Recurrent Neural Networks for object detection

Bin Qassim Ahmad
Technical University of Munich
Department of Informatics
Munich, Germany
ahmad.qasim@tum.de

Pettirsch Arnd
Technical University of Munich
Department of Informatics
Munich, Germany
a.pettirsch@outlook.de

***Abstract*—ToDo**
***Index Terms*—TBD.**

I. Introduction

A. Image and Video Object Detection in general

- Image object detection history.
  - Bayesian methods before deep learning
  - ImageNet challenge and VID [15]
  - Deep Learning and AlexNet [16]
- Single stage and 2-stage image object detectors.
  - A two-stage pipeline firstly generates region proposals, which are then classified and refined. [17]
  - A single-stage method is often more efficient but less accurate. Directly regress on bounding boxes and classes. [18], [19]
- Why is video object detection harder?
  - Large size
  - Motion blur
  - Quality of the dataset
  - Partial occlusion
  - Unconventional Poses

B. Recurrent Neural Networks in general

1) Delimitation to non-recurrent Neural Networks: Non-recurrent Neural Networks process on single inputs. In the field of image classification this could be for example a single image.
Recurrent Neural Networks process on sequences of date. In the field of classification those sequences consist often multiple frames. Recurrent Neural Network's core concept to enable the sequence processing is parameter sharing across different parts of a model. Parameter sharing can be reached by cycles in the architecture. [11]

2) Common Types of Recurent Neural Networks: Most of the paper which are described in this work use two pretty common approaches of Recurrent Neural Networks. The first one are LSTMs, first mentioned in [18] in 1997. The key of LSTMs is the inner state unit. LSTMs consist of four layers. Those four layers are able to remove or add information to the inner cell state.

The second type of Recurrent Neural Networks is Gated Recurrent Units (GRUs). The main difference to LSTMs is, that those GRUs consist out of a single gated unit which can simultaneously control the forgetting factor and decide to update the inner cell state. [11]

II. Feature-based Video Object Detection

A. Definition

First, we want to introduce feature-based Video Object Detection methods. As defined, for example, in [1] feature-based Video Object Detection methods fuse detectors which integrate features from multiple frames into their video detection. In most of our papers those detectors use recurrent units to integrate the visual features from different frames.

B. Recurrent Multi-frame Single Shot Detector for Video Object Detection

Broad, Jones and Lee have in [1] and [12] the idea to insert a fusion-layer into a classical Single Shot Detector (SSD) meta-architecture. Based on this idea they research in two main fields: On the one hand, they investigate different fusion methods and on the other hand they try several SSD meta-architectures. The final architecture is shown in Fig. 1.

SSDs is generally consist of two main components. First, an so called Feature Extractor which consist of convolutional and pooling layers. As an input the feature extractor gets an image and it outputs feature-maps. The other component is an Detection head which consists of convolutional layers and creates bounding boxes and class probabilities out of the feature maps created by the feature extractor.

As mentioned above Broad, Jones and Lee have the idea to insert a fusion layer in between those two components. As fusion techniques they test simple element-wise operations (e.g.: add or max), simple concatenations of features maps and recurrent layers. Their experiments show that the recurrent layers perform best. Element-wise operations do not lead to an improvement in the mAP. And concatenations just add another depth to the network, which increases the mAP slightly.Only the Recurrent Units add new parameters, to learn temporal context, to the Network. In addition, they observe that recurrent units do not slow down the computational speed significantly. The state of the art SSD, which is used

to test the different fusion techniques performs on 55 fps the recurrent one on 50 fps. As a recurrent unit they use GRU because they observ that the results were similar to the results when they use LSTMs but the GRUs perform faster.

In addition to the investigation on the type of fusion layer, Broad, Jones and Lee test different types of SSDs as a baseline for their architecture. They find out that for all baseline SSDs the mAP was higher in comparison to the non-recurrent models. The mAP increase by 2.7 to 5 percent on KITTI dataset. The best results is made with SequeezeNet+ as a baseline SSD network.

Unfortunately, there is only a little information on the training of the Recurrent Multi-frame Single Shot Detector. They use the SequeezeNet Training strategies and a pre-trained version of the baseline SSD. Finally, they use the SequezzeNet fine tuning strategy to train the whole network afterwards.

