Deep Neural Networks: a Comparison on Different Computing Platforms

Md Modasshir, Alberto Quattrini Li, and Ioannis Rekleitis
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC, USA
modasshm@email.sc.edu,{albertoq,yiannisr}@cse.sc.edu

***Abstract*—Deep Neural Networks (DNN) have gained tremendous popularity over the last years for several computer vision tasks, including classification and object detection. Such techniques have been able to achieve human-level performance in many tasks and have produced results of unprecedented accuracy. As DNNs have intense computational requirements in the majority of applications, they utilize a cluster of computers or a cutting edge Graphical Processing Unit (GPU), often having excessive power consumption and generating a lot of heat. In many robotics applications the above requirements prove to be a challenge, as there is limited power on-board and heat dissipation is always a problem. In particular in underwater robotics with limited space, the above two requirements have been proven prohibitive. As first of this kind, this paper aims at analyzing and comparing the performance of several state-of-the-art DNNs on different platforms. With a focus on the underwater domain, the capabilities of the Jetson TX2 from NVIDIA and the Neural Compute Stick from Intel are of particular interest. Experiments on standard datasets show how different platforms are usable on an actual robotic system, providing insights on the current state-of-the-art embedded systems. Based on such results, we propose some guidelines in choosing the appropriate platform and network architecture for a robotic system.**

***Keywords*-Deep Neural Networks; Embedded Systems; Comparison**

I. Introduction

The last few years have seen a tremendous increase in the popularity of the deep learning paradigm. In all the major computer vision conferences, Convolutional Neural Networks (CNNs), and their variants, dominate the scene. Furthermore, very impressive demonstrations of this technology appear with high frequency, from near perfect detection scores (97% accuracy) [1], to tracking of multiple people in a crowd [2], and to controlling a flying robot without providing explicit commands [3]. In the field of robotics while learning approaches have always been used, for the first time, at IEEE International Conference on Robotics and Automation (ICRA) 2018, learning was the most common keyword, followed by planning as the second.

Deep Neural Networks require high computational power that recently has become available thanks to GPU-computing [4]. However, such computational power has not been typically available on embedded systems mounted on mobile robots, making the deployment of DNNs a challenge.

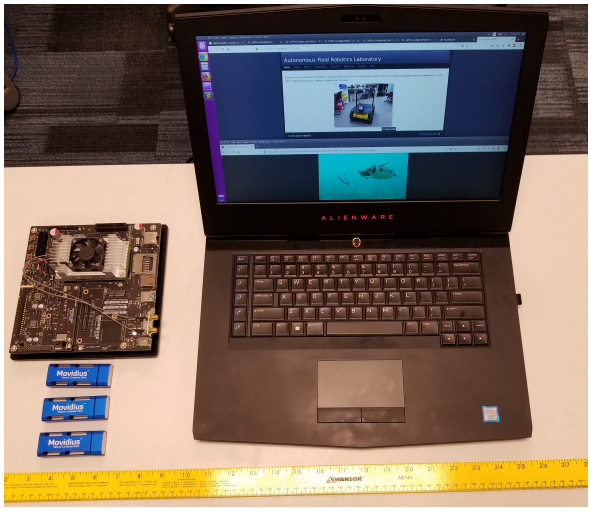

Figure 1: Computing platforms used in this paper.

In recent years, major companies, such as NVIDIA and Intel, produced small factor modules that are able to run DNNs.

Differently from other type of analysis, the goal of this paper is to analyze the current performance of such embedded systems for DNNs for tasks relevant to the robotics field. In particular, in this paper, we compare different networks for detection and classification on the computing platforms shown in Figure 1:

- the Alienware gaming laptop from Dell.
- Jetson TX2 from NVIDIA.
- Neural Compute Stick from Intel.

In the comparison a high-end gaming laptop is included for two reasons. First of all, most ground robotic platforms, ranging from the most accessible Turtlebot 2[1] to the bigger outdoor units such as the Warthog [2] can easily accommodate the addition of an extra laptop; especially as it does carry its own power. Second, quite often during field deployments, it is not feasible to carry a desktop equipped with a state-of-the-art GPU to remote locations, in which case a powerful laptop presents a working alternative. For the above two reasons, the gaming laptop is an excellent compromise

[1]http://www.turtlebot.com/turtlebot2/
[2]https://www.clearpathrobotics.com/warthog-unmanned-ground-vehicle-robot/

©2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/CRV.2018.00060

enabling training a DNN on-site.

