Mobile Video Object Detection with Temporally-Aware Feature Maps

Mason Liu
Georgia Tech*
masonliuw@gatech.edu

Menglong Zhu
Google
menglong@google.com

Abstract

This paper introduces an online model for object detection in videos designed to run in real-time on low-powered mobile and embedded devices. Our approach combines fast single-image object detection with convolutional long short term memory (LSTM) layers to create an interweaved recurrent-convolutional architecture. Additionally, we propose an efficient Bottleneck-LSTM layer that significantly reduces computational cost compared to regular LSTMs. Our network achieves temporal awareness by using Bottleneck-LSTMs to refine and propagate feature maps across frames. This approach is substantially faster than existing detection methods in video, outperforming the fastest single-frame models in model size and computational cost while attaining accuracy comparable to much more expensive single-frame models on the Imagenet VID 2015 dataset. Our model reaches a real-time inference speed of up to 15 FPS on a mobile CPU.

1. Introduction

Convolutional neural networks [24, 35, 36, 12] have been firmly established as the state-of-the-art in single-image object detection [8, 11, 32, 28, 4]. However, the large memory overhead and slow computation time of these networks have limited their practical applications. In particular, efficiency is a primary consideration when designing models for mobile and embedded platforms. Recently, new architectures [17, 40] have allowed neural networks to run on low computational budgets with competitive performance in single-image object detection. However, the video domain presents the additional opportunity and challenge of leveraging temporal cues, and it has been unclear how to make a comparably efficient detection framework for video scenes. This paper investigates the idea of building upon these single-frame models by adding temporal awareness while preserving their speed and low resource consumption.

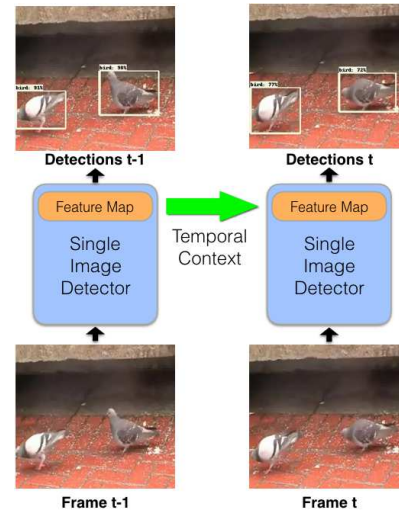Videos contain various temporal cues which can be ex-

Figure 1: Most recent video object detection methods still partially rely on single-image detectors, meaning that a key step in the process fails to utilize temporal information. Our work augments these base detectors by directly incorporating temporal context without sacrificing the efficiency of the fastest single-image detectors.

ploited to obtain more accurate and stable object detection than in single images. Since videos exhibit temporal continuity, objects in adjacent frames will remain in similar locations and detections will not vary substantially. Hence, detection information from earlier frames can be used to refine predictions at the current frame. For instance, since the network is able to view an object at different poses across frames, it may be able to detect the object more accurately. The network will also become more confident about predictions over time, reducing the problem of instability in single-image object detection [37].

Recent work has shown that this continuity extends into the feature space, and intermediate feature maps extracted from neighboring frames of video are also highly correlated [43]. In our work, we are interested in adding temporal awareness in the feature space as opposed to only on the final detections due to the greater quantity of information

---

*This work was done while interning at Google.

available in intermediate layers. We exploit continuity at the feature level by conditioning the feature maps of each frame on corresponding feature maps from previous frames via recurrent network architectures.

Our method generates feature maps with a joint convolutional recurrent unit formed by combining a standard convolutional layer with a convolutional LSTM. The goal is for the convolutional layer to output a feature map hypothesis, which is then fed into the LSTM and fused with temporal context from previous frames to output a refined, temporally aware feature map. Figure 2 shows an illustration of our method. This approach allows us to benefit from advances in efficient still-image object detection, as we can simply extend some convolutional layers in these models with our convolutional-recurrent units. The recurrent layers are able to propagate temporal cues across frames, allowing the network to access a progressively greater quantity of information as it processes the video.

To demonstrate the effectiveness of our model, we evaluate on the Imagenet VID 2015 dataset [33]. Our method compares favorably to efficient single-frame baselines, and we argue that this improvement must be due to successful utilization of temporal context since we are not adding any additional discriminative power. We present model variants that range from 200M to 1100M multiply-adds and 1 to 3.5 million parameters, which are straightforward to deploy on a wide variety of mobile and embedded platforms. To our knowledge, our model is the first mobile-first neural network for video object detection.