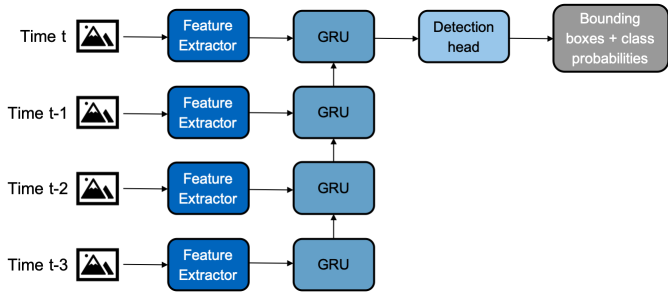


Fig. 1: Architecture Recurent Multi-frame single shot detector

C. Mobile Video Object Detection with Temporally Aware Feature Maps

Liu and Zhu have the goal to design a video object detection architecture, which can run real-time on low-powered mobile and embedded devices. The key of their method is to combine convolutional layers with convolutional LSTMs. On this core idea they do some further research. They test the benefit from adding an LSTM into the baseline SSD, different types of Recurrent Layers (LSTM, GRU and bottleneck LSTMs), different dimensions of their bottleneck LSTMs and different LSTM placement strategies (Single LSTM placement and multiple LSTM placement). Their outcome is described in [2]

In the beginning they simply add one LSTM to their baseline SSDs architecture - MobileNet. They observe that adding the LSTM improves the mAP in comparison to their baseline SSD architecture. Moreover, they investigate that the greatest improve is by adding the LSTM after the 13th convolutional Layer of the SSD.

Afterwards, they compare LSTMs, GRUs and Bottleneck LSTMs as different types of Recurrent units by placing them after the 13th convolutional layer. Bottleneck-LSTMs have been designed by Liu and Zhu to increase the efficiency of LSTMs. For that purpose they use slightly unusual the ReLU function as activation for the LSTM. Moreover they, compute a so called Bottleneck feature map with less input channel than the originally feature map and feed this map into the LSTM to reduce computational power. They come to the conclusion that Bottleneck-LSTMs are more effective the LSTMs and in the case of a convolutional kernel greater 1x1 even more effective than GRUs while attaining comparable perfomance.

In addition to the bottleneck LSTMs Liu and Zhu extend their network with multipliers. They use α_{base}, α_{ssd} and α_{lstm} as multiplier to scale the channel dimension of each layer. During their research they find out that the accuracy of the model remains near constant up to $\alpha_{ssd} = 0.25\alpha$ which means that the output of each LSTM is one-fourth the size of the input. For the other multipliers they use the values: $\alpha_{base} = \alpha$ and $\alpha_{ssd} = 0.5\alpha$.

As shown in Fig. 2 the final model uses LSTMs after all feature maps, because Liu and Zhu observe that there a slight performance improve by adding LSTMs after every feature map and nearly no change in computational cost.

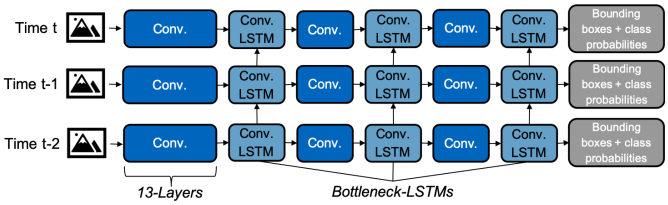


Fig. 2: Architecture Mobile Video Object Detection with temporally aware feature maps

On the training strategy and the loss function Liu and Zhu do not provide any information.

