Deep Drone: Object Detection and Tracking for Smart Drones on Embedded System

Song Han, William Shen, Zuozhen Liu
Stanford University

Abstract

In recent years, drones have been widely adopted for aerial photography at much lower costs. However, capturing high quality pictures or videos using most advanced drones requires precise manual control and are very error-prone. We are proposing Deep Drone, an embedded system framework, to power drones with vision: letting the drone to do automatic detection and tracking. In this project, we implemented the vision component which is an integration of advanced detection and tracking algorithms. We implemented our system onto multiple hardware platforms, including both desktop GPU (NVIDIA GTX980) and embedded GPU (NVIDIA Tegra K1 and NVIDIA Tegra X1) and evaluated frame rate, power consumption and accuracy on several videos captured by the drone. Our system achieved real time performance at 71 frames per second(fps) for tracking and 1.6 fps for detection on NVIDIA TX1. The video demo of our detection and tracking algorithm has been uploaded to Youtube: `https://youtu.be/UTx2-5a488s`.

1. Introduction

Modern drones are be equipped with cameras and are very prospective for a variety of commercial uses such as aerial photography, surveillance, etc. In order to massively deploy drones and further reduce their costs, it's necessary to power drones with smart computer vision and autopilot. In the application of aerial photography, object detection and tracking are essential to capturing key objects in a scene. Object detection and tracking are classic problems in computer vision. However, there are more challenges with drones due to top-down view angles and real-time constraints. Additionally, a challenging problem is the strong weight and area constraint of embedded hardware that limits the drones to run computation intensive algorithms, such as deep learning, with limited hardware resource.

Deep Drone is a framework that intends to tackle both problems while running on embedded systems that can be mounted onto drones. For this project, we present our vision system pipeline and its performance along with accuracy on multiple hardware platforms, and we analyzed the trade-off between accuracy and frame rate.

2. Related work

Deep neural networks, object detection and object tracking are the three major components in our work. We first present an overview of past work, and then describe our improvements.

Deep neural network is the state-of-the-art technique in computer vision tasks, including image classification, detection, and segmentation. AlexNet [11] is the classic network proposed in 2012 that has 8 layers and 60 million connections, and won the Imagenet contest in 2012, spawning a lot of improvements later. After that, VGGNet [16] was proposed, which has 16 layers and 130 million parameters. Both AlexNet and VGGNet have bulky fully connected layers which results in huge model sizes. GoogleNet [17] is a more compact CNN that consists mostly of conv layers, it uses the inception model that has 1x1, 3x3 and 5x5 convolution kernels with different scale. It also has multiple loss layers to prevent vanishing gradient problem. ResNet [6] was proposed in 2015 and greatly improved the image recognition accuracy, it adds bypass layers to let the network learn the residual rather than the absolute value. SqueezeNet [9] was proposed recently that is aggressively optimized for model size. It has 50x less connections and half the computation than AlexNet but has higher accuracy. After Deep Compression [5, 4], the model is only 470KB and fits well into the last level cache.

Fast R-CNN [3] is a Fast Region-based Convolutional Network method (Fast R-CNN) for object detection. In this algorithm, an input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

Faster R-CNN [15] made further improvements on Fast R-CNN that introduce a region proposal Network (RPN)

that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully-convolutional network that simultaneously predicts object bounding boxes and scores at each position. RPNs are trained end-to-end to generate high-quality region proposals. Because of the region proposal network are fused and could be trained end to end, the network is faster than fast R-CNN.

YoLo Detector [13] is a new approach to do object detection. Prior work on object detection re-purposes classifiers to perform detection. Instead, YoLo use object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. The unified architecture is extremely fast but not as accurate as Faster R-CNN.

KCF[7, 8] is the kernelized correlation filters used for detection. It uses the characteristic that under some conditions, the resulting data and kernel matrices become circulant. Their diagonalization by the DFT provides a general blueprint for creating fast algorithms that deal with translations, reducing both storage and computation by several orders of magnitude, obtaining state-of-the-art trackers that run at 70 frames per second on NVIDIA TK1 and is very simple to implement.