While our interests are with a variety of different robotic platforms, our focus in this paper is on underwater vehicles, such as the Aqua2 AUV [5] shown in Figure 2. Such a platform has several limitations when it comes to the deployment of a DNN on-line. First of all, the power available is limited, a restriction that is even more severe when it comes to aerial vehicles. Secondly, the physical space inside the vessel is rather confined; prohibiting the introduction of a standard GPU powered video card. Finally, even though the vehicle is inside the water, which provides a natural water cooling system, the heat dissipation inside the robot does not allow for excessive heat to be generated. In the analysis presented in this paper we have compared the different computing platforms using their overall energy consumption.

For generality sake the comparison is performed on publicly available data sets, in particular ImageNet [6] and PASCAL VOC [7], commonly used as benchmarks for evaluating DNNs [8]. In addition, the networks have also been tested in specific datasets pertaining to the marine domain such as the coral images of MLC dataset [9], and in data collected over several field trials by the authors; see Figure 3. As part of the analysis, we provide first a survey and taxonomy of several DNNs available according to their tasks. The goal is to provide guidelines based on a comparative evaluation in order to choose which computing platform suits best for a specific mobile robot and task. Specifically, the following insights/guidelines can be derived:

- How different networks perform on different computing platforms.
- The power required by each network on different computing platforms.
- The effect of different floating point precision.

This paper is structured as follows. The next section provides a brief overview of the approaches used to evaluate the different computing platforms and Section III describes the setting used for our experimental comparison together with the methods used. Section IV shows the quantitative results and Section V discusses them. Section VI concludes the paper and outlines future work.

II. Related Work

While referring to the latest work in Deep Learning, in an exhaustive manner, is nearly impossible, in this section we will define two major categories of deep learning approaches considered highly relevant to perception in robotic tasks. In each category, the focus will be on representative techniques which were tested on the different platforms.

Object Classification: The different DNNs focus on providing a label for a set of objects of interest. Top-1 error and top-5 error percentage is used to evaluate the performance of a network. In particular, top-1 error percentage means how many results with highest confidence

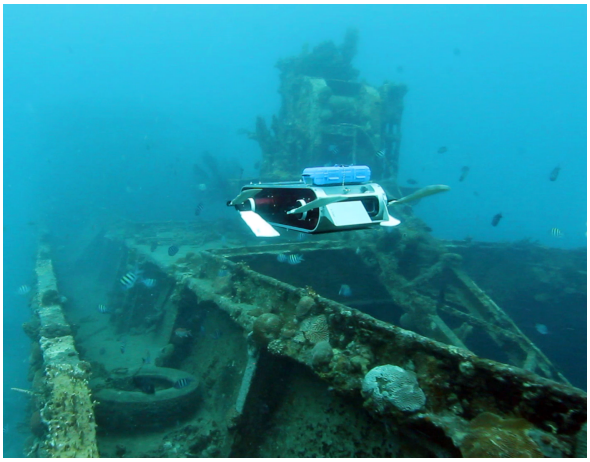

Figure 2: An Aqua underwater robot collecting data over a shipwreck.

do not match the ground truth with respect to the total size of the validation set. Top-5 error percentage is calculated similarly; however, if the ground truth label is within the 5 labels with highest confidence the classification is considered correct.

Alex Krizhevsky et al. crafted a CNN – AlexNet [10] – that contains five convolutional and three fully-connected layers. Such a network was able to achieve at the time top-1 and top-5 test set error rates of 37.5% and 17.0% on a subset of ImageNet (lower values are better). Simonyan and Zisserman proposed VGGNet [11], which has higher depth than other Convolutional Neural Networks. The architecture uses small convolutional layers and on ImageNet had 23.7% and 6.8% as top-1 and top-5 error, respectively. Iandola et al. presented SqueezeNet [1], a CNN that has 50 times less parameters than AlexNet. This was achieved by using smaller filters – i.e, 1×1 filters instead of 3×3 filters, decreasing the number of input channels to 3×3 filters, and downsampling deeper in the network. Results showed accuracy level comparable to that of AlexNet, for less parameters. GoogLeNet [12] designed by Szegedy et al. is a deeper network compared to the above mentioned ones that is able to keep the computational power required bounded. The main idea is to approximate the local sparse structure by available dense building blocks and reduce dimensions when computational requirements are too high. Such a network was able to achieve a top-5 error of 6.67%. He et al. crafted a new CNN, called ResNet [13], where stacked layers fit a residual mapping, instead of an underlying mapping. As a result, the network can be designed to have more layers, with a depth reported in the paper up to 152. The top-1 error and top-5 error are 19.38% and 4.49%, respectively, for a 152-layer ResNet. While all the above mentioned networks are not considering deployment on embedded systems, MobileNets [14] is a class of efficient models that

can be run on mobile and embedded systems. The idea is to use depthwise separable convolution instead of a standard convolution. The proposed networks achieve similar results to popular networks, such as AlexNet and SqueezeNet. The increase in accuracy parallels the time-line of the above DNNs invention.