D. Feature Selective Small Object Detection via Knowledge-based recurrent attentive neural networks

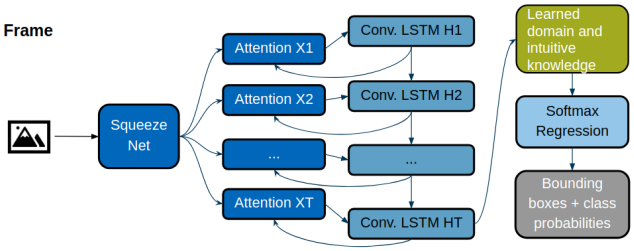


Fig. 3: Architecture Feature Selective Small Object Detection via Knowledge-based recurrent attentive neural networks

The aim of this paper is develop a real-time object detector for the purpose of autonomous driving. The network developed in this paper is termed as Knowledge-Based Recurrent Attentive Neural Network (KB-RANN) and the authors successfully compressed and accelerated their

proposed model and deployed it to their own self-developed autonomous car.

The authors use SqueezeNet for feature extractor because SqueezeNet can provide AlexNet level of accuracy but with 50 times less parameters. Although there are better performing models like VGGNet and ResNet, but these models are computationally expensive. They make a few changes in the SqueezeNet architecture, which include changing the kernel size and also fine-tuning the backend by adding two fire modules at the end.

Attention mechanism is used to find the saliency maps from the deep feature maps obtained from SqueezeNet. Attention module obtains the input tensor X and outputs the saliency map $\tilde{X}_t$.

LSTM is used as a memory mechanism in order to find long term dependencies between different frames and the characteristics of Attention and LSTM are exploited together by fusing them with each other, this fused module is termed as RANN. As saliency feature extraction is applied on the original deep feature maps, it is possible that some of the information is lost. Multiple RANNs are cascaded together into a chain to minimize the effect of feature information loss. Moreover, the output H_t of each LSTM is concatenated with the original feature map and used as an input for the next attention module to refine the saliency feature extraction process.

The authors also add a domain and intuitive knowledge module to the KB-RANN architecture. It is assumed that traffic signs detection is one of the most important aspect of autonomous driving. The major focus of attention of a driver is at the center of vision and the traffic lights are always located at a bias from that central region of peoples' gazes. With these assumptions in place, domain knowledge about traffic signals is learned from the data itself by constraining the distribution to a 2D Gaussian function and learning the mean and covariance matrices from the data.

E. Looking fast and slow: memory-guided mobile video object detection

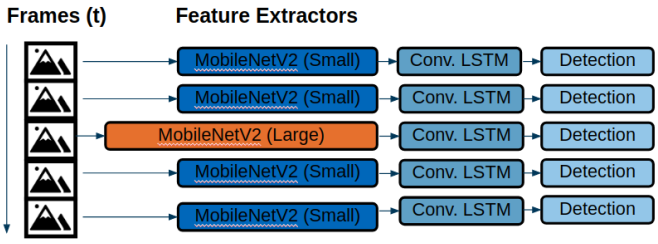


Fig. 4: Architecture Looking fast and slow: memory-guided mobile video object detection

The main contribution of this paper is the introduction of multiple feature extractors. Using multiple feature extractors, an interleaving policy can be defined which defines which feature extractor to use given the current state of detection. Using a intelligent interleaving policy and feature extractors with varying computational costs along with different feature representation capabilities, this paper aims to find a balance between detection accuracy and performance.

Let m being the number of feature extractors. For the implementation of this paper, the authors constraint $m = 2$, a d let f_1 be the feature extractor which is computationally more expensive while f_2 be the feature extractor which is cheaper. Both feature extractors use MobileNetV2 architecture. f_1 uses a depth of 1.4 with $320 * 320$ input frames resolution while f_2 uses a depth of 0.35 with a lower $160 * 160$ input frames resolution.

A modified LSTM is used as a memory mechanism to preserve long term dependencies. The modifications in the LSTM are responsible for making the memory mechanism faster and also better at preserving long term dependencies. For faster memory mechanism, the authors make three modifications. They introduce bottlenecking and add a skip connection between the bottleneck and the output. Lastly, the LSTM states are grouped together and convulations are applied on each group seperately and the resultant states are then concatenated together to obtain the final states. The grouped convulations provide a speed-up. In order to improve the preservation of long term dependencies by the LSTM, the LSTM states are only updated when f_1 is run and not the f_2, as feature maps obtained from f_2 are not of a higher quality as of f_1 and this can result in loss of important state information in the LSTM.

SSD-style detection is applied on refined feature maps obtained from the LSTM for classification and bounding boxes.

