APPLICATIONS OF DEEP LEARNING IN LARGE-SCALE OBJECT DETECTION AND SEMANTIC SEGMENTATION

by

WEI XIANG

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
March 2018

To my parents, my wife, and my lovely son, for their endless love and support.

ACKNOWLEDGEMENTS

There are many people whom I would like to thank and express my sincere gratitude to during my PhD study.

First and foremost I want to give my deepest thanks to my supervising professor Dr. Vassilis Athitsos for his constant encourage and motivation. He has taught me, with great patience, a systematic way to conduct research. He always held highest standards in research and set challenging targets for every milestone during my PhD career, and was confident that I would be able to achieve those targets. Besides, he also encouraged me to think out of academia world, by which I would have those chances to understand industrial demands and apply my research outcome in real-world problems.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Fillia Makedon, Prof. Farhad Kamangar and Dr. Chris McMurrough, for their supportive encouragements and insightful comments.

I want to also thank all my colleagues from the Vision-Learning-Mining Research Lab (VLM), at CSE Department. I was fortunate to meet these creative and super nice people. My special thanks to Zhong Zhang, Christopher Conly and Pat Jangyodsuk for their promote helps during my first two years of PhD.

The rest of my thanks go to my mentors and colleagues from the Media Lab of Futurewei Technologies, NVIDIA and JD.COM, US Research Center. Particularly, I want to express my sincere thanks to Dr. Heather Yu and Dr. Dong-Qing Zhang, for their trust and providing a valuable internship opportunity at Futurewei Technologies that enriched my career path. My special thanks also go to those nice persons during my stay at Futurewei Technologies, including Zecheng He, Hang Lv and Xing Li. For my short sojourn at NVIDIA, I want to particularly thank Baris Demiroz, Gordon Grigor, and Shane Murry for their trust and supports. As for those lovely people in JD.COM, I want to express my deepest thanks to Weidong Zhang, Hongda Mao, Yan Kang, Allison Lv, Wanyan Zheng, just to name a few. They make one of the sweetest memories during my life at US.

Last but not least, I would like to give my special thanks to my family: my parents, my wife Yi Xu and my little boy Harold Ziyu Xiang. Without their faithful trust, love and support, I would never have come this far of my PhD career. It was, has been, and alway remains my greatest pleasure to have them in my life.

March, 2018

ABSTRACT

APPLICATIONS OF DEEP LEARNING IN LARGE-SCALE OBJECT DETECTION AND SEMANTIC SEGMENTATION

Wei Xiang, Ph.D.

The University of Texas at Arlington, 2018

Supervising Professor: Vassilis Athitsos

With the massive storage of multimedia data and increasing computational power of mobile devices, developing scalable computer vision applications has become the primary motivation for both research and industrial community. Among these applications, object detection and semantic segmentation are two of the most popular topics which, in addition, serve as the fundamental features for many computer vision systems under platforms like mobile, healthcare, autonomous driving, etc. Inspired by the current and foreseeable trend, this thesis focuses on developing both effective and efficient object detection and semantic segmentation models, with the large-scale, publicly available data sets sourced for various applications.

In the last several years, object detection and semantic segmentation have received large attention in the literature, and have been significantly advanced with the emergence of deep learning methods. Particularly, by applying Convolutional Neural Networks (CNNs), researchers have leveraged unsupervised features in modeling which greatly simplified the tasks of classification and regression, compared to using merely hand-crafted features in those traditional approaches.

In object detection, however, there still exist many open research problems like integrating contextual information to the existing models, the missing relationship between proposal scales and receptive field sizes for different CNNs, etc. In this thesis, we study extensively such relationship, and further demonstrate that our statistical results can be used as a guideline to design both heuristically and efficiently new detection models, with an improvement of detection accuracy particularly for small objects.

