an architecture is an all-or-nothing operation that follows from the definition of the network. Relaxing the strict identification of architecture and execution instead opens up a range of potential schedules. These schedules can be encompassed by the introduction of one first-class architectural element: the clock.

A clock defines a dynamic cut in the computation graph of a network. As clocks mask state in the representation, as detailed in Equations 1 and 2, clocks likewise mask execution in the computation. When a clock is on, its edges are intact and execution traverses to the next nodes/modules. When a clock is off, its edges are cut and execution is blocked. Alternatives such as computing the next stage or caching a past stage can be scheduled by a paired clock C and counter-clock $\overline{C}$ with complementary sets of edges. Any layer (or composition of layers) with binary output can serve as a clock. As a layer, a clock can be fixed or learned. For instance, the following are simple clocks of the form $f(x, t)$ for features x and time t:

- 1 to always execute
- $t \equiv 0 \pmod 2$ to execute every other time
- $\|x_t - x_{t-1}\| > \theta$ to execute for a difference threshold

4.2 Networks in Time

Having incorporated scheduling into the network with clocks, we can optimize the schedule for various tasks by altering the clocks.

Pipelining To reduce latency for real-time recognition we pipeline the computation of sequential frames analogously to instruction pipelining in processors. We instantiate a three-stage pipeline, in which stage 1 reflects frame i, stage 2 frame $i-1$, and stage 3 frame $i-2$. The total time to process the frame is the time of the longest stage, stage 1

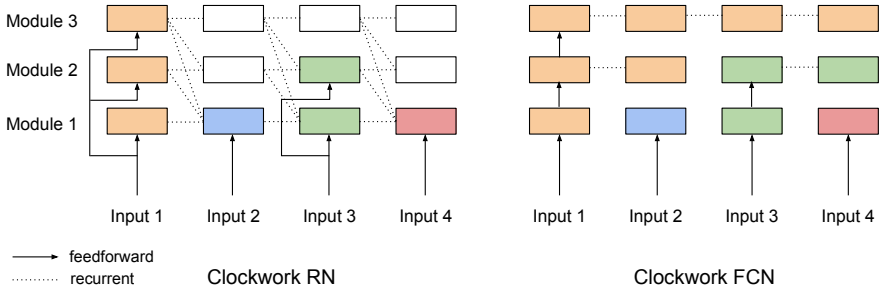


Fig. 4: A comparison of the layer connectivity and time course of outputs in the clockwork recurrent network [4] and in our clockwork FCN. Module color marks the time step of evaluation, and blank modules are disconnected from the network output. The clock RN is flat with respect to the input while our network has a hierarchical feature representation. Each clock RN module is temporally connected to itself and slower modules while in our network each module is only temporally connected to itself. Features persist over time in both architectures, but in our architecture they contribute to the network output at each step.

in our pipeline, plus the time for interpolating and fusing outputs. Our 3-stage pipeline FCN reduces latency by 59%. A 2-stage variation further balances latency and accuracy.

Fixed-Rate To reduce overall computation we limit the execution rates of stages and persist features across frames for skipped stages. Given the learned invariance and slow semantics of deep layers observed in Section 3, the deeper layers can be executed at a lower rate to save computation while other stages update. These clock rates are free parameters in the schedule for exchanging inference speed and accuracy. We again divide the network into three stages, and compare rates for the stages. The exponential clockwork schedule is the natural choice of halving the rate at each stage for more efficiency. The alternating clockwork schedule consolidates the earlier stages to execute these on every frame and executes the last stage on every other frame for more accuracy. These different sets of rates cover part of the accuracy/efficiency spectrum.

The current stages are divided into the original score paths of the FCN-8s architecture, but they need not be. One could prioritize latency, spatial refinement, or certain output classes by rebalancing the computation. It is possible to partially compute a span of layers and defer their full execution to a following stage; this can be accomplished by sparse evaluation through dynamic striding and dilation [34]. In principle the stage progression can be decided online in lieu of fixing a schedule for all inference. We turn to adaptive clockwork for deciding execution.