The interleaving policy which defines the feature extractor that should be used next is based on reinforcement learning. The state space of the reinforcement learning policy network π consists of the LSTM states, c_t, h_t, as well as the differences between the states at different timestamps i.e. $c_t - c_{t-1}$, $h_t - h_{t-1}$ and lastly the action history η_t. The action space has length m and the action a means that the feature extractor f_a should be run.

It is observed by the authors that, despite the employment of the interleaving policy the real-time detection is limited by the execution of the expensive f_1 feature extractor. They introduce an asynchronous framework for running the feature extractors f_1 and f_2 extractors in parallel. During testing, this asynchronous framework results in better results.

F. Detect to Track and track to detect

This paper is based on the R-FCN [19] model and extends it for multiple frame and tracking.

The R-FCN [19] detection process consists of two stages, firstly a Region Proposal Network (RPN) is used to find the candidate region of interests (ROIs) and then a ROI pooling layer is used to perform classification of the regions into classes or the background. The input to the ROI pooling layer

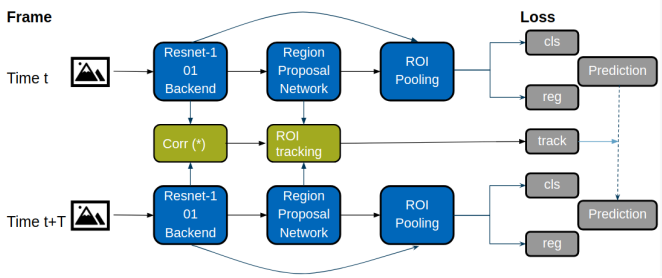


Fig. 5: Architecture Detect to Track and track to detect

comes from a convulational layer put in place at the end of the ResNet feature extractor with output x^t_{cls}. The ROI layer outputs position sensitive feature maps and using a softmax layer these position senstive feature maps can be converted to class probablities for each ROI. On a second branch, R-FCN places a convulational layer at the end of the ResNet feature extractor which outputs x^t_{reg} and this ouput is again used as an input to ROI pooling layer which generates the bounding boxes.

For each pair of frames I^t, $I^{t+\tau}$, a bounding box regression layer is introduced that performs position senstive ROI pooling on the concatentation of bounding box regression features x^t_{reg}, x^{t-1}_{reg}, which are also stacked with correlation maps, to perform bounding box transformation regression between frames. The correlation maps between the two frames are obtained by finding correlation between the feature maps of the two frames. Finding correlation on all the features in the feature maps will result in an explosion of dimensionality so the correlation maps are only limited to the local neighbors. Like mentioned before, the correlation maps are stacked with the bounding box features maps.

III. Box-Level-based Video Object Detection

A. Definition

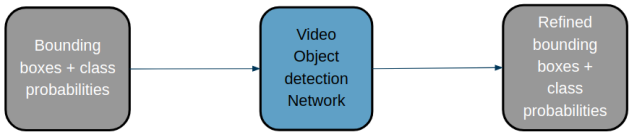


Fig. 6: Box-Level-based Video Object Detection

Bounding Boxes and Class probabilities are fed into the network and are refined temporally and/or spatially.

B. Optimizing Video Object Detection via Scale-Time Lattice

The primary contribution of this paper is the Scale-Time Lattice which allows coarse detection, both temporally and spatially and then use temporal propagation and spatial refinement to go from coarse to fine detection. The lattice is composed of structures which connect together to perform the temporal and spatial operations. These structures are called Propagation and Refinement Units (PRUs).

PRUs work on the basis of two operators F_τ and F_S, and by carefully allocating resources to the two operators, a balance between the detection accuracy and the performance can be acheived. Assuming that two frames are sampled from a video at time t and $t + 2\tau$, both at scale s, the F_τ operator tries to model the movement of the bounding boxes from time t to $t + \tau$ and from time $t + 2\tau$ to $t + \tau$ irrespective of the scale offset between the scale s and the ground truth frame scale $s + 1$. The modeling of the bounding boxes offset from scale s to $s + 1$ is the task of the F_S operator. The figure depicting the PRU gives more information on this.