In semantic segmentation, we investigate many of the state-of-the-art methods and figure out that current research have largely focused on using complicated backbones together with some popular meta-architectures and designs which, in turn, leads to the problem of overfitting and incapability for real-time tasks. To overcome this issue, we propose Turbo Unified Network (ThunderNet), which builds on a minimum backbone followed by a pyramid pooling module and a customized, two-level lightweight decoder. Our experimental results show that ThunderNet remains one of the fastest models that are currently available, while achieving comparable accuracy to a majority of methods in the literature. We also test ThunderNet with a GPU-powered embedded platform–NVIDIA Jetson TX2, whose results indicate that ThunderNet performs sufficiently fast and accurate, thus meeting the demands for embedded system.

Finally, this thesis also surveys on the joint calibration methods for RGB-D sensor. We summarize the related work and present our quantitative evaluation results thereafter.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In this thesis, we focus on developing novel methods for large-scale computer vision problems including object detection and semantic segmentation. Meanwhile, we introduce the applications of deep learning with our proposed network architectures, on both desktop and embedded platforms. Furthermore, we also investigate the joint calibration methods for both depth and RGB sensor.

1.1 Motivation

With the emergence of deep learning, unsupervised feature learning has advanced hand-crafted feature designs on many benchmarks of traditional computer vision tasks, including object detection and semantic segmentation. Among those, how to design an effective and yet efficient neural networks still remains an open problem. In this thesis, we aim at providing several computer vision methods and techniques that have been shown to alleviate the above problem in a large-scale setting.

Firstly, we discuss about neural network design and implementation techniques with their applications to both object detection and semantic segmentation. While the similar principles can also be utilized in many other computer vision tasks, these techniques provide a both effective and efficient way to build our model. Many of the state-of-the-art models adopted these techniques and thus advanced several popular benchmarks as reported. Convolutional Neural Networks (CNNs), while being extremely popular in object detection and semantic segmentation, vary a lot regarding to their combinations of network meta-architectures and designs. To name a few, they span from the basic convolutions, to various factorization and/or stacking approaches of those operations in an efficient way, and thus to the training policies that will most likely maximize the representation power of models built.

Secondly, we show that deep learning based approaches which have already demonstrated great representation power in unsupervised feature learning, can also be guided by the inherent characteristics of their basic operations, i.e. convolutions. For tasks like object detection and semantic segmentation, the contextual information must be leveraged to achieve higher accuracies. Given a multi-scale neural network, for example, the predictions rooted from early[1] convolution layers are restricted by how large areas they see/depend on the original image space. Such areas are called receptive fields (RFs), which are also referred to as the fields of view. We explore the missing relationship between context and receptive field, and further figure out that by fusing feature maps obtained from dilated convolutions of different rates/strides, the contextual information can thereof be learned in an intuitive approach.

Thirdly, all above discussions are about using deep neural networks to solve traditional computer vision tasks. However, there exist many other fundamental vision tasks that rarely depend on unsupervised feature learning, for instance 3D reconstruction with depth sensor. Therefore, in this thesis, we also survey on 3D depth sensor calibration methods in a quantitative approach.

1.2 Our Methods

The answer to designing and implementing an effective deep neural network is not a simple one. It depends on the task we are trying to solve, the model we are

[1] We refer to *early* convolution layers as those *bottom* layers close to getting the input data.

building, the platform our models are running and the tools and techniques we are utilizing.

To achieve our goal of getting high accuracies with fair runtime performance, we fix the applications of our proposed methods onto two platforms: desktop and embedded. The former provides a flexible environment for prototyping and has been verified to be more tolerant to the parameter redundancy resulted from different network designs, while the latter provides limited computing resources and therefore demands effective but yet efficient networks. In the literature, however, there exists a large gap between prototyping networks which aims at achieving high accuracies, and designing efficient models under embedded environment. More recently, researchers have explored and advanced this issue, as we can find many of the related works, with keywords like embedded neural networks, model compression/acceleration, parallel inference, etc [1, 2].