The contributions of this paper are as follows:

- We introduce a unified architecture for performing online object detection in videos that requires minimal computational resources.

- We propose a novel role for recurrent layers in convolutional architectures as temporal refinement of feature maps.

- We demonstrate the viability of using recurrent layers in efficiency-focused networks by modifying convolutional LSTMs to be highly efficient.

- We provide experimental results to justify our design decisions and compare our final architecture to various single-image detection frameworks.

2. Related Work

2.1. Object Detection in Images

Recent single-image object detection methods can be grouped into two categories. One category consists of region-based two-stage methods popularized by R-CNN [8] and its descendants [7, 32, 16]. These methods perform inference by first proposing object regions and then classifying each region. Alternatively, single-shot approaches

[28, 31, 4, 26] perform inference in a single pass by generating predictions at fixed anchor positions. Our work builds on top of the SSD framework [28] due to its efficiency and competitive accuracy.

2.2. Video Object Detection with Tracks

Many existing methods on the Imagenet VID dataset organize single-frame detections into tracks or tubelets and update detections using postprocessing. Seq-NMS [10] links high-confidence predictions into sequences across the entire video. TCNN [22, 23] uses optical flow to map detections to neighboring frames and suppresses low-confidence predictions while also incorporating tracking algorithms. However, neither of these methods perform online inference, nor do they focus on efficiency.

Instead of operating on final detection results, our approach directly incorporates temporal context on the feature level and requires no postprocessing steps or online learning. Since our method can be extended by applying any of these approaches to our detection results, our network is more comparable to the base single-frame detector used by these methods than the methods themselves.

A recent work, D&T [6], combines tracking and detection by adding an RoI tracking operation and multi-task losses on frame pairs. However, their experiments focus on expensive high-accuracy models, whereas our approach is tailored for mobile environments. Even with aggressive temporal striding, their model cannot be run on mobile devices due to the cost of their Resnet-101 base network, which requires $79\times$ more computation than our larger model and $450\times$ more than our smaller one.

2.3. Video Object Detection with Optical Flow

A different class of ideas which powered the winning entry of the 2017 Imagenet VID challenge [42] involves using optical flow to warp feature maps across neighboring frames. Deep Feature Flow (DFF) [43] accelerates detection by running the detector on sparse key frames and using optical flow to generate the remaining feature maps. Flow-guided Feature Aggregation [42] improves detection accuracy by warping and averaging features from nearby frames with adaptive weighting.

These ideas are closer to our approach since they also use temporal cues to directly modify network features and save computation. However, these methods require optical flow information, which is difficult to obtain both quickly and accurately. Even the smallest flow network considered by DFF is approximately twice as computationally expensive as our larger model and $10\times$ more expensive than our smaller one. Additionally, the need to run both the flow network and the very expensive key frame detector in tandem makes this approach require far more memory and storage consumption than ours, which poses a problem on mobile

devices. In contrast, our network contains fewer parameters than our already-efficient baseline detector and has no additional memory overhead.

2.4. LSTMs for Video Analysis

LSTMs [15] have been successfully applied to many tasks involving sequential data [27]. One specific variant, the convolutional LSTM [34, 30], uses a 3D hidden state and performs gate computations using convolutional layers, allowing the LSTM to encode both spatial and temporal information. Our work modifies the convolutional LSTM to be more efficient and uses it to propagate temporal information across frames.

Other works have also used LSTMs for video-based tasks. LRCNs [5] use convolutional networks to extract features from each frame and feed these features as a sequence into an LSTM network. ROLO [29] performs object tracking by first running the YOLO detector [31] on each frame, then feeding the output bounding boxes and final convolutional features into an LSTM network. While these methods apply LSTMs as postprocessing on top of network outputs, our method fully integrates LSTMs into the base convolutional network via direct feature map refinement.

2.5. Efficient Neural Networks

Finally, several techniques for creating more efficient neural network models have previously been proposed, such as quantization [39, 41], factorization [20, 25], and Deep Compression [9]. Another option is creating efficient architectures such as Squeezenet [19], Mobilenet [17], Xception [3], and Shufflenet[40]. In the video domain, NoScope [21] uses a modified form of distillation [14] to train an extremely lightweight specialized model at the cost of generalization to other videos. Our work aims to creating a highly efficient architecture in the video domain where the aforementioned methods are also applicable.