4.3 Adaptive Clockwork

All of the clocks considered thus far have been fixed functions of time but not the data. Setting these clocks gives rise to many schedules that can be tuned to a given task or video, but this introduces a tedious dimension of model search. Much of the video captured in the wild is static and dynamic in turn with a variable amount of motion and semantic progression at any given time. Choosing many stages or a slow clock rate may

reduce computation, but will likewise result in a steep decline in accuracy for dynamic scenes. Conversely, faster update rates or fewer stages may capture transitory details but will needlessly compute and re-compute stable scenes. Adaptive clocks fire based on the input and network state, resulting in a responsive schedule that varies with the dynamism of the scene. The clock can fire according to any function of the input and network state. A difference clock can fire on the temporal difference of a feature across frames. A confidence clock can fire on peaks in the score map for a single frame. This approach extends inference from a pre-determined architecture to a set of architectures to choose from for each frame, relying on the full FCN for high accuracy in dynamic scenes while taking advantage of cached representations in more static scenes.

$$\text{threshold clock } \|x_t - x_{t-1}\| > \theta \qquad \text{learned clock } f_\theta(x_t, x_{t-1})$$

The simplest adaptive clock is a threshold, but adaptive clocks could likewise be learned (for example as a temporal convolution across frames). The threshold can be optimized for a specific tradeoff along the accuracy/efficiency curve. Given the hierarchical dependencies of layers and the relative stability of deep features observed in Section 3, we threshold differences at a shallower stage for adaptive scheduling of deeper stages. The sensitivity of the adaptive clock can even be set on unannotated video by thresholding the proportional temporal difference of output labels as in Table 1. Refer to Section 5.3 for the results of threshold-adaptive clockwork with regard to clock rate and accuracy.

5 Results

Our base network is FCN-8s, the fully convolutional network of [1]. The architecture is adapted from the VGG16 architecture [12] and fine-tuned from ILSVRC pre-training. The net is trained with batch size one, high momentum, and all skip layers at once.

In our experiments we report two common metrics for semantic segmentation that measure the region intersection over union (IU):

- mean IU: $(1/n_{\text{cl}}) \sum_i n_{ii} / \left(t_i + \sum_j n_{ji} - n_{ii}\right)$
- frequency weighted IU: $\left(\sum_k t_k\right)^{-1} \sum_i t_i n_{ii} / \left(t_i + \sum_j n_{ji} - n_{ii}\right)$

for n_{ij} the number of pixels of class i predicted to belong to class j, where there are n_{cl} different classes, and for $t_i = \sum_j n_{ij}$ the total number of pixels of class i.
We evaluate our clockwork FCN on four video semantic segmentation datasets.

Synthetic sequences of translated scenes We first validate our method by evaluating on synthetic videos of moving crops of PASCAL VOC images [5] in order to score on a ground truth annotation at every frame. For source data, we select the 736 image subset of the PASCAL VOC 2011 segmentation validation set used for FCN-8s validation in [1]. Video frames are generated by sliding a crop window across the image by a predetermined number of pixels, and generated translations are vertical or horizontal according to the portrait or landscape aspect of the chosen image. Each synthetic video is six frames long. For each seed image, a "fast" and "slow" video is made with 32 pixel and 16 pixel frame-to-frame displacements respectively.

NYU-RGB clips The NYUDv2 dataset [6] collects short RGB-D clips and includes a segmentation benchmark with high-quality but temporally sparse pixel annotations (every tenth video frame is labeled). We run on video from the "raw" clips subsampled 10X and evaluate on every labeled frame. We consider RGB input alone as the depth frames of the full clips are noisy and uncurated. Our pipelined and fixed-rate clockwork FCNs are run on the entire clips and accuracy is reported for those frames included in the segmentation test set.

Youtube-Objects The Youtube-Objects dataset [7] provides videos collected from Youtube that contain objects from ten PASCAL classes. We restrict our attention to a subset of the videos that have pixelwise annotations by [35] as the original annotations include only initial frame bounding boxes. This subset was drawn from all object classes, and contains 10,167 frames from 126 shots, for which every 10th frame is human-annotated. We run on only annotated frames, effectively 10X subsampling the video. We directly apply our networks derived from PASCAL VOC supervision and do not fine-tune to the video annotations.

Cityscapes The Cityscapes dataset [8] collects frames from video recorded at 17hz by a car-mounted camera while driving through German cities. While annotations are temporally sparse, the preceding and following input frames are provided. Our network is learned on the `train` split and then all schedules are evaluated on `val`.