Afterwards, we realize that utilizing the hardware computational resources to its maximum does not necessarily indicate running an efficient neural network. The neural network, by itself, is composed of various convolution operations followed by many non-trivial non-linear and combinatory methods. Therefore, we present different network architectures for object detection and semantic segmentation, respectively. The proposed architectures apply those state-of-the-art techniques not only after purely empirical experiments, but also with heuristic design patterns. We firstly study the important factors of determining a best-fit neural network among many choices, from which we show that the contextual information, while being admittedly a very crucial factor in object detection and semantic segmentation, has not been explored and analyzed extensively in many of the current network architectures. Starting from this point, we further analyze the relationship between Theoretical Receptive Fields (TRFs) and Empirical Receptive Fields (ERFs), and conclude with extensive experiments that most of the state-of-the-art networks in object detection and semantic segmentation, can be improved by taking into account accurately such relationship from [3]. Admittedly, both the demand and realization of contextual information vary between object detection and semantic segmentation, by which we will discuss more details in this thesis.

Essentially, targets in object detection and semantic segmentation have different forms, whereas predictions for the former can be divided into two parts: proposal generation and classification, while for the latter we focus on predicting in a per-pixel manner, i.e. obtaining a probabilistic map for each class whose size corresponds to the input image. In object detection, predictions can either be made after the last convolution layer in a single-scale setting, or at multiple, different layers in a multi-scale setting. This is a rough categorization, however, as we can also make predictions from single feature map which combines information at different scales [4, 5], or like some other variants [6]. In this thesis, we focus on multi-scale network design and address the problems of lacking context particularly at early prediction layers, by using dilated convolutions and fusing the feature maps into one for future predictions. Afterwards, we validate the effectiveness of proposed method with different settings of hierarchical CNNs like VGGNet [7].

Regarding to the semantic segmentation, though it seems very easy to generate pixel-wise predictions via CNN, the difficult part lies in how to combine multi-level contextual information on pooled feature maps which most likely have lost many detailed pixel-wise correlations. That being said, it remains a challenging task in semantic segmentation, to keep accurately both geometry shapes and boundary details. In this thesis, we propose an extremely lightweight network that achieves comparable accuracy, but yet runs significantly faster than the methods currently available. We also show with our experiments on a GPU-powered embedded platform that, our proposed network meets the demand of fast segmentation under an embedded platform with much better results than the fastest network so far [8].

Finally, in this thesis, we present a survey with quantitative evaluations about 3D depth sensor calibration methods, where the topic of joint calibration has been found practically useful in many computer vision systems of gesture recognition, robotics, etc.

1.3 Thesis Overview

We briefly provide an overview of this thesis: In Chapter 2 we introduce deep learning neural networks, and talk about several popular tools, technologies and topics in building convolutional neural networks. In Chapter 3, we analyze the empirical receptive fields, and examine the context aggregations in our proposed network architecture for object detection. In Chapter 4, we study the current trend of research in semantic segmentation, and propose an effective and efficient network architecture for real-time applications. In Chapter 5, we survey on the 3D depth sensor calibration methods.

Finally, in Chapter 6 we draw our conclusions of this thesis. We summarize the contributions of our proposed methods for large-scale object detection and semantic segmentation.

CHAPTER 2

DEEP NEURAL NETWORKS

This chapter introduces deep neural networks, with focus on the convolutional neural network and its recent advances in large-scale image classification. We also discuss some other relevant topics and implementation details, which cover but should not be limited to, deep learning development frameworks, GPU acceleration, model compression, etc. From these discussions and studies, we show that deep learning is a rapid evolving approach in the field of machine learning, and it still requires ongoing efforts from both research community and industrial practitioners.

2.1 Introduction

Deep *convolutional neural networks* (CNNs, also referred to as ConvNets) [9] have made significant advances in the fields of image classification [10], object detection [11, 12, 13, 14], image segmentation [15, 16, 17], pose estimation [18] and many other classification and regression based tasks. CNNs have enabled a large number of flexible, data-driven methods for object recognition, detection and segmentation. Besides, deep learning has already shown some promising results for even more challenging vision tasks, like multilabel classification and co-localization, where we aim either to recognize multiple classes in a single image, or to detect multiple different instances of the same class in one image. Furthermore, by utilizing the computational power of GPUs, the state-of-the-art object detection frameworks can now achieve near real-time rates with very large networks [19].