3. Approach

In this section, we describe our approach for efficient online object detection in videos. At the core of our work, we propose a method to incorporate convolutional LSTMs into single-image detection frameworks as a means of propagating frame-level information across time. However, a naive integration of LSTMs incurs significant computational expenses and prevents the network from running in real-time. To address this problem, we introduce a Bottleneck-LSTM with depthwise separable convolutions [17, 3] and bottleneck design principles to reduce computational costs. Our final model outperforms analogous single-frame detectors in accuracy, speed, and model size.
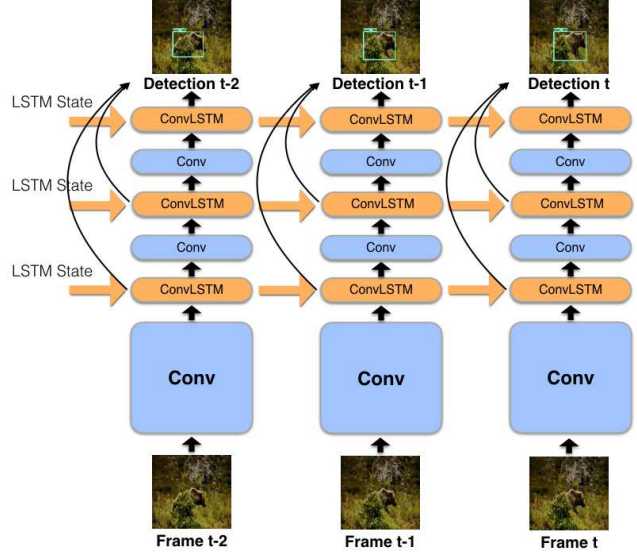

Figure 2: An example illustration of our joint LSTM-SSD model. Multiple Convolutional LSTM layers are inserted in the network. Each propagates and refines feature maps at a certain scale.

3.1. Integrating Convolutional LSTMs with SSD

Consider a video as a sequence of image frames $\mathcal{V} = \{I_0, I_1, \ldots I_n\}$. Our goal is to recover frame-level detections $\{D_0, D_1, \ldots D_n\}$, where each D_k is a list of bounding box locations and class predictions corresponding to image I_k. Note that we consider an online setting where detections D_k are predicted using only frames up to I_k.

Our prediction model can be viewed as a function $\mathcal{F}(I_t, \boldsymbol{s_{t-1}}) = (D_t, \boldsymbol{s_t})$, where $\boldsymbol{s_k} = \{s_k^0, s_k^1, \ldots s_k^{m-1}\}$ is defined as a vector of feature maps describing the video up to frame k. We can use a neural network with m LSTM layers to approximate this function, where each feature map of $\boldsymbol{s_{t-1}}$ is used as the state input to one LSTM and each feature map of $\boldsymbol{s_t}$ is retrieved from the LSTM's state output. To obtain detections across the entire video, we simply run each image through the network in sequence.

To construct our model, we first adopt an SSD framework based on the Mobilenet architecture and replace all convolutional layers in the SSD feature layers with depthwise separable convolutions. We also prune the Mobilenet base network by removing the final layer. Instead of having separate detection and LSTM networks, we then inject convolutional LSTM layers directly into our single-frame detector. Convolutional LSTMs allow the network to encode both spatial and temporal information, creating a unified model for processing temporal streams of images.

3.2. Feature Refinement with LSTMs

When applied to a video sequence, we can interpret our LSTM states as features representing temporal context. The LSTM can then use temporal context to refine its input at each time step, while also extracting additional temporal cues from the input and updating its state. This mode of refinement is general enough to be applied on any intermediate feature map by placing an LSTM layer immediately after it. The feature map is used as input to the LSTM, and the LSTM's output replaces the feature map in all future computations.

Let us first define our single-frame detector as a function $\boldsymbol{G}(I_t) = D_t$. This function will be used to construct a composite network with m LSTM layers. We can view placing these LSTMs as partitioning the layers of $\boldsymbol{G}$ into $m + 1$ sub-networks $\{g_0, g_1, \ldots g_m\}$ satisfying

$$\boldsymbol{G}(I_t) = (g_m \circ \cdots \circ g_1 \circ g_0)(I_t). \tag{1}$$

We also define each LSTM layer $\mathcal{L}_0, \mathcal{L}_1, \ldots \mathcal{L}_{m-1}$ as a function $\mathcal{L}_k(M, s_{t-1}^k) = (M_+, s_t^k)$, where M and M_+ are feature maps of the same dimension. Now, by sequentially computing

$$(M_+^0, s_t^0) = \mathcal{L}_0(g_0(I_t), s_{t-1}^0)$$
$$(M_+^1, s_t^1) = \mathcal{L}_1(g_1(M_+^0), s_{t-1}^1)$$
$$\vdots$$
$$(M_+^{m-1}, s_t^{m-1}) = \mathcal{L}_{m-1}(g_{m-1}(M_+^{m-2}), s_{t-1}^{m-1})$$
$$D_t = g_m(M_+^{m-1}),$$