5.1 Pipelining

16 pixel shift	Time (% of full)	Mean IU	fwIU	Mean IU-bdry	fwIU-bdry
3-Stage Baseline	59%	9.2	52.6	6.1	9.4
3-Stage Pipeline	59%	56.0	76.5	44.6	42.9
2-Stage Baseline	77%	22.5	64.7	16.6	21.9
2-Stage Pipeline	77%	**63.3**	**81.7**	**52.3**	**51.0**
Frame Oracle	100%	65.9	83.6	57.0	56.3
32 pixel shift	Time (% of full)	Mean IU	fwIU	Mean IU-bdry	fwIU-bdry
3-Stage Baseline	59%	9.2	52.6	6.0	9.4
3-Stage Pipeline	59%	45.5	67.4	37.7	36.0
2-Stage Baseline	77%	22.4	62.8	16.2	21.7
2-Stage Pipeline	77%	**57.8**	**76.6**	**46.6**	**45.1**
Frame Oracle	100%	65.6	82.6	55.8	55.3

Table 3: Pipelined segmentation of translated PASCAL sequences. Synthesized video of translating PASCAL scenes allows for assessment of the pipeline at every frame. The pipelined FCN segments with higher accuracy in the same time envelope as the every-other-frame evaluation of the full FCN. Metrics are computed on the standard masks and a 10-pixel band at boundaries.

Pipelined execution schedules reduce latency by producing an output each time the first stage is computed. Later stages are persisted from previous frames and their outputs

are fused with the output of the first stage computed on the current frame. The number of stages is determined by the number of clocks. We consider a full **3-stage pipeline** and a condensed **2-stage pipeline** where the stages are defined by the modules in Figure 3. In the pipelined schedule, all clock rates are set to 1, but clocks fire simultaneously to update every stage in parallel. This is made possible by asynchrony in stage state, so that a later stage is independent of the current frame but not past frames.

To assess our pipelined accuracy and speed, we compare to reference methods that bound both recognition and time. A frame oracle evaluates the full FCN on every frame to give the best achievable accuracy for the network independent of timing. As latency baselines for our pipelines, we truncate the FCN to end at the given stage. Both of our staged, pipelined schedules execute at lower latency than the oracle with better accuracy for fixed latency than the baselines. We verify these results on synthetic PASCAL sequences as reported in Table 3. Results on PASCAL, NYUD, and YouTube are reported in Table 4.

Our pipeline scheduled networks reduce latency with minimal accuracy loss relative to the standard FCN run on each frame without time restriction. These quantitative results demonstrate that the deeper layer representations from previous frames contain useful information that can be effectively combined with low-level predictions for the current frame.

		NYUD		Youtube		Pascal Shift 16	
Schedule	Time (% of full)	Mean IU	fwIU	Mean IU	fwIU	Mean IU	fwIU
3-Stage Baseline	59%	8.1	22.2	12.2	74.2	9.2	54.7
3-Stage Pipeline	59%	25.1	38.0	58.1	87.0	56.0	76.5
2-Stage Baseline	77%	16.5	32.1	21.5	7.8	22.5	64.7
2-Stage Pipeline	77%	**26.4**	**39.5**	**64.0**	**89.2**	**63.3**	**81.7**
Frame Oracle	100%	31.1	45.5	70.0	91.5	65.9	83.6

Table 4: Pipelined execution of semantic segmentation on three different datasets. Inference approaches include pipelines of different lengths and a full FCN frame oracle. We also show baselines with comparable latency to the pipeline architectures. Our pipelined network offers the best accuracy of computationally comparable approaches running near frame rate. The loss in accuracy relative to the frame oracle is less than the relative speed-up.

We show a qualitative result for our pipelined FCN on a sequence from the YouTube-Objects dataset [7]. Figure 5 shows one example where our pipeline FCN is particularly useful. Our network quickly detects the occlusion of the car while the baseline lags and does not immediately recognize the occlusion or reappearance.