Before CNNs advanced previous methods by a large margin on ImageNet [20] in 2012, which nowadays has become a standard testing benchmark for large-scale object recognition, CIFAR-10 [21] was extensively challenged: In 2010, the best result was 77% by Adam Coates [22]. Lately, given the rank lists in [23], after four years Graham [24] presented a state-of-the-art result of 96.53%. The results look quite impressive, however, it is not merely about getting higher numbers. As Karpathy reported in [25] that the human accuracy is approximately 94% for CIFAR-10! That being said, as deep learning proceeds on its fruitful road, it also makes breakthroughs addressing many visual tasks, including object recognition, scene understanding, etc, where in some fields its performance is even beyond human strength.

Back to the ImageNet ILSVRC challenges (see Fig. 2.1 for the best results from 2012 to 2016), Microsoft announced their results in 2015 about ImageNet's top-5 classification error of 3.57%. This number has already surpassed the human accuracy, which is approximately 5.1% (tested by Karpathy in [26]. It should also be noted that ImageNet is much more challenging comparing to CIFAR-10 with respect to the number of categories (1000 vs. 10) and input sizes. Therefore, given the support from GPU clusters, such a result proves again that deep learning methods are capable to handle large-scale vision tasks, which is even better than we humans do. However, we quote from [26] that "Human accuracy is not a point. It lives on a tradeoff curve." The human accuracy was obtained by someone who was not trained extensively (the hypothetical expert human ensemble of labelers will match the performance down to 3%). From this perspective, it is not fair to compare human accuracy with deep learning methods. But we humans are proud to say that our brain do not take that huge amount of energy consumption to learn and infer new objects than running costly deep learning programs.

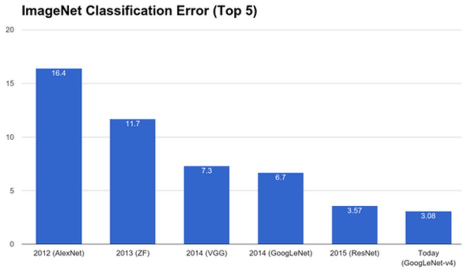


Figure 2.1: State-of-the-art methods in ImageNet benchmark till 2016. Source by [27].

In this thesis, we study particularly convolutional neural networks which include the most popular backbones *VGGNet* [7] and *ResNet* [28], detectors *Faster R-CNN* [29] and *SSD* [30], image semantic segmentation networks *FCN* [15] and *PSPNet* [31], etc. It is noteworthy that while some topics in the deep learning research still remain crucial, for e.g. Recurrent Neural Networks (RNNs), Auto-Encoder (AE), Reinforcement Learning (RL), Generative Adversarial Network (GAN), etc., they may be mentioned in this thesis but not brought into very details, albeit their significantly important applications in natural language processing (NLP), speech recognition and many others.

2.2 Tools

As deep learning becomes extremely popular in these years, many frameworks and toolkits have been developed, bridging between the research community and industrial applications. Gulcehre [32] provides a list of software links to the publicly available deep learning toolkits (with a total number of 46 softwares found so far). Among them, there are five most popular deep learning toolkits for the general learning tasks, and those have also been actively maintained, including *Caffe*/*Caffe2* [33], *CNTK* [34], *Theano* [35], *Torch*/*PyTorch* [36] and *TensorFlow* [37].