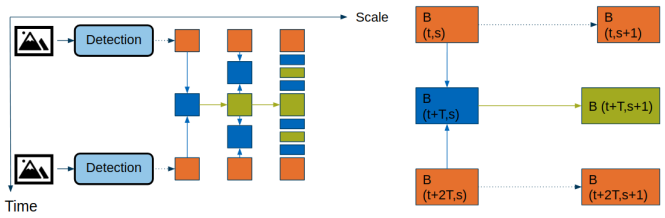


Fig. 7: Architecture Optimizing Video Object Detection via Scale-Time Lattice

C. Context Matters: Refining Object Detection in Video with Recurrent Neural Networks

In [4] Tripathi, Lipton, Belongie, Nguyen come up with a neural network architecture for video-based object detection. Their architecture consists two parts: A pseudo-labeler, which assigns labels to all video frames and a recurrent unit which refines those pseudo-labels by using the contextual information. Moreover, they describe a training strategy for their architecture and compare their approach to other models on the YouTube-Objects dataset.

The final architecture can be found in **Fig 2**. Tripathi, Lipton and Belongie first train the pseudo-labeler, which is an YOLO object detection network originally trained for 20-class PASCAL VOC on the YouTube-Video Dataset. As specified in YOLO [] they minimize the weighted squared detection loss and optimize classification and localization error simultaneously.

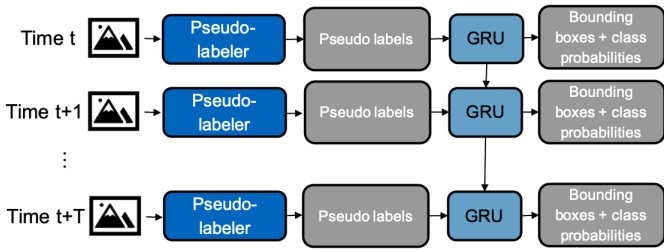


Fig. 8: Architecture Context Matters: Refining Object Detection in Video with Recurrent Neural Networks

After training the pseudo-labeler they train the Recurrent Neural Network, which takes as an input the pseudo-labels and outputs improved prediction. The RNN consists of two GRU layers.

For training the whole network they use the following loss function to take both accuracy at the target frame and consistency of predictions across adjacent time steps into consideration:

$$loss = d_{loss} + \alpha \cdot s_{loss} + \beta \cdot c_{loss} + \gamma \cdot pc_{loss}$$

For the final output Tripathi, Lipton, Belongie and Nguygen use the object detection loss as described in YOLO []:

ToDo

The similarity loss considers the dissimilarity between the pseudo-labels and prediction at each frame t:

ToDo

The category loss takes wrong class probabilities into consideration:

•

And the prediction-consistency loss regularizes the model by encouraging smoothness predictions across the time-steps:

•

During the evaluation they find two possible areas of improvement for their approach. On the one hand the RNN is not able to recover from wrong predictions made by the pseudo-labeler after they have been fed into the RNN. On the other hand, they observe that their network is not robust to motion.

D. Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking

Ning, Zhang, Huang, He, Ren and Wang design in [5] a combination of box-level and feature-level based Video detectors. They use the YOLO network to create high-level visual features and preliminary location inferences and feed both into a recurrent unit. The architecture, called ROLO, is shown in **figure**

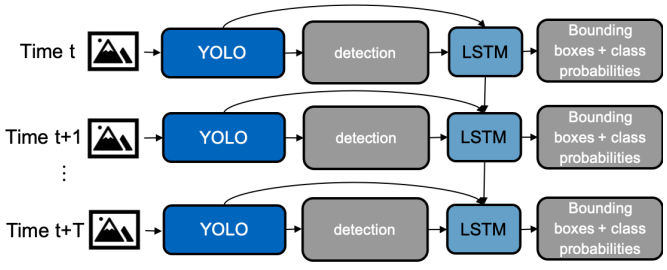


Fig. 9: Architecture ROLO

They use three phases to train their model: The pre-training phase of the convolutional layers for feature learning, the traditional YOLO training phase for object proposal and the LSTM training phase for object tracking.