5.2 Fixed-Rate

Fixed-rate clock schedules reduce overall computation relative to full, every frame evaluation by assigning different update rates to each stage such that later stages are exe-

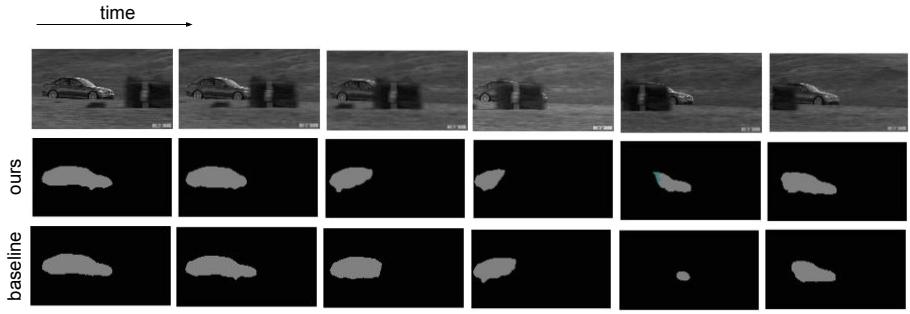


Fig. 5: Pipelined vs. standard FCN on YouTube video. Our method is able to detect the occlusion of the car as it is happening unlike the lagging baseline computed on every other frame.

cuted less often. Rates can be set aggressively low for extreme efficiency or conservatively high to maintain accuracy while sparing computation. The **exponential clockwork** schedule executes the first stage on every frame then updates following stages exponentially less often by halving with each stage. The **alternating clockwork** schedule combines stages 2 and 3, executes the first stage on every frame, then schedules the following combined stage every other frame.

A frame oracle that evaluates the full FCN on *every* frame is the reference model for accuracy. Evaluating the full FCN on *every other* frame is the reference model for computation. Due to the distribution of execution time over stages, this is faster than either clockwork schedule, though clockwork offers higher accuracy. Alternating clockwork achieves higher accuracy than the every other frame reference. See Table 5.

16 pixel shift	Clock Rates	Mean IU	fwIU	Mean IU-bdry	fwIU-bdry
Skip Frame Baseline	(2,2,2)	63.0	81.5	60.2	52.2
Exponential	(1,2,4)	61.4	80.4	50.5	49.1
Alternating	(1,1,2)	**64.7**	**82.6**	**54.8**	**53.7**
Frame Oracle	(1,1,1)	65.9	83.6	57.0	56.3
32 pixel shift	**Clock Rates**	**Mean IU**	**fwIU**	**Mean IU-bdry**	**fwIU-bdry**
Skip Frame Baseline	(2,2,2)	59.5	77.9	49.4	48.2
Exponential	(1,2,4)	55.5	74.7	46.3	44.8
Alternating	(1,1,2)	**61.9**	**79.6**	**51.7**	**50.6**
Frame Oracle	(1,1,1)	65.6	82.6	55.8	55.3

Table 5: Fixed-rate segmentation of translated PASCAL sequences. We evaluate the network on synthesized video of translating PASCAL scenes to assess the effect of persisting layer features across frames. Metrics are computed on the standard masks and a 10-pixel band at boundaries.

Exponential clockwork shows degraded accuracy yet takes $1.5\times$ the computation of evaluation on every other frame, so we discard this fixed schedule in favor of adaptive clockwork. Although exponential rates suffice for the time series modeled by the clockwork recurrent network [4], these rates deliver unsatisfactory results for the task of video semantic segmentation. See Table 6 for alternating clockwork results on NYUD, YouTube-Objects, and Cityscapes.

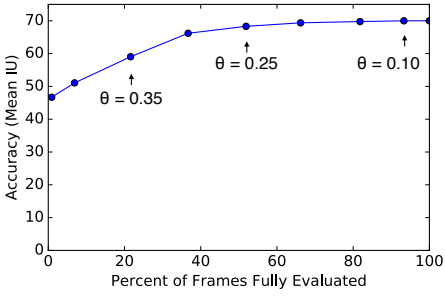
	NYUD		Youtube		Cityscapes	
Schedule	Mean IU	fwIU	Mean IU	fwIU	Mean IU	fwIU
Skip Frame Baseline	27.7	41.3	65.6	89.7	62.1	87.4
Alternating	28.5	42.4	67.0	90.3	**64.4**	**88.6**
Adaptive	**28.9**	**43.3**	**68.5**	**91.0**	61.8	87.6
Frame Oracle	31.1	45.5	70.0	91.4	65.9	83.6

Table 6: Fixed-rate and adaptive clockwork FCN evaluation. We score our network on three datasets with an alternating schedule that executes the later stage every other frame and an adaptive schedule that executes according to a frame-by-frame threshold on the difference in output. The adaptative threshold is tuned to execute the full network on 50% of frames to equalize computation between the alternating and adaptive schedules.