	Creator	Written in	Platform	Multi-GPU Support	Pros	Cons
Caffe/Caffe2	Berkeley Vision and Learning Center	C++, Python	Linux, Mac OS, Windows, AWS	Yes(6)	1. First industry-grade DL toolkit, most popular in CV. 2. Plenty pre-trained models available in Caffe Model Zoo [38]. 3. Excellent model deployment capability (cross-platform).	1. Poor support for RNNs and language modeling. 2. Tedious labor needed to construct deep net. 3. Poor interface design.
CNTK	Microsoft	C++	Linux, Windows	Yes	1. Better-known in speech community. 2. Easy to invent new layers.(1) 3. Integrated to Visual Studio. 4. Excellent model deployment capability (except ARM).	1. Hard to debug.(2) 2. Not usable for tasks like seq2seq [39]. 3. Poor interface design.
Theano	University of Montreal	Python	Cross-platform	No (Needs work-around [40])	1. Easy to invent new layers.(1) 2. Higher-level frameworks (e.g. Blocks, Keras, Lasagne, etc.) exist for fast prototyping. 3. Excellent for RNNs definition.(4) 4. Lack of low-level interface, poor model deployment capability.	1. Hard to debug.(2) 2. Difficult tensor-level implementation. 3. Compilation of large networks is time-consuming.
Torch/PyTorch	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet	C, Lua/ Python	Linux, Mac OS, Android, iOS	Yes	1. Excellent for conv net. (native support for 3D conv). 2. Rich set of RNNs extensions available [41]. 3. Excellent for new network definition.(5) 4. Amazingly fast (due to running on LuaJIT).	1. New layer definition is relatively hard.(5) 2. Hard to integrate into a large production pipline. 3. Lua is still not accepted in industry. (at least not as a mainstream language). 4. Too few extensions.
TensorFlow	Google Brain	C++, Python	Cross-platform	Yes	1. Excellent for new network definition. 2. Easy implementation for RNNs. 3. Excellent graph visualization with TensorBoard.	1. Hard to debug.(2) 2. Static Computational flow.(3) 3. Lack of native support for 3D/temporal conv.
Note	(1) A network is specified as a symbolic graph of vector/matrix operations, such as matrix add/multiply or convolution. By using fine-grained vector operations as building blocks allows for easy invention of new, complex layers, which are often hard to implement in a low-level language (e.g. C++ in Caffe). (2) Error messages from symbolic compiler used in TensorFlow/Theano/CNTK is notoriously hard to track, compared to the traditional compilers. (3) Indicating that it is difficult to apply Beam Search [39] for decoding. Fortunately, there exist some workarounds [42]. (4) Theano's loop control API (so-called *Scan* [43]) can be utilized for easy and efficient implementation of RNNs. (5) The difference between new layer definition and new network definition is minimal but non-trivial. Torch is excellent for new network definition without having to program in a low-level language, while it can define a new layer in an intuitive way, i.e., the users will have to implement the full forward, backward and gradient update. (6) As of the time this chapter is written, python interface of caffe (pycaffe) does not support multi-GPU yet.					

Table 2.1: Comparisons of five most popular deep learning toolkits.

We give a brief comparison and evaluation in Table 2.1. In this table, evaluation was done based on the newest online sources [44][45], as well as our extensive experiments. All toolkits were evaluated from a user's perspective, meaning that for each toolkit, factors including architecture, code maintenance and future roadmap were not taken into consideration. Besides, the training speed varies a lot by the factors of machine configurations (for e.g. throughputs of data transfer), library version of

BLAS, CUDA, cuDNN supported and configured by the toolkit, and the optimization degree of client's code. Also, due to the lack of hardwares, their performance of data-parallelism and model-parallelism with multi-GPUs were not evaluated. However, given a sufficiently large dataset, parallelism remains a critical factor in both design and implementation of the deep neural networks. In summary, we mainly investigated on each toolkit into the supported platforms, multi-GPU support, low-level built-on language(s), language interfaces, training speed, visibility for tracking debugging information, deployment capability and modeling capabilities including new layer and network definition, extensions, etc. After carefully reviewing Table 2.1, we draw a conclusion that:

- Given a dataset that shares many similar characteristics with one of the many public datasets, and also suppose that we just want to quickly get results with the state-of-the-art models, Caffe/Caffe2 will be our first bet due to its plug and play nature (inside Caffe/Caffe2 it uses Google Protocol Buffers, with which you do not have to literally write any code), along with a large set of publicly available pre-trained models.
- When dealing with RNNs, we can choose among toolkits that were built on symbolic compilers including Theano, CNTK and TensorFlow.
- When researching the newest CNN models, we prefer using Torch/PyTorch since it has the greatest features for researchers such as quickly prototyping new layers and networks, while the Torch/PyTorch toolkit has been actively maintained. In addition, as a great add-on for large datasets, Torch/PyTorch has been embedded with very convenient APIs for data-parallelism. Last, compared to other toolkits that have been practically found very difficult to debug, the output of Torch shows clear hints to the hidden bugs which helps easing our work a lot when programming.