In the pre-training phase the convolutional layers, which create the feature maps are training on ImageNet data. Afterwards the YOLO-architecture is adopted as detection module. At least they add LSTMs for tracking. The LSTMs are fed with, the feature representations from the convolutional layers, the Bounding boxes from the detection module and the output states from the LSTM in the previous time-step. For training they use the Mean Squared Error (MSE):

$$L_{MSE} = \frac{1}{n} TODO$$

As an alternative, Ning, Zhang, Huang, He, Ren and Wang mention a Heatmap as input for the LSTM. Therefor the prediction and the visual features are concatenated before adding them to the LSTM. This is helpful to visualize the intermediate results.

Special to this architecture is, that LSTM does regression in two folds. There is a regression within one frame between the location inferences and the high-level features and there is also regression over the different frames of the sequence.

IV. Flow-based Object Detection

A. Definition

Another type of architectures for Video Object Detection defined, for example in [], are architectures which use Flow-Networks to consider the temporal context. The flow network estimates the optical flow which means it projects back the location in the current frame to an earlier frame.

B. Deep Feature Flow for Video Recognition

Zhu, Xiong, Dai, Yuan and Wei develop in [2] a way to use flow nets to detect objects in video frames. As shown in figure 10 their architecture consists of three main parts: A network to create visual features, a network to create class probabilities and bounding boxes out of feature maps, and a network which estimates the optical flow.

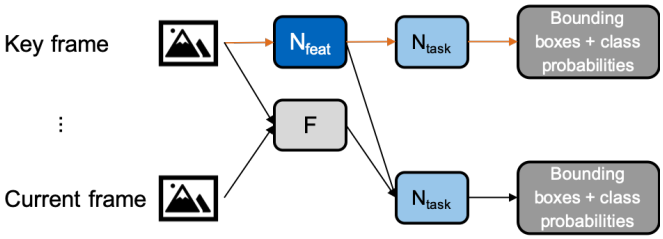


Fig. 10: Architecture Deep Feature Flow for Video Recognition

The paper differs between key frames and other frames. The network to create visual features N_{feat} only processes on key frames. It is a fully convolutional network, which takes as an input the image and outputs a feature maps. As a feature network they use a pretrained version of ResNet.

The second network N_{task} does the recognition task over the feature maps. It performs on every frame. Zhu, Xiong, Dai, Yuan and Wei use an R-FCN network for the recognition task.

For all non-key frames the feature maps are not constructed by the feature network. Instead, they are propagated by a flow network N_{flow}. Zhu, Xiong, Dai, Yuan and Wei use a state-of-the-art CNN based FlowNet architecture as a flow network. The flow estimation is given by:

$$f_i = W(f_k, M_{i->k}, S_{i->k})$$

f_k is the key frame's feature map, $M_{i->k}$ is a two dimensional flow field, it projects back the location of an object in the current frame to the location in the key frame by using bilinear interpolation and $S_{i->k}$ is scale field.

Zhu, Xiong, Dai, Yuan and Wei use a fixed key frame scheduling and they already see a potential improvement in changing the key frame policy.

V. Comparison of different approaches

A. General

TABLE I: Results on KITTI Dataset

Model	MAP	FPS	Machine	Architecture
Recurrent [1]	86.0	50	Nvidia TITAN X	Feature-Level
Feature Selective [6]	81.3	30.8	Nvidia TITAN X	Feature-Level

TABLE II: Results on ImageNet Dataset

Model	MAP	FPS	Machine	Architecture
DT [8]	82.0	7	Nvidia TITAN X	Feature-Level
DT [8]	78.5	55	Nvidia TITAN X	Feature-Level
Scale-Time Lattice [10]	79.6	20	Nvidia TITAN X	Box-Level
Scale-Time Lattice [10]	79	62	Nvidia TITAN X	Box-Level
DeepFeature Flow [3]	73.9	3	-	Flow-Based
DeepFeature Flow [3]	73.1	20.5	-	Flow-Based
Looking Fast and Slow [7]	60.7	48.8	Pixel Phone	Feature-Level
Object Detection with Temporally-Aware [2]	54.4	15	Pixel Phone	Feature-Level

