COMS4995 Deep Learning Project Final Report
Group 11
Detection in Videos with Modernized SSD

Jiajun Zhang (jz2979)
Columbia University
jz2979@columbia.edu

Shawn Pachgade (snp2128)
Columbia University
snp2128@columbia.edu

Aditya Huddedar (ah3426)
Columbia University
ah3426@columbia.edu

Abstract

We implement and extend the ideas presented in the Single Shot MultiBox Detector (SSD)[4] paper and apply it to real-time object detection in videos. In the past, there have been seminal developments in the field of object detection, specifically, You Only Look Once (YOLO)[6] and Faster R-CNN[7]. The value proposition of having an accurate, yet fast object detector includes autonomous vehicle driving, security camera surveillance, and an increase in the ability of artificial intelligence in general. By improving these systems, we can not only retrieve more quality and meaningful results, but also reduce the amount of resources dedicated to menial tasks, both with respect to time and money.

Our main goal is to implement and experiment with the SSD algorithm. Our ultimate goal is to create a video analysis demo combining the SSD architecture with Kernelized Correlation Filters (KCF) [3] to create a robust object tracking algorithm.

1. Introduction

The state-of-the-art object detection systems are highly accurate but too slow for real-time applications. There have been many attempts to improve the speed of the detectors, but so far, any improvement in the speed has come only at the expense of the detection accuracy.

SSD is the first single shot deep network based object detector that is significantly faster than previous state-of-the-art single shot detectors (YOLO). It is also significantly more accurate than the best of the slower techniques like R-CNN. KCF is the most efficient and economical approach to object tracking in videos.

With the advances in autonomous vehicle technology, robotics, etc., real-time object detection is of paramount importance now. It can be used to track rogue vehicles using the surveillance videos, settle an insurance dispute over road accidents, track the trajectory of an object, etc.

Our attempt in this project is to combine the two approaches of SSD and KCF to create a robust object detection and tracking algorithm.

2. Related Work and References

We have seen the development of object detection throughout the years, the most famous perhaps being YOLO and the in-the-wild performance it boasted. Other models include Fast and Faster R-CNN, which are also praised as part of the state-of-the-art for object detection.

These object detection systems are essentially variants of the following method: hypothesize bounding boxes, resample pixels or features for each box, and apply a high-quality classifier. The complexity of these models does not allow them to run fast enough on more economical hardware like embedded systems, especially the R-CNN series. On top of that, these models do not intelligently handle video, and purely rely on a forward operation on every time-step at test time.

SSD gets rid of the bounding box hypothesis by discretizing the output space of bounding boxes into a set of default bounding boxes over different aspect ratios and scales for each feature map location. In the forward pass, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. This elimination of bounding boxes proposal generation and subsequent feature resampling makes SSD significantly faster than the aforementioned approaches.

3. SSD Model

In this section, we present an overview of the different components of the SSD model. SSD stands for Single Shot Multibox Detector. Each of these phrases signifies a different aspect of the model.

- Single Shot: the object localization and detection are done using a single deep neural network forward pass

- Multibox: the SSD model uses a set of default boxes for a Multibox regression technique described in [8]
- Detection: the model also predicts the class of an object inside the box

3.1. Design

First few layers of the network are based on a standard high-quality image classification architecture, truncated before any of its classification layers. We refer to it as the Base Network.

We add several convolutional layers on top of the base network to predict the offsets to default boxes of different scales and aspect ratios along with the confidence for the presence of each object class instance in those boxes. These auxiliary convolutional layers decrease in size progressively. This allows both extracting features and predictions of detections at multiple scales.

Each feature map cell from multiple feature maps is associated with a set of default bounding boxes. These boxes tile the feature map in a convolutional manner, giving a high number of fixed candidate boxes. This helps in efficiently discretizing the space of possible output box shapes. The candidate boxes at each feature map cell are then regressed against the ground truth to learn the offsets and per-class scores.

3.2. Training

Initially, each ground truth box is matched to a default box with the best Jaccard overlap. Subsequently, default boxes are matched to any ground truth with Jaccard overlap higher than a threshold of 0.5. This allows the network to predict high scores for multiple overlapping default boxes rather than restricting the choice to only one with maximum overlap.