we have formed the function $\mathcal{F}(I_t, \boldsymbol{s_{t-1}}) = (D_t, \boldsymbol{s_t})$ representing our joint LSTM-SSD network. Figure 2 depicts the inputs and outputs of our full model as it processes a video. In practice, the inputs and outputs of our LSTM layers can have different dimensions, but the same computation can be performed as long as the first convolutional layer of each sub-network $\mathcal{F}$ has its input dimensions modified.

In our architecture, we choose the partitions of $\boldsymbol{G}$ experimentally. Note that placing the LSTM earlier results in larger input volumes, and the computational cost quickly becomes prohibitive. To make the added computation feasible, we consider LSTM placements only after feature maps with the lowest spatial dimensions, which is limited to the Conv13 layer (see Table 1) and SSD feature maps. In section 4.2, we empirically show that placing LSTMs after the Conv13 layer is most effective. Within these constraints, we consider several ways to place LSTM layers:

1. Place a single LSTM after the Conv13 layer.
2. Stack multiple LSTMs after the Conv13 layer.
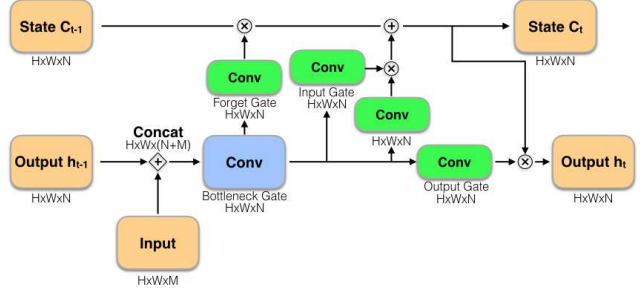3. Place one LSTM after each feature map

Figure 3: Illustration of our Bottleneck-LSTM. Note that after the bottleneck gate, all layers have only N channels.

3.3. Extended Width Multiplier

LSTMs are inherently expensive due to the need to compute several gates in a single forward pass, which presents a problem in efficiency-focused networks. To address this, we introduce a host of changes that make LSTMs compatible with the goal of real-time mobile object detection.

First, we address the dimensionality of the LSTM. We can obtain finer control over the network architecture by extending the channel width multiplier α defined in [17]. The original width multiplier is a hyperparameter used to scale the channel dimension of each layer. Instead of applying this multiplier uniformly to all layers, we introduce three new parameters $\alpha_{base}, \alpha_{ssd}, \alpha_{lstm}$, which control the channel dimensions of different parts of the network.

Any given layer in the base Mobilenet network with N output channels is modified to have $N\alpha_{base}$ output channels, while α_{ssd} applies to all SSD feature maps and α_{lstm} applies to LSTM layers. For our network, we set $\alpha_{base} = \alpha$, $\alpha_{ssd} = 0.5\alpha$, and $\alpha_{lstm} = 0.25\alpha$. The output of each LSTM is one-fourth the size of the input, which drastically cuts down on the computation required.

3.4. Efficient Bottleneck-LSTM Layers

We are also interested in making the LSTM itself more efficient. Let M and N be the number of input and output channels in the LSTM respectively. Since the definition of convolutional LSTMs varies among different works [34, 30], we will define a standard convolutional LSTM as:

$$f_t = \sigma\left(^{(M+N)}W_f^N \star [x_t, h_{t-1}]\right)$$
$$i_t = \sigma\left(^{(M+N)}W_i^N \star [x_t, h_{t-1}]\right)$$
$$o_t = \sigma\left(^{(M+N)}W_o^N \star [x_t, h_{t-1}]\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \phi\left(^{(M+N)}W_c^N \star [x_t, h_{t-1}]\right)$$
$$h_t = o_t \circ \phi(c_t).$$

This LSTM takes 3D feature maps x_t and h_{t-1} as inputs and concatenates them channel-wise. It outputs a feature map h_t and cell state c_t. Additionally, $^jW^k \star X$ denotes