5.3 Adaptive Clockwork

The best clock schedule can be data-dependent and unknown before segmenting a video. Therefore, we next evaluate our adaptive clock rate as described in Section 4.3. In this case the adaptive clock only fully processes a frame if the relative difference in `pool4` score is larger than some threshold θ. This threshold may be interpreted as the the fraction of the score map that must switch labels before the clock updates the upper layers of the network. See Table 6 for adaptive clockwork results on NYUD, YouTube-Objects, and Cityscapes.

We experiment with varying thresholds on the Youtube-Objects dataset to measure accuracy and efficiency. We pick thresholds in $\theta = [0.1, 0.5]$ as well as $\theta = 0.0$ for unconditionally updating on every frame.



Method	% Full Frames	Mean IU
Adaptive [$\theta = 0.10$]	93%	70.0
Adaptive [$\theta = 0.25$]	52%	68.3
Adaptive [$\theta = 0.35$]	21%	59.0
Frame Oracle	100%	70.0

Fig. 6: Adaptive Clockwork performance across the Youtube-Objects dataset. We examine various adaptive difference thresholds θ and plot accuracy (mean IU) against the percentage of frames that the adaptive clock chooses to fully compute. A few corresponding thresholds are indicated.

In Figure 6 (left) we report mean IU accuracy as a function of our adaptive clock firing rate; that is, the percentage of frames the clock decides to fully process in the network. The thresholds which correspond to a few points on this curve are indicated with mean IU (right). Notice that our adaptive clockwork is able to fully process only 52% of the frames while suffering a minimal loss in mean IU ($\theta = 0.25$). This indicates that our adaptive clockwork is capable of discovering semantically stationary scenes and

saves significant computation by only updating when the output score map is predicted to change.

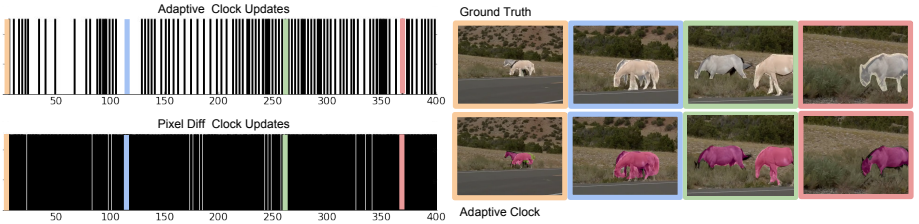


Fig. 7: An illustrative example of our adaptive clockwork method on a video from Youtube-Objects. On the left, we compare clock updates over time (shown in black) of our adaptive clock as well as a clock based on pixel differences. Our adaptive clock updates the full network on only 26% of the frames, determined by the threshold $\theta = 0.25$ on the proportional output label change across frames, while scheduling updates based on pixel difference alone results in updating 90% of the frames. On the right we show output segmentations from the adaptive clockwork network as well as ground truth segments for select frames from dynamic parts of the scene (second and third frames shown) and relatively static periods (first and second frames shown).

For a closer inspection, we study one Youtube video in more depth in Figure 7. We first visualize the clock updates for our adaptive method (top left) and for a simple pixel difference baseline (bottom left), where black indicates the clock is on and the corresponding frame is fully computed. This video has significant change in certain sections (ex: at frame $\sim$ 100 there is zoom and at $\sim$ 350 there is motion) with long periods of relatively little motion (ex: frames $110 - 130$). While the pixel difference metric is susceptible to the changes in minor image statistics from frame to frame, resulting in very frequent updates, our method only updates during periods of semantic change and can cache deep features with minimal loss in segmentation accuracy: compare adaptive clock segmentations to ground truth (right).

6 Conclusion

Generalized clockwork architectures encompass many kinds of temporal networks, and incorporating execution into the architecture opens up many strategies for scheduling computation. We define a clockwork fully convolutional network for video semantic segmentation in this framework. Motivated by the stability of deep features across sequential frames, our network persists features across time in a temporal skip architecture. By exploring fixed and adaptive schedules, we are able to tune processing for latency, overall computation time, and recognition performance. With adaptive, data-driven clock rates the network is scheduled online to segment dynamic and static scenes alike while maintaining accuracy. In this way our adaptive clockwork network is a bridge between convnets and event-driven vision architectures. The clockwork perspective on temporal networks suggests further architectural variations for spatiotemporal video processing.