SSD's loss function is a weighted sum of the localization loss and confidence loss:

$$\text{ssd\_loss} = \text{confidence\_loss} + \alpha * \text{localization\_loss}$$

Confidence Loss measures how confident the network is of the classification of the object inside the computed bounding box. It is the Softmax loss over multiple classes confidence. Localization Loss measures how far away the network's predicted bounding boxes are from the ground truth ones from the training set. The authors use a Smooth L1 loss function here. The authors set α to 1 by cross-validation.

The paper combines predictions from all default boxes with different scales and aspect ratios from all locations of many feature maps, to generate a diverse set of predictions that handle objects of different scales. The key here is the utilization of feature maps from different layers in the same network. The feature maps from lower layers improve the semantic segmentation quality, while the global context pooled from a feature map helps smooth the segmentation results.

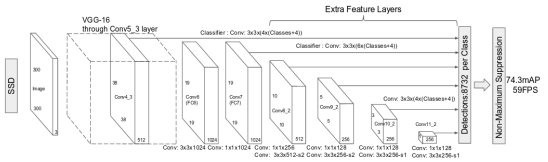

Figure 1. SSD Original Architecture

During training, most of the boxes are classified as negatives due to the sheer high number of default boxes and the Jaccard overlap threshold of 0.5. This skews the training examples distribution heavily towards the negatives, making the training harder. For a stable and faster optimization, one would want to have an even distribution of negatives and positives. The authors propose hard negative mining, which removes some of the low confidence loss examples for each of the default boxes in order to maintain the ratio between the positives and negatives above 1:3.

The authors also augment the training data by randomly sampling some patches of the images to make the model more robust.

4. Architecture of the Network

We implement a variant of the SSD model and train our model on PASCAL VOC2007 [1] dataset. The Figure 1 above shows the original architecture of the SSD model. The input to the model is a 300×300 image. For the base network, instead of the VGG16, we use VGG19, which is pre-trained on ImageNet (training of VGG19 on ImageNet takes several weeks).

On top of the base network, we use a series of convolutional layers with decreasing sizes, 1024, 512, 256 and 128, in that order. conv4_4 is the layer where we cut off the VGG19 network, equivalent to the layer conv4_3 of VGG16 from the figure above. The layers conv4_4, conv7, conv8_2, conv9_2, conv10_2, conv11_2 are used to predict both the offsets to the default boxes and the confidence scores for the presence of an object class inside the boxes.

For conv4_4, conv10_2 and conv11_2, 3 default boxes with aspect ratios of $\{1, 2, \frac{1}{2}\}$ are used. We also add one extra default box for the aspect ratio of 1.0 with a different scale, based on the authors' suggestion. For rest of the layers, default boxes with aspect ratios of 3 and $\frac{1}{3}$ are also added.

All the auxiliary layers (non-VGG19 layers) use a consistent scale scheme described by the authors in the original SSD paper. Hence, the authors suggest using an L2 normalization technique (introduced in [5]) for the VGG19 layer conv4_4 to scale the feature norm at each location in the

feature map to 20 and learn the scale during backpropagation.

We retain the loss function and the hard negative mining techniques from the original implementation of SSD. We train our model with an Adam optimizer with a learning rate of 10^{-3} and an early stopping condition.

For more details of our implementation of the SSD model, please refer to `https://github.com/shpach/ssd_keras`.

5. SSD Experiments and Results

We are interested in learning about SSD as much as we can by carrying out different experiments with the architecture described in previous section 4.

5.1. Experiments

Following are some of the experiments we carried out and our findings from those experiments.

5.1.1 VGG19

This is the same network as the one described in 4 where we replace the VGG16 base network from authors' implementation with a VGG19 network. By using a more complex model, we expect to gain more representational power and thus better image classification. We keep the same multi-scale feature maps except for conv4_3 of VGG16, which we replace with conv4_4 from VGG19. We also want to tweak the weights in the base network to better suit the problem of object detection, so we fine-tune these weights with a learning rate of 0.001 using Adam.

During the first couple of epochs of training, our loss started to shoot up, most likely due to the fact that we were ruining the integrity of the pre-trained weights. However, after 50,000 iterations, the training loss stabilized around 5.13 and validation loss was 5.085.