MDNet [12] is the state-of-the-art visual tracker based on a CNN trained on a large set of tracking sequences, and the winner tracker of The VOT2015 Challenge. The network is composed of shared layers and multiple branches of domain-specific layers, where domains correspond to individual training sequences and each branch is responsible for binary classification to identify the target in each domain. The network is trained with respect to each domain iteratively to obtain generic target representations in the shared layers. Online tracking is performed by evaluating the candidate windows randomly sampled around the previous target state.

However, the drawback of MDNet is that it needs to run CNN to extract image features, making it very slow. Considering the frame rate required by real time tracking, we used the KCF algorithm for tracking, which achieves 70 frames per second on our hardware: a NVIDIA Tegra K1.

3. Contribution

Fast and Faster R-CNN originally used VGGNet for feature extraction. It is accurate but slow. Drones have limited hardware resource both in memory and in computation power, so we need to have smaller network. In order to run the CNN fast enough on embedded device, we didn't use those off-the-shelf network architectures. Instead, we used a relatively shallow and small network to extract image features and to detection. The architecture is shown in Fig 1.

Even using our small network architecture, the detection frame rate is still low on TK1 mobile GPU. To compensate for the slow speed of detection, we used the cheap KCF tracker, although it's less accurate than MDNet, to track on the bonding box returned by the detection pipeline. Thus we have the accurate but slow Faster R-CNN for detection, and have the less accurate but super fast KCF for tracking. Detection is only called when the confidence of the tracker is below certain threshold, which is very infrequent. This architecture makes the pipeline accurate, robust and fast.

Accuracy is not our sole target in this project. We have a thorough evaluation with respect to accuracy, power consumption, speed, and area of different hardware running detection and tracking algorithm. Balancing these hardware constraints, rather than only optimizing for MAP, is our top priority.

4. System Architecture

The software architecture of our vision system consists of two components. The first component is a detection algorithm running Convolutional Neural Network (CNN) and the second part is a tracking algorithm using HOG feature and KCF. These two algorithms are seamlessly integrated to ensure smooth and real-time performance. The detection algorithm, e.g. Faster RCNN, is expensive to compute since CNN-based detection has lots of GOPs per frame and is only called to initialize a bounding box for key object in the scene. The tracking algorithm, e.g. KCF, is relatively inexpensive to compute and can run at a high frame rate to track the bounding box provided by detection algorithm. The main algorithm loop is shown in the pseudo code below, and we discuss the detection and tracking details in the next two subsections.

Algorithm 1 Detection and Tracking Pipeline for Deep Drone

```
boxFound ← false
while true do
    f ← new frame
    while boxFound == false do
        detection(f)          ▷ Invoke detection algorithm
        if Box is detected then
            boxFound ← true
        end if
    end while
    tracking(f)               ▷ Invoke tracking algorithm
    if Tracking is lost then
        boxFound ← false
    end if
end while
```

4.1. Detection

Drones are mainly used to take pictures of human, so we focus on detecting people as first step. We made further assumption in this project that there is only one person of interest to track, so that the person with the highest detection score is our target. We have used two detectors to do people detection: Faster RCNN and Yolo detector. We analyze them both in below sections.

4.1.1 Using Faster R-CNN

We used a 7-layer convolutional neural network on Faster R-CNN[15] for people detection. The framework takes raw image frames from a video stream and outputs bonding boxes and target classes for detected objects.

We used a in-house model trained on the KITTI[2] dataset. KITTI contains a rich amount of training samples that include objects such as cars, pedestrians and cyclists and can easily generalize to our task. In this project, our interested detection target is people so we modified the script to detect people only. A detailed architecture is shown in Figure 1.

We measured the accuracy(mAP) and speed of our in-house model and compare it with the baseline, shown in table1. Our in house model has slightly worse accuracy than the baseline, but has 12x faster speed.

Table 1. Accuracy and runtime for our detection network (runtime is measured on GTX980 GPU)

Model	mAP	Runtime
Baseline[15]	65.9 %	2s
Ours	62.0 %	0.17s

4.1.2 Using Yolo Detector

We also tried Yolo detector[14], because it's easier to compile to mobile. It's also a faster alternative, at the cost of worse accuracy. In our experiments, we found Yolo detector unable to detect the cases where the people is small and remote, as shown in Figure 4.1.2, so we didn't use this method.