References

1. Shelhamer, E., Long, J., Darrell, T.: Fully convolutional networks for semantic segmentation. In: PAMI. (2016)
2. Eigen, D., Fergus, R.: Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In: ICCV. (2015) 2650–2658
3. Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., van der Smagt, P., Cremers, D., Brox, T.: Learning optical flow with convolutional networks. In: ICCV. (2015)
4. Koutník, J., Greff, K., Gomez, F., Schmidhuber, J.: A Clockwork RNN. In: ICML. (2014)
5. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. IJCV **88**(2) (June 2010) 303–338
6. Silberman, N., Hoiem, D., Kohli, P., Fergus, R.: Indoor segmentation and support inference from rgbd images. In: ECCV. (2012)
7. Prest, A., Leistner, C., Civera, J., Schmid, C., Ferrari, V.: Learning object class detectors from weakly annotated video. In: CVPR, IEEE (2012) 3282–3289
8. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: CVPR. (2016)
9. Muybridge, E.: The horse in motion. Library of Congress Prints and Photographs Division (1882)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. (2012)
11. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR. (2015)
12. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR. (2015)
13. Xie, S., Tu, Z.: Holistically-nested edge detection. In: ICCV. (2015)
14. Ji, S., Xu, W., Yang, M., Yu, K.: 3d convolutional neural networks for human action recognition. PAMI **35**(1) (2013) 221–231
15. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: CVPR. (2014) 1725–1732
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8) (1997) 1735–1780
17. Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., Darrell, T.: Long-term recurrent convolutional networks for visual recognition and description. In: CVPR. (2015) 2625–2634
18. Laptev, I.: On space-time interest points. IJCV **64**(2-3) (2005) 107–123
19. Yeung, S., Russakovsky, O., Mori, G., Fei-Fei, L.: End-to-end learning of action detection from frame glimpses in videos. In: CVPR. (2016)
20. He, K., Sun, J.: Convolutional neural networks at constrained time cost. In: CVPR. (2015)
21. Vanhoucke, V., Senior, A., Mao, M.Z.: Improving the speed of neural networks on cpus. In: Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop. Volume 1. (2011)
22. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: NIPS. (2014) 1269–1277
23. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. In: BMVC. (2014)
24. Yang, Z., Moczulski, M., Denil, M., de Freitas, N., Smola, A., Song, L., Wang, Z.: Deep fried convnets. In: ICCV. (2015)
25. Grundmann, M., Kwatra, V., Han, M., Essa, I.: Efficient hierarchical graph-based video segmentation. In: CVPR, IEEE (2010) 2141–2148

26. Xu, C., Corso, J.J.: Evaluation of super-voxel methods for early video processing. In: CVPR, IEEE (2012) 1202–1209
27. Shi, J., Malik, J.: Motion segmentation and tracking using normalized cuts. In: ICCV, IEEE (1998) 1154–1160
28. Papazoglou, A., Ferrari, V.: Fast object segmentation in unconstrained video. In: ICCV. (December 2013)
29. Fragkiadaki, K., Arbelaez, P., Felsen, P., Malik, J.: Learning to segment moving objects in videos. In: CVPR. (June 2015)
30. Hartmann, G., Grundmann, M., Hoffman, J., Tsai, D., Kwatra, V., Madani, O., Vijayanarasimhan, S., Essa, I., Rehg, J., Sukthankar, R.: Weakly supervised learning of object segmentations from web-scale video. In: ECCV-W, Springer (2012) 198–208
31. Tang, K., Sukthankar, R., Yagnik, J., Fei-Fei, L.: Discriminative segment annotation in weakly labeled video. In: CVPR, IEEE (2013) 2483–2490
32. Liu, X., Tao, D., Song, M., Ruan, Y., Chen, C., Bu, J.: Weakly supervised multiclass video segmentation. In: CVPR, IEEE (2014) 57–64
33. Wiskott, L., Sejnowski, T.J.: Slow feature analysis: Unsupervised learning of invariances. Neural computation **14**(4) (2002) 715–770
34. Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions. In: ICLR. (2016)
35. Jain, S.D., Grauman, K.: Supervoxel-consistent foreground propagation in video. In: ECCV. (2014)