TABLE III: Results on COCO Dataset

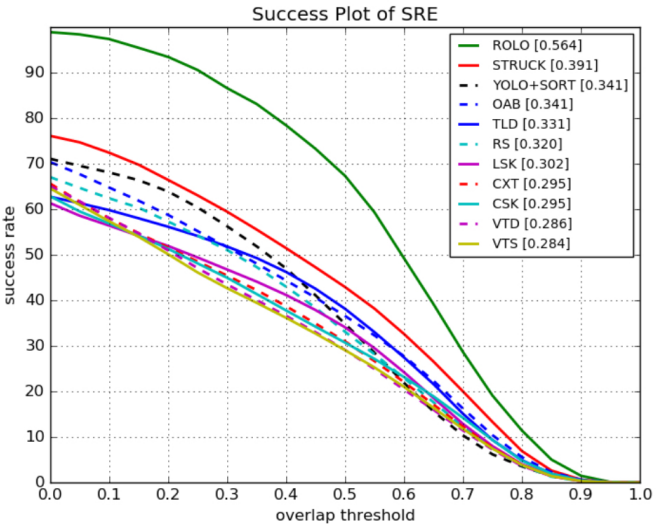
Model	MAP	FPS	Machine	Architecture
Feature Selective [6]	57.8	37.5	Nvidia TITAN X	Feature-Level

TABLE IV: Results on YT Dataset

Model	MAP	FPS	Machine	Architecture
Context Matters [4]	68.73	-	-	Box-Level

TABLE V: Results on OTB Challenge Dataset

Model	Success Rate	IoU	FPS	Machine
Spatially Supervised [5]	0.564	0.455	20/60	Nvidia TITAN X



B. Conclusion Perfomance

Todo

C. Conclusion Prediction Quality

Todo

VI. Outro

A. Conclusion

Todo

B. Further work

Todo

Acknowledgment

Todo

References

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be

cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] Alexander Broad, Michael Jones, Teng-Yok Lee. Recurrent Multi-frame Single Shot Detector for Video Object Detection. 2018.

[2] Mason Liu, Menglong Zhu. Mobile Video Object Detection with Temporally-Aware Feature Maps. 2018.

[3] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, Yichen Wei. Deep Feature Flow for Video Recognition. 2017.

[4] Subarna Tripathi, Zachary C. Lipton, Serge Belongie, Truong Nguyen. Context Matters: Refining Object Detection in Video with Recurrent Neural Networks.

[5] Guanghan Ning, Zhi Zhang, Chen Huang, Zhihai He, Xiaobo Ren, Haohong Wang. Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking. 2016.

[6] Kai Yi, Zhiqiang Jian, Shitao Chen, Nanning Zheng. Feature Selective Small Object Detection via Knowledge-based Recurrent Attentive Neural Network. 2019.

[7] Mason Liu, Menglong Zhu, Marie White, Yinxiao Li, Dmitry Kalenichenko. Looking Fast and Slow: Memory-Guided Mobile Video Object Detection. 2019.

[8] Christoph Feichtenhofer, Axel Pinz, Andrew Zisserman. Detect to Track and Track to Detect. 2017.

[9] Kai Kang, Wanli Ouyang, Hongsheng Li, Xiaogang Wang. Object Detection from Video Tubelets with Convolutional Neural Networks. 2016.

[10] Kai Chen, Jiaqi Wang, Shuo Yang, Xingcheng Zhang, Yuanjun Xiong, Chen Change Loy, Dahua Lin. Optimizing Video Object Detection via a Scale-Time Lattice. 2018.

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning (Adaptive Computation and Machine Learning). 2017.

[12] Alexander Broad, Michael Jones, Teng-Yok Lee. Supplementary Material for Recurrent Multi-frame Single Shot Detector for Video Object Detection. 2018.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database.

[14] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. 2012.

[15] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2016.

[16] Joseph Redmon, Ali Farhadi. YOLOv3: An Incremental Improvement.

[17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. 2016.

[18] Sepp Hochreiter, Jürgen Schmidhuber. Long short-term memory In: Neural Computation (journal), vol. 9, issue 8, p. 1735-1780. 1997.

[19] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: Object detection via region-based fully convolutional networks. In NIPS, 2016.