Object Detection: The task is to detect a single or multiple objects in an image and returning bounding boxes identifying the image area containing the objects of interest. To evaluate the quality of the detection, the precision is considered. In particular, first the area overlapping between a predicted bounding box and the corresponding one in the ground truth is found, and then, according to a threshold, that number becomes either a true positive or a false positive [15]. Again, frame per second is used to evaluate computational efficiency.

Compared to object classification tasks, there are fewer networks available. One of the first networks which was able to achieve relatively good performance is OverFeat [16], where a multi-scale and sliding window approach is applied to the image to then pass it to a Convolutional Network.

After the above breakthrough, the mainstream approach designed networks that take the full image as input. Faster RCNN [17] introduced the Region Proposal Network, namely a fully-convolutional network that detects bounds of objects and associates scores at each position. YOLO [18] was the first network at the time presented which was able to achieve real-time performance, by formulating object detection as a regression problem. SSD [19] improves the results of YOLO, by using multi-scale convolutional bounding box outputs, connected to multiple feature maps at the top of the network. DetectNet [20] from NVIDIA researchers uses a fixed 3-dimensional label format to capture the class and the coordinates of the object. Such a data structure allows DetectNet to get images of any size with a variable number of objects present. Full experiments are yet to run, but preliminary results showed good computational efficiency. The MobileNet framework has been used also for object detection, modifying some of the networks – e.g., SSD, Faster-RCNN – achieving good performance [14] in systems with lower computational power.

While precision for the different networks is comparable, work pushed the efficiency so that such methods can be applied to real-time applications.

Typically, in all of the papers presenting a new network, a comparison is reported to show improvements in accuracy/precision and/or computation time measured in frames per second (FPS). In the literature, there are some benchmark analysis, focusing for example on the effect of floating point precision [21], [22] and testing deep learning software tools on different GPU for servers and desktops [23]. Benchmark tools can also be found online [24]–[28] to automatically test several networks. These studies, however, did not focus on embedded systems and their performance, an aspect that is fundamental for robotics systems. As such, in this paper, we bridge this gap, by looking at the performance of several networks for object classification and detection.

Figure 3: Representative example of the two processes applied to environmental monitoring: (a) Object Classification: four classes of corals (Mustard, Starlet, Star, and Brain); (b) Object Detection: detecting a brain coral.

III. Experimental Setting

Following emerging best practices for experimental evaluation [29], we report the details for repeatability and reproducibility. In the experimental evaluation performed in this paper, we consider the following experimental dimensions, which define an *experiment*:

- Platform used.
- Task.
- Deep Neural Network used for each task.

The DNNs used are implemented within the Caffe framework [30], which is implemented in C++.

The metrics considered to evaluate the performance are the following:

- Quality of prediction (specified for each task).
- Frames per second.
- Energy consumed, measured with a power meter and recorded every 5 seconds for every frame.

Each experiment is run by considering the standard datasets that are commonly used for the evaluated task. Note that any preprocessing is done initially, so that all the metrics – i.e., FPS, energy consumed – consider only the GPU processing.

In the following section, details on each experimental dimension are reported.

A. Platforms

Considering mobile robots, we tested three computing platforms running Ubuntu 16.04.3 LTS.

A Dell Alienware gaming[3] laptop with Intel Core i7-7820HK as CPU, 32 GB DDR4 as RAM, and an NVIDIA

[3]http://www.dell.com/en-us/shop/dell-laptops/alienware-17/spd/alienware-17-laptop/dkcwkblg0744?ref=519_tnt_prodTitleNoStrip

Figure 4: Sample of images for classification, where the task is to recognize whether one of the classes considered is in the image or not.

GTX1080 as GPU was the most powerful machine tested.

NVIDIA Jetson TX2[4] is a computation processor board equipped with an NVIDIA Tegra processor. Both platforms with NVIDIA GPU have the latest current CUDA version – i.e., 9.0. As suggested by NVIDIA, we ran the deployed CNN with FP16.