5.1.2 InceptionResNetV2

For our next experiment, we decided to use a stronger image classifier as our base network: InceptionResnetV2. VGG19 has a top-5 error rate of 9.00% whereas the InceptionResnetV2 has 3.7%. However, the model is much more complex, and we want to avoid the exploding loss problem we encountered earlier. For this reason, we freeze the weights in the new base network and use the same SSD architecture in the lower layers. We set the learning rate to 0.001 using Adam here as well.

Unfortunately, this new model overfits aggressively. After 60,000 iterations the training loss was 5.07 and validation loss was 83.36.

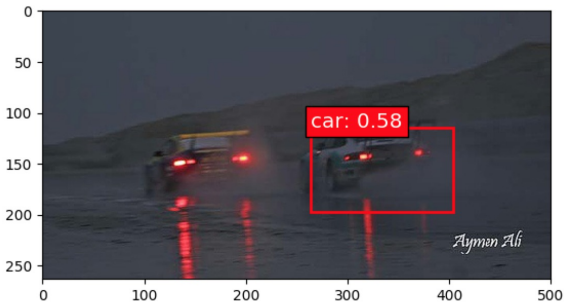

Figure 2. SSD Result

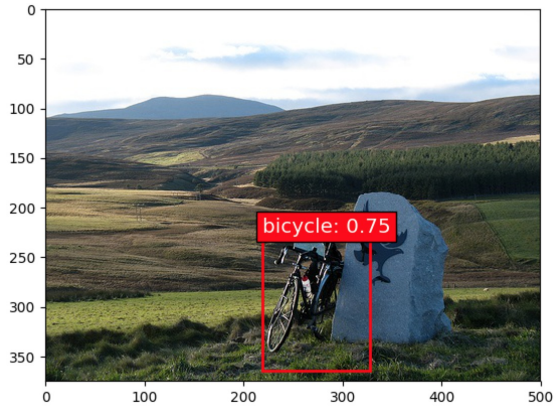

Figure 3. SSD Result

5.1.3 VGG19 and Other Changes

For our last major experiment, we went back to the VGG19 base network, but this time we added batch normalization layers in between the convolutional layers of the SSD-specific network. On top of this, we added more aspect ratios combinations for the default boxes. Specifically, for every prediction layer, we had the aspect ratios: $\{1, 2, \frac{1}{2}, 3, \frac{1}{3}\}$. The original model only had these aspect ratios for the middle three layers. For the upper two feature maps that are used in the classifier, we added aspect ratios 4 and $\frac{1}{4}$.

Since the original SSD paper mentioned issues in detecting small objects, by adding these extra default boxes at the upper layers, we give the model more chances to detect small objects. The final training loss was 4.72 and validation loss 4.76 after 60,000 iterations.

5.2. Results

Figures 2, 3, 4 show the result of our best model (VGG19 and Other Changes) on some images. As we can see, the object detection is on point, but the location of boxes is slightly off in one of the cases.

In the other two examples, figures 5 and 6, we can see

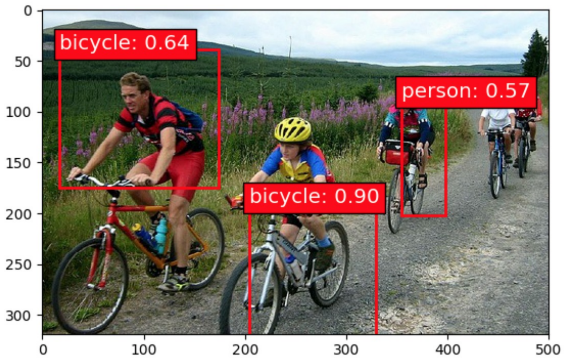

Figure 4. SSD Result

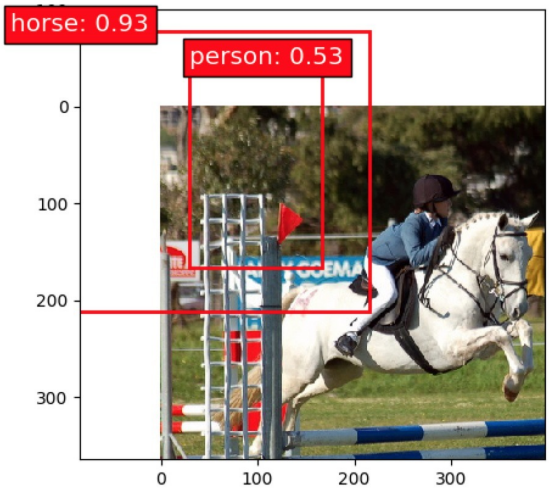

Figure 5. SSD Box Detection Issues

that the model actually guesses the relative positions of the objects correctly, as well as their class with fairly high probabilities, but the bounding boxes look completely off.