4.2. Tracking

For tracking we chose KCF algorithm over other state-of-the-art tracking algorithm like MDNet [12], SRDCF [1] or EBT [18] despite them having a better accuracy and performance on the VOT 2015 challenge [10]. The reason behind this decision is that we want real-time tracking for our drone so that we could give consistent control command to the drone while MDNet, SRDCF and EBT all have performance under 5 frames per second.

4.2.1 KCF

KCF [7][8] is a more old school tracking algorithm than MDNet, but it's supposed to be faster and more succinct. The algorithm uses Discrete Fourier Transform to diagonalize data matrix, which is then processed to train a discriminative classifier through linear regression and kernel trick; This new approach is called Kernelized Correlation Filter (KCF).

We found out that KCF runs very fast on video, it takes on average around 8.8 milliseconds to check per frame on a Macbook pro CPU.

The downside of KCF is the requirement that the video has to be continuous. If the video fades to black, the object moves very fast or a jump cut occurs, KCF will have a hard time to recover. When this occur, the peak value of detection score from running Gaussian kernel on correlation filter will suffer a significant drop; it will also return negative bounding box value if it can't find any match. We leveraged this feature to combine KCF with faster RCNN or other detection algorithm to solve the problem. Namely, when KCF is not confident or fail entirely, it will call a detection algorithm in hope of recover.

Table 2. Speed of detection and tracking on different hardware platforms

Hardware Platform	GTX 980	TX1	TK1
Power	150W	10W	7W
Detection	0.17s	0.6s	1.6s
Tracking	5.5ms	14ms	14ms
Tracking Frames/Sec	182fps	71fps	71fps

4.3. Hardware Platform

Drones are special hardware platform with limited space and weight, so computation power is not free lunch. we can't afford to use desktop GPUs to do the detection computation, although it's much faster. To compare hardware platforms, we measured the computation time for detection and tracking in Figure 1, and we show the power consumption of these hardware platforms in the same table. GTX980 is roughly 10x faster than the TK1, but consumes 20x more power. TX1 takes roughly the same power consumption, but is 3x faster than TK1. So, in conclusion, TX1 would be the ideal hardware platform to use for drone's detection.

Looks like TX1 is ideal with respect to speed and power consumption, but it's form factor limited us to put it on the drone. See in Figure 4.3, the TX1 development board is much larger and much heavier than the TK1 board, making it very hard to put on the drone. However, TK1 is smaller and DJI provides the Manifold box to hold the TK1, which makes it very easy to use, and we managed to put our algorithms on the drone.