Layer	Filter Size	Stride
Conv1	$3 \times 3 \times 3 \times 32$	2
Conv2	$3 \times 3 \times 32$ dw	1
	$1 \times 1 \times 32 \times 64$	1
Conv3	$3 \times 3 \times 64$ dw	2
	$1 \times 1 \times 64 \times 128$	1
Conv4	$3 \times 3 \times 128$ dw	1
	$1 \times 1 \times 128 \times 128$	1
Conv5	$3 \times 3 \times 128$ dw	2
	$1 \times 1 \times 128 \times 256$	1
Conv6	$3 \times 3 \times 256$ dw	1
	$1 \times 1 \times 256 \times 256$	1
Conv7	$3 \times 3 \times 256$ dw	2
	$1 \times 1 \times 256 \times 512$	1
Conv8-12	$3 \times 3 \times 512$ dw	1
	$1 \times 1 \times 512 \times 512$	1
Conv13	$3 \times 3 \times 512$ dw	2
	$1 \times 1 \times 512 \times 1024$	1
Bottleneck-LSTM	$3 \times 3 \times 1024$ dw	1
	$1 \times 1 \times (1024 + 256) \times 256$	1
	$3 \times 3 \times 256$ dw	1
	$1 \times 1 \times 256 \times 1024$	1
Feature Map 1	$1 \times 1 \times 256 \times 128$	1
	$3 \times 3 \times 128$ dw	2
	$1 \times 1 \times 128 \times 256$	1
Feature Map 2	$1 \times 1 \times 256 \times 64$	1
	$3 \times 3 \times 64$ dw	2
	$1 \times 1 \times 64 \times 128$	1
Feature Map 3	$1 \times 1 \times 128 \times 64$	1
	$3 \times 3 \times 64$ dw	2
	$1 \times 1 \times 64 \times 128$	1
Feature Map 4	$1 \times 1 \times 128 \times 32$	1
	$3 \times 3 \times 32$ dw	2
	$1 \times 1 \times 32 \times 64$	1

Table 1: Convolutional layers in one of our LSTM-SSD architectures using a single LSTM with a 256-channel state. "dw" denotes a depthwise convolution. Final bounding boxes are obtained by applying an additional convolution to the Bottleneck-LSTM and Feature Map layers. Note that we combine all four LSTM gate computations into a single convolution, so the LSTM computes 1024 channels of gates but outputs only 256 channels.

a depthwise separable convolution with weights W, input X, j input channels, and k output channels, ϕ denotes the activation function, and $\circ$ denotes the Hadamard product.

Our use of depthwise separable convolutions immediately reduces the required computation by 8 to 9 times compared to previous definitions. We also choose a slightly unusual $\phi(x) = ReLU(x)$. Though ReLU activations are not commonly used in LSTMs, we find it important not to alter the bounds of the feature maps since our LSTMs are interspersed among convolutional layers.

We also wish to leverage the fact that our LSTM has substantially fewer output channels than input channels. We introduce a minor modification to the LSTM equations by first computing a bottleneck feature map with N channels:

$$b_t = \phi(^{(M+N)}W_b^N \star [x_t, h_{t-1}]). \qquad (2)$$

Then, b_t replaces the inputs in all other gates as shown in Figure 3. We refer to this new formulation as the Bottleneck-LSTM. The benefits of this modification are twofold. Using the bottleneck feature map decreases computation in the gates, outperforming standard LSTMs in

all practical scenarios. Secondly, the Bottleneck-LSTM is deeper than the standard LSTM, which follows empirical evidence [12, 13] that deeper models outperform wide and shallow models.

Let the spatial dimensions of each feature map be $D_F \times D_F$, and let the dimensions of each depthwise convolutional kernel be $D_K \times D_K$. Then, the computational cost of a standard LSTM is:

$$4(D_K^2 \cdot (M + N) \cdot D_F^2 + (M + N) \cdot N \cdot D_F^2). \qquad (3)$$

The cost of a standard GRU is nearly identical, except with a leading coefficient of 3 instead of 4. Meanwhile, the cost of a Bottleneck-LSTM is:

$$\begin{aligned} D_K^2 \cdot (M + N) \cdot D_F^2 + (M + N) \cdot N \cdot D_F^2 \\ + 4(D_K^2 \cdot N \cdot D_F^2 + N^2 \cdot D_F^2). \end{aligned} \qquad (4)$$

Now, set $D_K = 3$ and let $k = \frac{M}{N}$. Then, Equation (3) is greater than Equation (4) when $k > \frac{1}{3}$. That is, as long as our LSTM's output has less than three times as many channels as the input, it is more efficient than a standard LSTM. Since it would be extremely unusual for this condition to not hold, we claim that the Bottleneck-LSTM is more efficient in all practical situations. The Bottleneck-LSTM is also more efficient than a GRU when $k > 1$. In our network, $k = 4$, and the Bottleneck-LSTM is substantially more efficient than any alternatives. One of our complete architectures is detailed in Table 1.