Our model is actually a very good image classifier in this sense and understands context. However, for a fair amount of images, the bounding box coordinates are negative. This leads us to believe that there is something flawed about our scheme to match default boxes to an object, or that we are still suffering from a bias issue and that we need more training time. Our model is learning to predict outside of the bounds of the image, so we might also be able to drastically increase our localization accuracy by explicitly restricting the domain of values our model can predict.

From the Figure 7, we can see that, at the end of 60k iterations, our best model is quite close to the authors' version of SSD in terms of both the training and validation losses.

Following is a brief summary of the training comparisons of different models we tried.

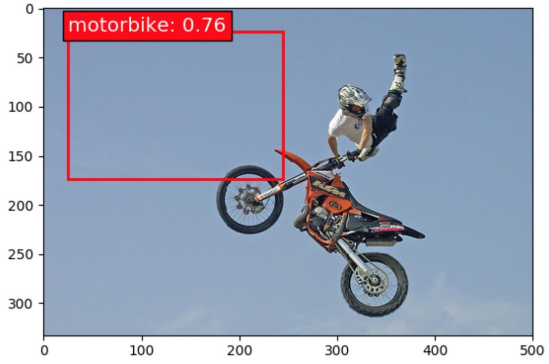

Figure 6. SSD Box Detection Issues

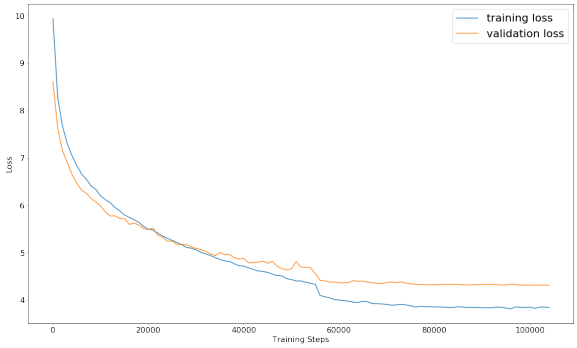

Figure 7. Original Author's SSD Loss History

	Training Loss	Validation Loss
Original SSD300	3.9	4.6
VGG19 SSD300	5.13	5.09
InceptionResNetV2 SSD300	5.07	83.36
VGG19 + more default boxes SSD300	4.72	4.76

We also compute the Mean Average Precision (mAP) of our best model to compare against the authors' version of SSD. mAP denotes the precision of the model averaged over all the 20 classes.

	Mean Average Precision
Original SSD300	0.738
VGG19 + more default boxes SSD300	0.52

5.3. Conclusion

We have designed a novel object detection architecture which modifies the existing implementation of SSD using modern base networks and additional default boxes. Although our model matches the authors' version in training

and validation losses, it falls short on the mAP metric.

The issue most likely lies in the way we predict the bounding box offsets. Unlike the authors' version, our model sometimes predicts negative co-ordinates for the boxes causing the overlap with the default boxes to decrease.

In our knowledge, there is no implementation of SSD which uses modern base networks or modern techniques like batch normalization. We have taken the first step in that direction and are quite optimistic about our approach.

5.4. Future Work

For the future, we may want to tweak some more parameters of our model. Initially, when we tried to jointly optimize the feature extraction and object detection mechanisms of SSD, we started training with a learning rate of 0.001. However, this is a typical learning rate we would use to train such a network from scratch. If we unfreeze the weights of VGG19, we may experiment with lower learning rates (e.g. 1e-6) as to not tamper with the existing weights. However, this may come with the tradeoff of training the entire model slowly.

Currently, we are training our model for the same number epochs as the authors' original approach. However, we may want to train the model for more time, as we saw both our training and validation loss slowly and steadily decrease over the last couple epochs. This is indicative of the model's ability and potential to learn more.

Additionally, we may want to restrict the domain of the box offsets our model can predict, in order to decrease the localization loss even further.