Intel Movidius Neural Compute Stick[5] is a USB stick with a Myriad 2 processor. The SDK version used is 1.12.00. We used it by plugging into an Intel NUC 5i7RYH[6] with a an Intel Core i7-5557U and 16 GB of RAM. The NUC computer is compatible with what is inside an Aqua2 vehicle. Table I shows a summary of the main characteristics of each of the platforms able to run DNNs, where the details were taken from the corresponding website.

B. Tasks and Compared Deep Neural Networks

We consider the tasks of object classification and detection.

As described in Section II, we consider as prediction quality metric for object classification top-1 error. Experiments are run over a subset of ImageNet, which includes 10 classes. Each class contains 400 images randomly selected in a uniform way. We chose the 10 classes that have a large number of instances in the dataset, so that the compared networks do not suffer from underfitting. Figure 4 shows a representative sample of images for each class.

For object detection, we calculate the precision. PASCAL VOC [7] is used as dataset. Specifically, 1000 images were randomly selected with 20 classes. Each image could contain multiple objects of the same class, as shown in Figure 5.

Note that, instead of considering the ground truth, we consider the classification from the laptop as a baseline for the other two devices. The reason is that we are interested in comparing the performance of the platforms, and the laptop has the highest computational power.

[4]https://developer.nvidia.com/embedded/buy/jetson-tx2-devkit

[5]https://developer.movidius.com/

[6]https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc5i7ryh.html

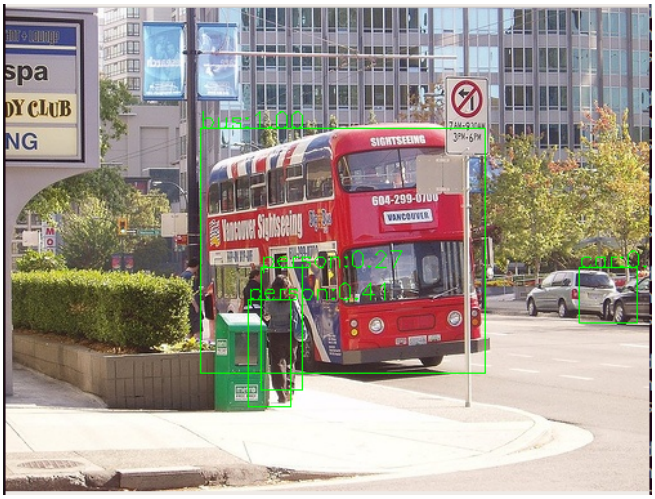

Figure 5: Sample image for object detection, where the task is to find the boundary of objects of interest.

We chose the networks based on two criteria: first, the DNN should be able to run on all the platforms compared in this paper, and second, if a DNN's performance was below 15 FPS on the laptop it was dropped as they would definitely not satisfy real-time constraints on the embedded systems. Note that, however, we kept MobileNet as it is supposed to run on mobile and embedded systems and shown to run efficiently. The platform that introduces most constraints is the Intel Movidius Neural Compute Stick, because its current SDK does not support many operations.

Table II shows the number of layers and hyper-parameters for each DNN compared. Note that some of the papers did not report the number or the architecture explicitly, as such, it was inferred from the description of the network.

For each of this network, we downloaded the available trained model from the authors and we tuned the hyper-parameters for the 10 classes used for object classification. Figure 6 shows a sample graph plotted during the training process for one of the networks. Please note that the loss (in green) decreases, while accuracy increases. As PASCAL uses only 20 classes and networks were pre-trained on those, we just downloaded the available model online.

IV. Results

First, results from the object classification task are presented. Table III shows the top-1 error of the classification results obtained with the Jetson and Movidius, compared to the results obtained with the laptop. While both NVIDIA based platforms perform on par, networks deployed on Movidius are not able to consistently predict classes, compared to the Alienware. Indeed, the top-1 error is generally high.

Table IV reports results on FPS for the different networks over the three different platforms.

The Alienware equipped with the NVIDIA GTX1080 is generally the fastest one; for example with AlexNet 463 FPS

Platform	Model	Architecture	Memory	Mem. bandwidth	CUDA Cores	T-FLOPS	FP Precision	Nominal Power
Dell Alienware 17	NVIDIA GTX1080	Pascal	8 GB GDDR5 (dedicated)	320 GB/s	2560	9	FP16,FP32	240 W
NVIDIA Jetson	TX2	Pascal	8 GB 128-bit LPDDR4 (shared)	58.3 GB/s	256	1.5	FP16,FP32	15 W
Intel Movidius	Neural Compute Stick	Myriad 2	4 GB LPDDR3 (dedicated)	400 GB/s	0	0.1	FP16	1 W

Table I: Computing platforms.