4. Experiments

4.1. Experiment Setup

We train and evaluate on the Imagenet VID 2015 dataset. For training, we use all 3,862 videos in the Imagenet VID training set. We unroll the LSTM to 10 steps and train on sequences of 10 frames. We train our model in Tensorflow [1] using RMSprop [38] and asynchronous gradient descent. Like the original Mobilenet, our model can be customized to meet specific computational budgets. We present results for models with width multiplier $\alpha = 1$ and $\alpha = 0.5$. For the $\alpha = 1$ model, we use an input resolution of 320×320 and a learning rate of 0.003. For the $\alpha = 0.5$ model, we use 256×256 input resolution and a learning rate of 0.002.

We include hard negative mining and data augmentation as described in [28]. We adjust the original hard negative mining approach by allowing a ratio of 10 negative examples for each positive while scaling each negative loss by 0.3. We obtain significantly better accuracy with this modification, potentially because the original approach harshly penalizes false negatives in the groundtruth labels.

To deal with overfitting, we train the network using a two-stage process. First, we finetune the SSD network without LSTMs. Then, we freeze the weights in the base

Placed After	mAP
No LSTM (baseline)	50.3
Conv3	49.1
Conv13	**53.5**
Feature Map 1	51.0
Feature Map 2	50.5
Feature Map 3	50.8
Feature Map 4	51.0
Outputs	51.2

Table 2: Performance of our model ($\alpha = 1$) with a single LSTM placed after different layers. Layer names match Table 1. For the outputs experiment, we place LSTMs after all five final prediction layers.

network, up to and including layer Conv13, and inject the LSTM layers for the remainder of the training.

For evaluation, we randomly select a segment of 20 consecutive frames from each video in the Imagenet VID evaluation set for a total of 11080 frames. We designate these frames as the minival set. For all results, we report the standard Imagenet VID accuracy metric, mean average precision @0.5 IOU. We also report the number of parameters and multiply-adds (MAC) as benchmarks for efficiency.

4.2. Ablation Study

In this section, we demonstrate the individual effectiveness of each of our major design decisions.

Single LSTM Placement   First, we place a single LSTM after various layers in our model. Table 2 confirms that placing the LSTM after feature maps results in superior performance with the Conv13 layer providing the greatest improvement, validating our claim that adding temporal awareness in the feature space is beneficial.

Recurrent Layer Type   Next, we compare our proposed Bottleneck-LSTM with other recurrent layer types including averaging, LSTMs, and GRUs [2]. For this experiment, we place a single recurrent layer after the Conv13 layer and evaluate our model for both $\alpha = 1$ and $\alpha = 0.5$. As a baseline, we use weighted averaging of features in consecutive frames, with a weight of 0.75 for the current frame and 0.25 for the previous frame. The Bottleneck-LSTM's output channel dimension is reduced according to the extended width multiplier, but all other recurrent layer types have the same input and output dimensions since they are not designed for bottlenecking. Results are shown in Table 3. Our Bottleneck-LSTM is an order of magnitude more efficient than other recurrent layers while attaining comparable performance.

α	Type	mAP	Params (M)	MAC (M)
0.5	Averaging	37.6	–	–
	LSTM	43.3	2.11	135
	GRU	**44.5**	1.59	102
	Bottleneck-LSTM	42.8	**0.15**	**5.6**
1.0	Averaging	50.7	–	–
	LSTM	53.5	8.41	840
	GRU	**54.0**	6.33	632
	Bottleneck-LSTM	53.5	**0.60**	**34**

Table 3: Performance of different recurrent layer types. The parameters and multiply-adds columns are computed for the recurrent layer only.
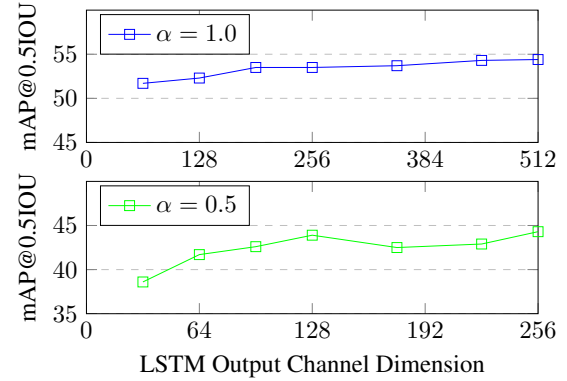

Figure 4: Model Performance vs. LSTM Output Channels. This figure shows model performance as a function of the LSTM's output channel dimension.