6. Real-time Object Detection Approach

As mentioned in the abstract, we present a video analysis demo with object detection and tracking. SSD offers a method for detecting objects in images using a single deep neural network. A single forward pass of the network identifies all the object classes in a frame with high accuracy. This makes SSD a perfect candidate for real-time object detection. We build a simple real-time object detection algorithm with SSD architecture for object detection and Kernelized Correlation Filters (KCF) for tracking.

6.1. KCF

Most modern trackers use a discriminative classifier which distinguishes between the target and the surrounding environment. In order to cope with natural image changes, the classifier is usually trained with translated and scaled sample patches. In such samples, any overlapping pixels are constrained to be the same, therefore they are riddled with redundancies.

Based on this simple observation, Henriques et al. proposed an analytic model for datasets of thousands of translated patches in [3]. If the resulting data matrix is circular, it can be diagonalized with the discrete Fourier transform, reducing both storage and computation by several orders of magnitude. Interestingly, for linear regression, the formulation is equivalent to a correlation filter, used by some of the fastest competitive trackers. For kernel regression, Henriques et al. derived a new Kernelized Correlation Filter (KCF), that unlike other kernel algorithms has the exact same complexity as its linear counterpart. KCF outperforms top-ranking trackers such as Struck [2] on a 50 videos benchmark, despite running at hundreds of frames-per-second and being implemented in a few lines of code.

6.2. Algorithm

In real-time applications, pure detection algorithms often encounter issues such as missing boxes, merged or colliding boxes, single frame classification error. In video applications, however, we can use a tracking method to utilize information from the detection history to refine the detection result and avoid these errors.

Our algorithm achieves two things, it performs a smoother object tracking for videos and it stores an object image with an ID to the local server. The algorithm performs detection and tracking periodically at an interval of four frames. The first frame is used for object detection, while the following frames are used for both detection and tracking. For each of the following frames, we have a list of tracking boxes and a list of detection boxes. We calculate the Intersection over Union ratio (IOU) among the tracking and detection boxes. And then we find a two-way best match between the tracking box list and the detection box list (from one list to another and vice versa) according to the IOU and labels.

To each bounding box, there are two additional variables which help with the tracking algorithm, Live Count and Save Count. Live Count (livecnt) counts how many times the box had missed the object detection. This variable can solve conflicts in the cases where an object has already left the frame and can also avoid the situation that the detection module missed some of the consecutive frames of the object movement. Save Count (savecnt) defines the lifetime of a box for matching. It varies according to the video and the speed of the object.

In order to obtain the best matching, we apply the following steps to each tracking box from the list:

- Matching found: In this scenario, there is a matching detection box which has an IOU greater than the threshold and the same label as the tracking box. This implies the object has is still alive in this current frame. As far as the position with respect to the frame is concerned, we give preference to the detection box and adjust the position of the tracking box accordingly.

Figure 8. Video Demo Screenshot

- No matching found: In this case, the tracking box has no matching detection box. In the algorithm, we deduct variable livecnt by 1. But we still preserve the box and add it to a potential list in case it is a miss by the detection algorithm. If the box can not be matched in six consecutive frames, we consider it has left the fame and remove it.

We then iterate over the detection boxes in the list. If there is a new box in the detection list and it is not immediately matched with a tracking box, we consider it as a new object. We append the new box to the existing detection box list and try to match it with a tracking box in the next six consecutive frames.

For every box in the potential list, we then check the overlap. We still use IOU to remove duplicate object detections. Since we have object ids for each of the new boxes, we only reduce the duplicates to the same object, not the same label.

In the end, the variables Live Count and Save Count are a function of the frame rate and average speed of the objects in the video. Thus, they can be set after a single pass over the video frames.

6.3. Results

We use the original SSD model pre-trained on the VOC2007 dataset for object detection. A video demo can be found at `https://youtu.be/UvZ9-yo7Xgg`. The video is taken during rush hours at the intersection of 59th Street and 5th Avenue in Manhattan, New York.

Figure 8 shows a screenshot from the video demo. We can clearly see that the algorithm is able to detect all the cars and people in the image.

In Figure 9, we can see SSD at work successfully differentiating between a car and a bus. Figure 10 shows a proper detection of people, cars and a bicycle in the same frame.

Figures like 11 often appear in insurance claims for road accidents. A fast, accurate object detection and tracking algorithm, like ours, could help resolve such cases.