Net		Layers	Hyper-parameters
AlexNet	[10]	8	60M
SqueezeNet	[1]	10	0.8M
GoogLeNet	[12]	22	4M
ResNet	[13]	18	0.27M
SSD	[19]	11	-
MobileNet-SSD300	[14]	40	6.8M

Table II: Deep Neural Network Characteristics with the number of trainable layers and number of hyper-parameters ('-' indicates that the paper did not report such details).

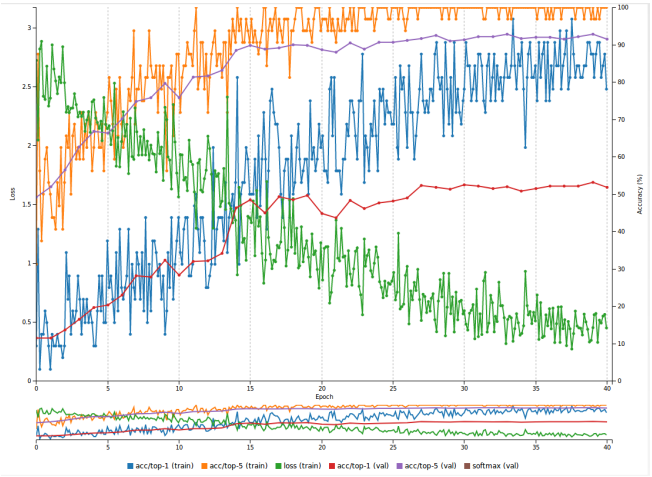

Figure 6: Accuracy graph over training iterations for ResNet18.

with the Alienware vs. 70 FPS from the Jetson and 11 FPS from the Movidius. Note, however, that different networks have higher or lower impact on the computational efficiency.

In Table V, power used by the different platforms using different networks are reported in Watt.

Second, we report results from the object detection task. Table VI shows the precision of the detection results obtained with the Jetson and Movidius, compared to the results obtained with the Alienware. Interestingly, the Movidius has comparable results as the Jetson. Note that Alienware and Jetson obtain slightly different predictions and/or difference

	Jetson	Movidius
AlexNet	0.0	0.87
SqueezeNet	0.0	0.62
GoogLeNet	0.00025	0.58
ResNet18	0.0	0.89

Table III: Top-1 error for object classification task.

	Alienware	Jetson	Movidius
AlexNet	463	70	11
SqueezeNet	400	131	34
GoogLeNet	122	29	10
ResNet18	160	46	9

Table IV: FPS of the different platforms for object classification task.

	Alienware	Jetson	Movidius
AlexNet	23.1	7.0	1.2
SqueezeNet	20.1	2.3	1.0
GoogLeNet	25.0	6.3	3.1
ResNet18	26.9	8.6	3.7

Table V: Power consumption (Watt) of the different platforms for object classification task.

in confidence, probably because of the difference in floating point precision.

	Jetson	Movidius
SSD	0.99	0.94
MobileNet-SSD300	0.90	0.90

Table VI: Precision for object detection task.

Table VII reports results on FPS for the different networks over the three different platforms. While for SSD the Alienware performs better, for MobileNet-SSD300 the Movidius is able to be comparable with the other platforms. The reason is that the optimizations introduced by MobileNet architecture is not yet efficiently implemented by Caffe.

	Alienware	Jetson	Movidius
SSD	59.8	8.0	1.5
MobileNet-SSD300	4.22	8.5	8.2

Table VII: FPS of the different platforms for object detection task.

In Table VIII, power used by the different platforms using different networks are reported in Watt.

V. Discussion

From the results above, some interesting insights can be drawn, providing some directions of work for researchers working in deep learning:

- NVIDIA platforms appear to be generally consistent in terms of prediction quality, especially in object classification.
- Movidius SDK should be enhanced to include many of the successful CNNs for robust deployment in mobile robots.

	Alienware	Jetson	Movidius
SSD	15.0	8.0	3.0
MobileNet-SSD300	7.0	3.0	2.2

Table VIII: Power consumption (Watt) of the different platforms for object detection task.