Bottleneck Dimension   We further analyze the effect of the LSTM output channel dimension on accuracy, shown in Figure 4. A single Bottleneck-LSTM is placed after the Conv13 layer in each experiment. Accuracy remains near-constant up to $\alpha_{lstm} = 0.25\alpha$, then drops off. This supports our use of the extended width multiplier.

Multiple LSTM Placement Strategies   Our framework naturally generalizes to multiple LSTMs. In SSD, each feature map represents features at a certain scale. We investigate the benefit of incorporating multiple LSTMs to refine feature maps at different scales. In Table 4, we evaluate different strategies for incorporating multiple LSTMs. In this experiment, we incrementally add more LSTM layers to the network. Due to difficulties in training multiple LSTMs simultaneously, we finetune from previous checkpoints while progressively adding layers. Placing LSTMs after feature maps at different scales results in a slight performance improvement and nearly no change in computational cost due to the small dimensions of later feature maps and the efficiency of our LSTM layers. However, stacking two LSTMs

Place Bottleneck-LSTMs After					mAP		Params (M)		MAC (B)	
Conv13	FM1	FM 2	FM 3	FM 4	$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 0.5$	$\alpha = 1.0$
✓					42.8	53.5	**0.81**	**3.01**	**0.19**	**1.12**
✓✓					42.9	53.3	0.91	3.41	0.20	1.16
✓	✓				43.3	53.7	0.82	3.07	0.19	1.13
✓	✓	✓			43.5	54.0	0.82	3.11	0.19	1.13
✓	✓	✓	✓		**43.8**	**54.4**	0.85	3.21	0.19	1.13
✓	✓	✓	✓	✓	**43.8**	**54.4**	0.86	3.24	0.19	1.13

Table 4: Performance of our architecture with multiple Bottleneck-LSTMs. Layer names match Table 1, where FM stands for Feature Map. ✓denotes a single LSTM placed after the specified layer, while ✓✓denotes two LSTMs placed.

Model	mAP	Params (M)	MAC (B)
Resnet-101 Faster-RCNN	62.5	63.2	143.37
Inception-SSD	56.3	13.7	3.79
Mobilenet-SSD ($\alpha = 1$)	50.3	4.20	1.20
Ours ($\alpha = 1$)	**54.4**	**3.24**	**1.13**
Mobilenet-SSD ($\alpha = 0.5$)	40.0	1.17	0.20
Ours ($\alpha = 0.5$)	**43.8**	**0.86**	**0.19**

Table 5: Final results on our Imagenet VID minival set.

Model	mAP		
	$p = 0.25$	$p = 0.50$	$p = 0.75$
Inception-SSD	43.3	34.6	25.9
Mobilenet-SSD ($\alpha = 1$)	43.0	33.8	24.6
Ours ($\alpha = 1$)	**49.1**	**42.4**	**33.3**
Mobilenet-SSD ($\alpha = 0.5$)	33.2	26.0	19.3
Ours ($\alpha = 0.5$)	**39.8**	**33.9**	**25.6**

Table 6: Model performance after randomly occluding a fraction p of the bounding boxes.

Model	big core (ms)	LITTLE core (ms)
Inception-SSD	1020	2170
Mobilenet-SSD ($\alpha = 1$)	357	794
Ours ($\alpha = 1$)	**322**	**665**
Mobilenet-SSD ($\alpha = 0.5$)	72	157
Ours ($\alpha = 0.5$)	**65**	**140**

Table 7: Runtime comparison of single-frame detection models and our LSTM-SSD model on both the big and LITTLE core of a Qualcomm Snapdragon 835 on a Pixel 2.

after the same feature map is not beneficial. For the last two feature maps (FM3, FM4), we do not further bottleneck the LSTM output channels because the channel dimension is already very small. We use the model with LSTMs placed after all feature maps as our final model.

4.3. Comparison With Other Architectures

In Table 5, we compare our final model against state-of-the-art single-image detection frameworks. All baselines are trained with the open source Tensorflow Object Detection API [18]. Among these methods, only the Mobilenet-SSD variants and our method can run in real-time on mobile devices, and our method outperforms Mobilenet-SSD on all metrics. Qualitative differences are shown in Figure 5. We also include performance-focused architectures which are much more computationally expensive. Our method approaches the accuracy of the Inception-SSD network at a fraction of the cost. Additionally, we include the parameters and MAC of a Resnet Faster-RCNN to highlight the vast computational difference between our method and non-mobile video object detection methods, which generally include similarly expensive base networks.