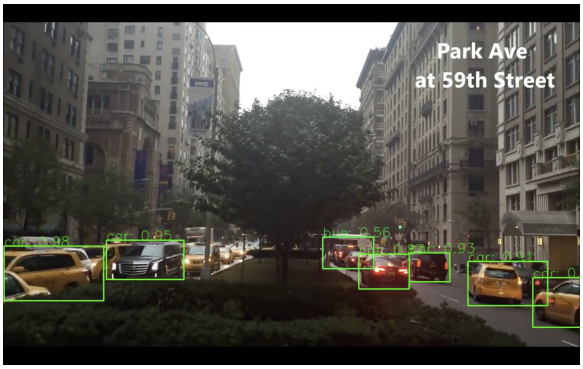

Figure 9. Video Demo Screenshot

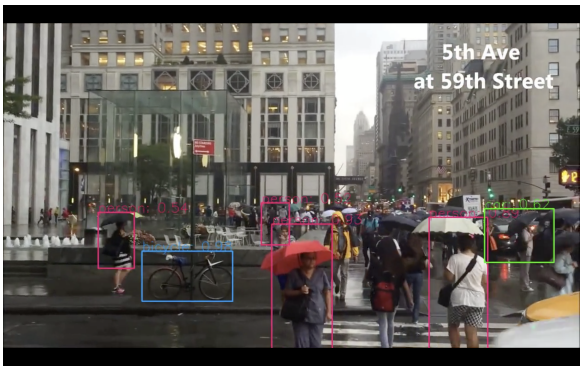

Figure 10. Video Demo Screenshot

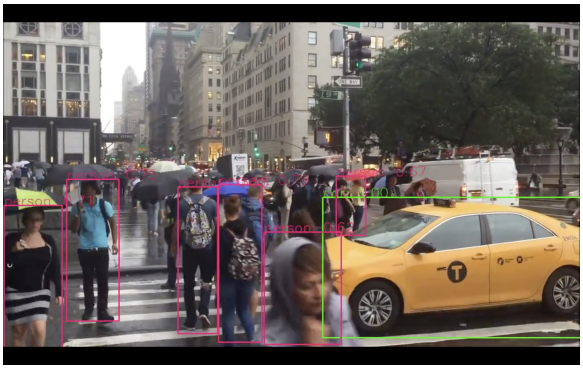

Figure 11. Video Demo Screenshot

Please visit the youtube link of our video to see our object tracking algorithm in action.

6.4. Applications

The real purpose of this demo is to showcase the potential of a fast object detection model like SSD in video tracking. Since our algorithm associates each object image with an ID, we can use it to track individual objects rather than simple object classes. Applications of real-time object tracking include:

- tracking down rouge vehicles using surveillance videos

- pedestrian detection in self-driving cars
- settling insurance disputes over road accidents

6.5. Future Work

To further improve our video application, we may want to add a feedback loop and train another network with this feedback. Specifically, this network's input would take the concatenation of the SSD's output on the current frame and the SSD's output with KCF tracking on the previous frame, and then finally apply KCF on the new output.

While we would have liked to do this, we not only need a labeled video data that had bounding boxes for each frame but also each bounding box must be labeled with one of the twenty classes that exist within the PASCAL VOC 2007 dataset.

7. Achievements

Our goal was two-fold. We wanted to experiment with the SSD model and use it along with KCFs to present a real-time object detection and tracking algorithm.

We have been able to implement the original SSD model in Keras and perform some experiments with the architecture as described in section 5. Although we managed to only match the training and validation losses to that of the author's model, we have gained a significant insight into building complex networks from scratch, training them and utilizing them. Our work on modernizing SSD architecture provides a strong foundation for other Deep Learners to build upon.

We have successfully implemented an object tracking algorithm on top of the SSD model. Our video analysis demo presents insightful applications of the algorithm.

8. Acknowledgement

We would like to thank Prof. Iddo Drori for giving us an opportunity to work on this project. We would like to thank Chenquin Xu for guiding us along the path.

References

[1] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[2] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M. Cheng, S. L. Hicks, and P. H. S. Torr. Struck: Structured output tracking with kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2096–2109, Oct 2016.

[3] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, March 2015.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[5] W. Liu, A. Rabinovich, and A. C. Berg. Parsenet: Looking wider to see better. *CoRR*, abs/1506.04579, 2015.

[6] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[7] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[8] C. Szegedy, S. E. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. *CoRR*, abs/1412.1441, 2014.