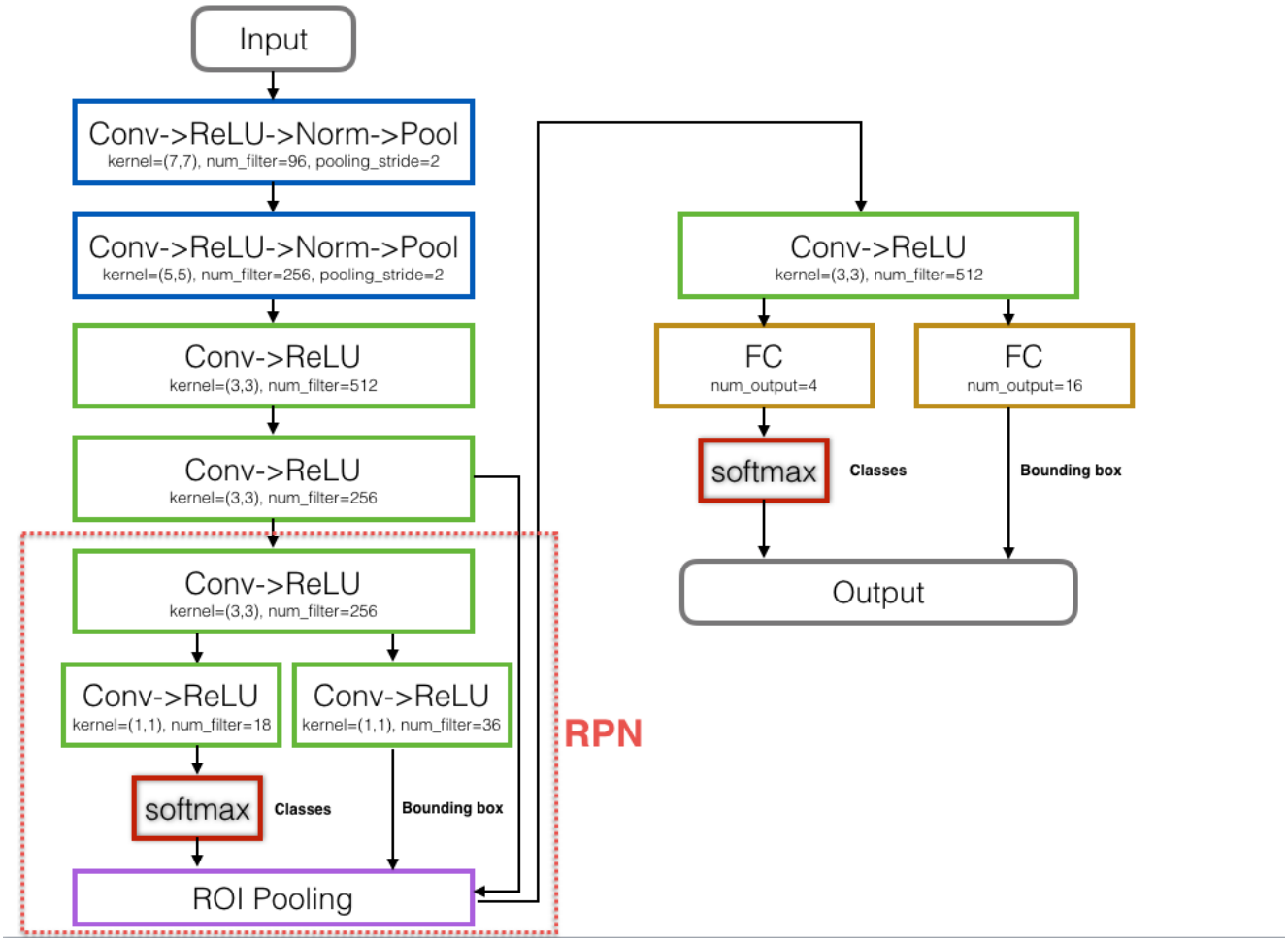

Figure 1. The CNN architecture that we used for detection

In order to deal with the large form factor of the TX1 development board, we bought a small carrier board for TX1. It is shown in Fig 4.3. This carrier board has the size as small as the heat sink, we can unplug the TX1 central board from the full development board and only plug the central board that contains the actual TX1 chip to this carrier board. This makes the size even smaller than TK1. However, there's no free lunch. The interface, especially the power supply isn't compatible with the DJI drone, we haven't got a chance to connect the carrier board with the drone yet, which could be future work. The TK1 is fully working, if not optimized for speed, the TX1 carrier board is not the critical path of our project.

5. Implementation and Experiments

5.1. Detection

Faster RCNN builds on top of Caffe, a deep learning framework that requires multiple dependencies, and has a number of customized region pooling layers. We spent great effort in installing CUDA and Faster RCNN onto all of our desktop and embedded platforms. During installation, we ran into the following issues.

1. We flashed our TK1 with the latest L4T (Linux for Tegra), however, TK1 doesn't support CUDA version higher than CUDA 6.5, so we only installed CUDA 6.5 dev kit on TK1. However, the latest Caffe is not backward compatible with CuDNN v3 and before, if we revert the Caffe to an ealier branch, it won't support the new layers required by faster r-cnn. So we turned off the CuDNN switch for Caffe installation to bypass this problem.

2. Some of Faster RCNN libraries were written in python and compiled into native c++ code using Cython. When installing these libraries onto embedded systems ,i.e. TK1 and TX1, we ran into compilation error on gpu_nms.cpp, a GPU implementation for non

Figure 2. Faster RCNN performs really well on detecting people from drones perspective. Even when the object is far away and twisted (last figure).