- The choice of the framework might heavily affect the performance of the network. It is usually advisable to choose the one that is suggested. A standardization of the framework might help making the field more united.
- The power consumed is generally higher than the nominal power reported in the technical sheets. As such, when analyzing the power requirements for a robot, the nominal power should be at least doubled.
- The advantage of using a Movidius is that, even if it is not as efficient as the NVIDIA counterparts, it is power-bounded by the USB specifications. As such, bigger networks will not have much effect on the power consumed, contrarily to the Jetson TX2.
- Among the networks for object classification, SqueezeNet is the one that is lightweight. Object detection is a challenging task, where networks are not very fast.
- At this point, Movidius does not support training. This limits all applications of on-line learning of the robot.

In general, the new embedded systems are promising, however, the related SDK should be improved to accommodate new operations and networks.

VI. Conclusion

In this paper, after presenting a short taxonomy of several different deep neural network architectures, we evaluated some of them on three different mobile platforms. The main focus of our work was the deploy-ability of the different networks on a variety of robotic platforms, with a special interest on robots with constrained space, such as underwater and aerial vehicles.

As part of future work, we are currently running tests for the scene reconstruction and semantic mapping applications which have different computational requirements. Furthermore, in the results presented in this paper the Caffe framework was used as it was the one more easily available on all three platforms. We are currently deploying the same networks on the three platforms utilizing the TensorFlow framework. Furthermore, we are exploring the parallelism capability of multiple Intel Movidius Neural Compute Sticks.

Acknowledgment

The authors would like to thank the National Science Foundation for its support (NSF 1513203, 1637876).

References

[1] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," *arXiv:1602.07360*, 2016.

[2] Z. Cao, T. Simon, S. E. Wei, and Y. Sheikh, "Realtime Multi-person 2D Pose Estimation Using Part Affinity Fields," in *Proc. CVPR*, July 2017, pp. 1302–1310.

[3] A. Loquercio, A. Maqueda, C. D. Blanco, and D. Scaramuzza, "DroNet: Learning to Fly by Driving," *IEEE Robot. Autom. Lett.*, 2018, accepted.

[4] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[5] G. Dudek, M. Jenkin, C. Prahacs, A. Hogue, J. Sattar, P. Giguere, A. German, H. Liu, S. Saunderson, A. Ripsman, S. Simhon, L. A. Torres-Mendez, E. Milios, P. Zhang, and I. Rekleitis, "A visually guided swimming robot," in *Proc. IROS*, 2005, pp. 1749–1754.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009, pp. 248–255.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[8] 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," *Int. Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[9] O. Beijbom, P. J. Edmunds, D. Kline, B. G. Mitchell, D. Kriegman *et al.*, "Automated annotation of coral reef survey images," in *Proc. CVPR*, 2012, pp. 1170–1177.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:409.1556*, 2014.

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

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778.

[14] A. G. 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:1704.04861*, 2017.

[15] P. Henderson and V. Ferrari, "End-to-end training of object class detectors for mean average precision," *arXiv:1607.03476*, 2016.

[16] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," *arXiv:1312.6229*, 2013.

[17] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proc. CVPR*, 2016, pp. 779–788.

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. ECCV*, 2016, pp. 21–37.

[20] A. Tao, J. Barker, and S. Sarathy, "Detectnet: Deep neural network for object detection in DIGITS," https://devblogs.nvidia.com/detectnet-deep-neural-network-object-detection-digits/, 2016, accessed on Feb 08, 2018.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. ICML*, 2015, pp. 1737–1746.

[22] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep Learning with Low Precision by Half-Wave Gaussian Quantization," in *Proc. CVPR*, 2017, pp. 5406–5414.

[23] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," *arXiv:1608.07249*, 2016.

[24] "Benchmarks for popular cnn models," https://github.com/jcjohnson/cnn-benchmarks, commit #83d441f.

[25] "Deep learning benchmark for comparing the performance of dl frameworks, gpus, and single vs half precision," https://github.com/u39kun/deep-learning-benchmark, commit #0364856.

[26] "Deepbench," https://github.com/baidu-research/DeepBench, commit #6a50a01.

[27] "Benchmarks of deep neural networks," https://github.com/doody1986/DNNMark, commit #fa12f02.

[28] "Benchmarks of deep neural networks," https://www.tensorflow.org/performance/benchmarks, accessed on Feb 08, 2018.

[29] F. Amigoni, M. Reggiani, and V. Schiaffonati, "An insightful comparison between experiments in mobile robotics and in science," *Auton. Robot.*, vol. 27, no. 4, pp. 313–325, 2009.

[30] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv:1408.5093*, 2014.