- Regarding to the software deployment, both Caffe and CNTK are able to integrate their code into a large production pipline without much headache. However, they also came with worse interface design, and sometimes it leads to the effort-consuming development for new models. In Caffe/Caffe2, for e.g. you will have to specifically write C++ and CUDA code respectively to implement a new layer on GPU. Such implementation has been found error-prone due to that the data flow in GPU layers are often harder to track and debug compared to its counterpart in CPU layers (not to mention the utilization of Prototype Buffer makes it even more cumbersome for very large networks such as GoogLeNet and ResNet).

Note that our evaluation in this section may not be accurate and up-to-date, because all frameworks are still developing rapidly and their roadmaps are difficult to predict and track. At the time of this writing, there are many ongoing projects that have drawn great attention especially from industrial practitioners, including Mxnet [46], PaddlePaddle [47], ncnn [48], etc.

2.3 GPU Acceleration

2.3.1 Principles

With GPU becoming more of a commodity in the deep learning field, computationally intensive methods, such as convolutions, Fourier Transforms and other matrix-based linear algebra operations, opt to be addressed in GPU clusters with massive amounts of training data. Normally, within the code framework of deep learning, both input batch and model parameters need to be converted into GPU arrays and fed into the CUDA framework. We expect the code that results in computationally intensive operations being executed in a pure GPU environment. Taking Fig. 2.2 as an example, compute-intensive functions that make up about 5% of code run on the GPU.

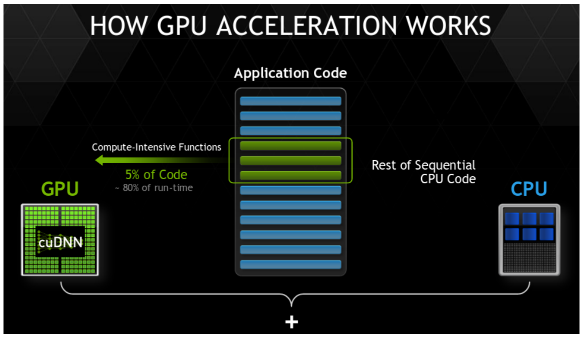
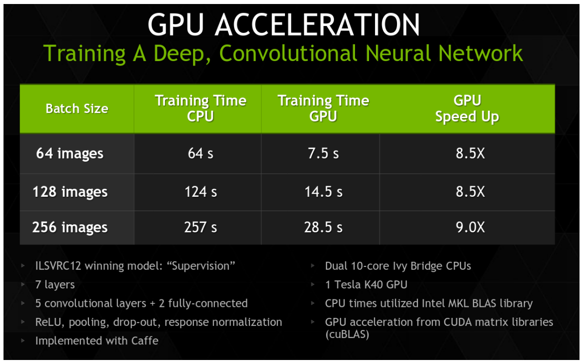

HOW GPU ACCELERATION WORKS
Application Code
Compute-Intensive Functions
5% of Code
~ 80% of run-time
Rest of Sequential CPU Code
GPU
cuDNN
CPU
+

Figure 2.2: Illustration of how GPU acceleration works in space of the application code. Source by: http://www.nvidia.com/content/events/geoInt2015/LBrown_DL.pdf.

2.3.2 High-performance Computing Libraries



GPU ACCELERATION

Training A Deep, Convolutional Neural Network

Batch Size	Training Time CPU	Training Time GPU	GPU Speed Up
64 images	64 s	7.5 s	8.5X
128 images	124 s	14.5 s	8.5X
256 images	257 s	28.5 s	9.0X

- ILSVRC12 winning model: “Supervision”
- 7 layers
- 5 convolutional layers + 2 fully-connected
- ReLU, pooling, drop-out, response normalization
- Implemented with Caffe

- Dual 10-core Ivy Bridge CPUs
- 1 Tesla K40 GPU
- CPU times utilized Intel MKL BLAS library
- GPU acceleration from CUDA matrix libraries (cuBLAS)

Figure 2.3: GPU acceleration compared to its CPU counterpart using Caffe. Source by: http://www.nvidia.com/content/events/geoInt2015/LBrown_DL.pdf.