Figure 3. Yolo detector performs not as well as Faster RCNN. When the target person is small, the detector fails.

maximum suppression. The root cause was identified as compiler incompatibility in the embedded systems.

Figure 4. KCF tracker performs very well on videos that doesn't have jump cuts. It fails if the video fades to black. However, this is not a problem since detection will be called on this scenario.

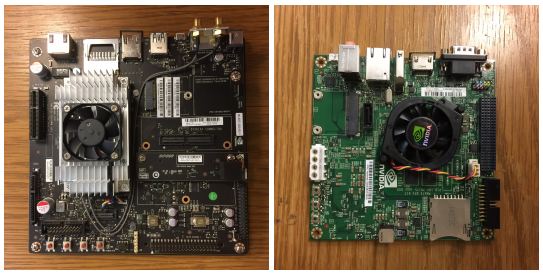

Figure 5. NVIDIA TX1 and TK1 development board we used

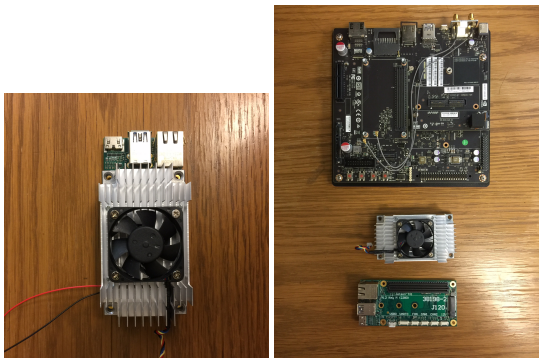

Figure 6. In order to reduce the size of TX1, we bought a small carrier board.

We eventually found a workaround by manually modifying source code in the generated cpp file and successfully passed the compilation.

Finally, we trained a 7-layer CNN Caffe model using GTX 980 and ported the model onto all platforms to evaluate testing performance using both offline and online video streams.

5.2. Tracking

For tracking, the original algorithm for KCF is implemented in Matlab. However, since the hardware limitation of TK1 (storage space and computing power), we implement the algorithm using C++. As the original model proposed, the tracker interface provides an init method that takes in a frame and the starting bounding box, then it learns a model using linear regression from sample patches (with help of cyclic shift) of the frame. Upon receiving a new frame, the tracker's update function is called, it first detects patches of the same size; if the resulting detection score doesn't reach a threshold, different patches under different scales will be detected. We used 110% and 90% re-scaling factor. Finally we return the one with highest score (the new positive patches and negative patches are used here for online learning).

5.3. Handshake between detection and tracking

Since the detection algorithm is implemented in Python, we uses the **Python.h** to realize a C++/Python binding between the two algorithms. The initialization of detection algorithm (loading neural network under Caffe) is stored in the C++ main program as a `PyObject`. Upon receiving a new frame, C++ main program converts the frame (stored in byte array) to a Python ndarray object and passes the resulting array to a call to the detection method. It then parse the result and determine if the detection result matches a threshold (how confident the object is a person). An interesting bug that arise is when importing a Python Module under sudo(which is needed for activating and using DJI drone's camera); this is because some of the PythonThe NVIDIA TX1 and TK1 development board that we used packages are installed with root rw permission only, we worked around this problem by command **sudo su**, and hacked the privilege of using the DJI live camera.

5.4. Interacting with DJI camera library

We use two sets of DJI libraries. It only contains interface code to talk to the camera, we didn't use any DJI code for vision algorithms. First, the camera input module provided by **djicam.h**, which leverage **libdcam** to read in from the built-in camera on the drone. The library only provides three simple functions (`manifold_cam_init`, `manifold_cam_read` and `manifold_cam_exit`) which means we need to manipulate all data from raw pixel arrays; we initialize the camera with TRANSFER_MODE (transferring image input to controller and mobile app) and GET_BUFFER_MODE (store the video input to local buffer byte array in NV12 format, more on format conversion in later subsection). We use the CAM_NON_BLOCK mode (which means not waiting the camera to fully initialized) to ensure that we can give constant control to the drone even if the camera is not set up. We sleep the program and wait for the camera to exit at the end when we return. The specifics of the library can be found here: `https://github.com/dji-sdk/Manifold-Cam/blob/master/djicam.h`