4.4. Robustness Evaluation

We test the robustness of our method to input noise by creating artificial occlusions in each video. We generate these occlusions as follows: for each groundtruth bounding box, we assign a probability p of occluding the bounding box. For each occluded bounding box of dimension $H \times W$,

we zero out all pixels in a randomly selected rectangular region of size between $\frac{H}{2} \times \frac{W}{2}$ and $\frac{3H}{4} \times \frac{3W}{4}$ within that bounding box. Results for this experiment are reported in Table 6. All methods are evaluated on the same occlusions, and no method is trained on these occlusions prior to testing. Our method outperforms all single-frame SSD methods on this noisy data, demonstrating that our network has learned the temporally continuous nature of videos and uses temporal cues to achieve robustness against noise.

4.5. Mobile Runtime

We evaluate our models against baselines on the latest Pixel 2 phone with Qualcomm Snapdragon 835. Runtime is measured on both Snapdragon 835 big and LITTLE core, with single-threaded inference using a custom on-device implementation of Tensorflow [1]. Table 7 shows that our model outperforms all baselines. Notably, our $\alpha = 0.5$ model achieves real-time speed of 15 FPS on the big core.

Figure 5: Example clips in Imagenet VID minival set where our model ($\alpha = 1$) outperforms an analogous single-frame Mobilenet-SSD model ($\alpha = 1$). Our network uses temporal context to provide significantly more stable detections across frames. The upper row of each sequence corresponds to our model, and the lower row corresponds to Mobilenet-SSD.

5. Conclusion

We introduce a novel framework for mobile objection detection in videos based on unifying mobile SSD frameworks and recurrent networks into a single temporally-aware architecture. We propose an array of modifications that allow our model to be faster and more lightweight than mobile-focused single-frame models despite having a more complex architecture. We proceed to examine each of our design decisions individually, and empirically show that our modifications allow our network to be more efficient with minimal decrease in performance. We also demonstrate that our network is sufficiently fast to run in real-time on mobile devices. Finally, we show that our method outperforms comparable state-of-the-art single-frame models to indicate that our network benefits from temporal cues in videos.

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. 5, 7

[2] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 6

[3] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016. 3

[4] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016. 1, 2

[5] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015. 3

[6] C. Feichtenhofer, A. Pinz, and A. Zisserman. Detect to track and track to detect. In *ICCV*, 2017. 2

[7] R. Girshick. Fast r-cnn. In *ICCV*, 2015. 2

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 2

[9] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016. 3

[10] W. Han, P. Khorrami, T. L. Paine, P. Ramachandran, M. Babaeizadeh, H. Shi, J. Li, S. Yan, and T. S. Huang. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465*, 2016. 2

[11] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014. 1

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 5

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 5

[14] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 3

[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 3

[16] S. Hong, B. Roh, K.-H. Kim, Y. Cheon, and M. Park. Pvanet: Lightweight deep neural networks for real-time object detection. *arXiv preprint arXiv:1611.08588*, 2016. 2

[17] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 3, 4

[18] J. Huang, V. Rathod, D. Chow, C. Sun, and M. Zhu. Tensorflow object detection api, 2017. 7

[19] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 3

[20] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 3

[21] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. Noscope: Optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529*, 2017. 3

[22] K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang, and W. Ouyang. T-cnn: Tubelets with convolutional neural networks for object detection from videos. *arXiv preprint arXiv:1604.02532*, 2016. 2

[23] K. Kang, W. Ouyang, H. Li, and X. Wang. Object detection from video tubelets with convolutional neural networks. In *CVPR*, 2016. 2

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[25] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 3

[26] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal loss for dense object detection. In *ICCV*, 2017. 2

[27] Z. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015. 3

[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 1, 2, 5

[29] G. Ning, Z. Zhang, C. Huang, and Z. He. Spatially supervised recurrent convolutional neural networks for visual object tracking. *arXiv preprint arXiv:1607.05781*, 2016. 3

[30] V. Patraucean, A. Handa, and R. Cipolla. Spatio-temporal video autoencoder with differentiable memory. *arXiv preprint arXiv:1511.06309*, 2015. 3, 4

[31] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 2, 3

[32] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 2

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 2

[34] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015. 3, 4

[35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1

[37] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 1

[38] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012. 5

[39] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016. 3

[40] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017. 1, 3

[41] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *ICLR*, 2017. 3

[42] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei. Flow-guided feature aggregation for video object detection. In *ICCV*, 2017. 2

[43] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei. Deep feature flow for video recognition. In *CVPR*, 2017. 1, 2