Fig. 2.3 demonstrates how GPU greatly boosts training performance. To get further details of its performance compared to other CPU-ready high-performance computing (HPC) libraries such as ATLAS[1], OpenBLAS[2] and Intel MKL[3], we empirically benchmarked their performance using Caffe and Theano. The results are shown in Fig. 2.4 and Fig. 2.5 respectively.

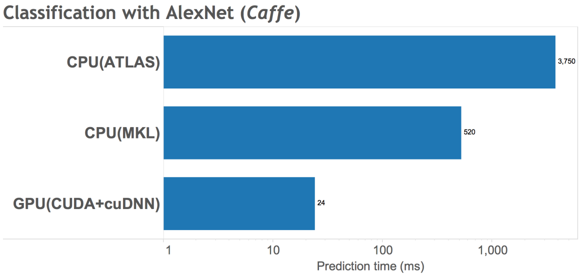


Figure 2.4: Comparison of classification time (in ms) using different high performance computing libraries. The test was done in Caffe, using AlexNet with input of a batch of images (50 in total, each of size 227 × 227).

Fig. 2.4 gives a comparison of prediction time using AlexNet to classify 50 images, given different HPC libraries including ATLAS, MKL and CUDA+cuDNN. Obviously, the combination of CUDA and cuDNN outperforms ATLAS with a factor of 73, while accelerating MKL with a factor of 22. It also shows that when running on CPUs, MKL achieves the best performance compared to other open source Basic Linear Algebra Subprograms (BLAS) libraries, while it is still much slower than CUDA.

In Fig. 2.5, We also tested their performance on matrix multiplication, for both 32-bit and 64-bit floats (namely FP32 and FP64). Still, matrix multiplication using CUDA had the least computational time. Regarding to the experiments on FP32, it outperform MKL and OpenBLAS by a factor of 77 and 170, respectively. There is an interesting finding in Fig. 2.5 that if we take a side-by-side view to their performance

[1] `http://math-atlas.sourceforge.net/`

[2] `http://www.openblas.net/`

[3] `https://software.intel.com/en-us/intel-mkl`

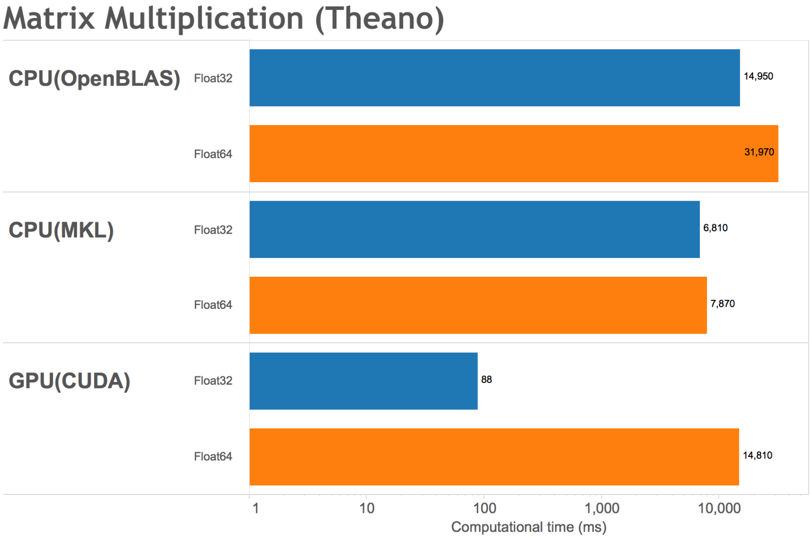


Figure 2.5: Comparison of computational time (in ms) using different high performance computing libraries, for 32-bit and 64-bit floats respectively. The test was done in Theano, with 10 times of matrix multiplication $\boldsymbol{A} \times \boldsymbol{B}$, each of size 5000×5000.