5.5. Offline Detection and Tracking

We first tested our detection and tracking module on our own offline video recordings from different perspectives (a DJI Inspire recording of Song Han playing nunchaku and a GoPro recording of Song Han snowboarding) with the TK1 board mounted on the drone. For detection using faster r-cnn, each frame takes 1.6 second on average (comparing to 0.6s on TX1 which is small enough to mount on the drone but not yet supported by DJI). For tracking under KCF, each frame only takes 14ms (71 fps). Running our detection and tracking module on the two videos exposes several interesting problems to us. First, the tightness of detection's bounding box result will affect how well tracking algorithm performs. If the bounding box is not tight enough, it might incorporate irrelevant subject (in the nunchaku case incorporating big chunk of shadow, Figure 2 sub-image 4), tracking will then mistakenly think that it's the irrelevant subject it wants to track. Second, detection doesn't work well when the person is disguised as a bulkier figure. As in the snowboarding video, when the viewpoint is on the side of the person, with the help of helmet, face mask and bulky cloth, it's hard to detect the person. We address the two problem by adjusting the confidence threshold for detection score, and retrain the neural network with images from different angles and viewpoints.

5.6. Online detection and tracking on DJI live cam

We then adjust the detection and tracking module to function with DJI M100's live cam. As mentions in section earlier, the camera read in data as raw byte array in NV12 format, the format has three components for each pixel: a luma component (the brightness) Yánd two chrominance (color) components U and V; since the detection and tracking algorithm are all based on RGB pixel values. We do the following conversion to obtain a RGB representation of the frame (clamping means restrict the value within the 0-255 range for RGB):

$$R = clamp(Y + 1.4075(V - 128))$$

$$G = clamp(Y - 0.3455(U - 128) - 0.7169(V - 128))$$

$$B = (int)(Y + 1.779(U - 128));$$

After obtaining the frame encoded in RGB format, we use similar approach as offline video and detect and track video frame by frame. A challenge surfaces in this step is that since we are doing detection on a live stream, the subject might be moving rapidly while we are detecting, this means that there is a possibility that the object has already moved out of the bounding box while detection finishes analyzing a frame from Δt seconds ago. We addressed this issue by initialize the tracker with the frame from Δt seconds ago instead of the current frame. The tracker will then train it's positive patches and negative patches with correct bounding area. This solved the problem unless the object has deformed too much in the Δt time frame.

5.7. Controlling the Drone

We are controlling the camera using the OnBoard SDK provided by DJI. We first need to send an activation data to the CoreAPI driver. Since DJI recently upgraded their drone operating system and their Onboard SDK not updated accordingly, this step causes tremendous trouble. We had to contact DJI's engineer who wrote their Onboard SDK to get the new version of encryption key (a magic

number) to successfully activate. Then we can gain control of the camera by sending `GimbalAngleData` to it. A `GimbalAngleData` contain three import field corresponding to the three spatial degree of freedom for the camera: yaw, roll and pitch.

6. Conclusion

We present Deep Drone, a detection and tracking system running real time on embedded hardware, that powers the drones with vision. We presented our software architecture that combines the accurate but slow detection algorithm and the less accurate but fast tracking algorithm, to make the system both fast and accurate. We also compared the runtime, power consumption and size of different hardware platforms, and discussed implementation issues and corresponding solutions dealing with those embedded hardware.

Acknowledgment

We thank Amber Garage for equipment support.

References

[1] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Learning spatially regularized correlation filters for visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4310–4318, 2015.

[2] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[3] R. Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015.

[4] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[5] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[7] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *Computer Vision–ECCV 2012*, pages 702–715. Springer, 2012.

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

[9] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[10] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernandez, T. Vojir, G. Hager, G. Nebehay, and R. Pflugfelder. The visual object tracking vot2015 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1–23, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. *arXiv preprint arXiv:1510.07945*, 2015.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*, 2015.

[14] 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.

[15] 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.

[16] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[18] G. Zhu, F. Porikli, and H. Li. Tracking randomly moving objects on edge box proposals. *arXiv preprint arXiv:1507.08085*, 2015.