on both FP32 and FP64, we realize that as for CUDA, the performance downgrades 168X with FP64 than FP32. While running MKL with FP64 incurs the minimal penalty, performance of OpenBLAS also has downgraded 2X. We used K40 for this test, but according to [49]: “K40 is given a special double precision processing unit, and its performance of FP32:FP64 should not exceed 1:3”. In light of this finding, we assume that Theano does not have an expected optimization at its core for the FP64 computation. The best bet is to avoid FP64 computations as best as we can when the performance downgrades heavily with FP64. However, FP64 computations are still very helpful, when running other applications like physics modeling/simulation, high accuracy financial computations, etc. that require high precision results.

2.3.3 Data Parallelism vs. Model Parallelism

Data parallelism (also called “replicated training” in Tensorflow) is when running the same model across all GPUs, we feed each GPU different parts of the data (see Fig. 2.6). The basic idea of data parallelism is simply, however, the dilemma we have to face is that data parallelism does not scale with respect to the number of GPUs, which is true especially during the backpropagation.

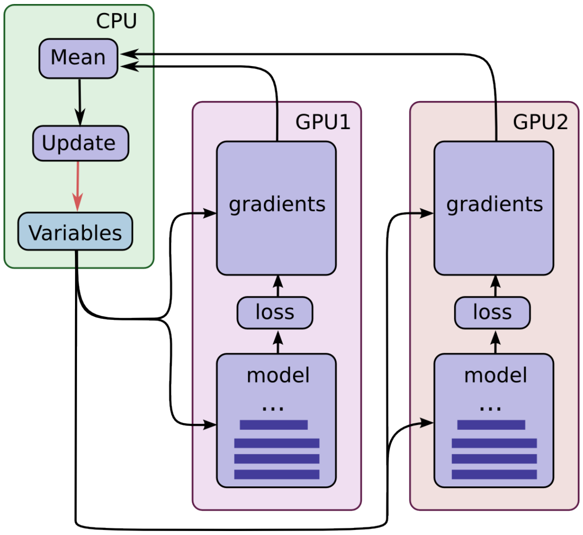


Figure 2.6: Illustration of synchronous data parallelism on a local configuration. Notably, gradients calculated from GPU1 and GPU2 individually are averaged in order to get the optimal training solution. Source by `https://www.tensorflow.org/versions/r0.7/images/Parallelism.png`.

For e.g., given four K40s, and suppose that we have a 1000×1000 matrix (of float type, definitely), meaning that we need $\frac{1000 \times 1000 \times 4}{2^{30}} = 0.00093\text{GB}$ to store this matrix, and it takes $0.00093\text{GB}/15.75\text{GB/sec} \times 1000\text{ms/s} = 0.059\text{ms}$ to transfer this matrix from one GPU to another through PCIe3.0 X16, where 15.75 GB/sec is the throughput of PCIe3.0 X16. Note that here, we neglect additional overhead of direct GPU-to-GPU data transfer happening in data parallelism. Instead, to upload data from GPU0 to GPU1, it has to go through CPU0 with the Intel QuickPath Interconnect (QPI) link whose speed is around 25GB/sec. Obviously, data transfer between GPUs is bottlenecked on the PCI lane anyway. Given four GPUs, it is necessary to synchronize this matrix three times across all GPUs, which in total takes $0.059 \times 3 = 0.177$ms. Besides, on K40, it takes only about 1 ms (or even less, depending on the number of nodes in previous hidden layer) to forward the layer by calculating $\boldsymbol{A} \times \boldsymbol{B}$, with each matrix of size 1000×1000. Compared to the computational time of forward/backward matrix multiplication[4], data synchronization during backpropagation takes around 20% of its time. However, this is ideal (and is only applicable on single PC) cause we have not taken into account the cost of data transfer via network card (which is often up to 40GB/sec and can only be used by one GPU at any time) from one node to another. Therefore, if we are running the model with data parallelism on a cluster, where data synchronization between nodes can only be done via network card, things become worse due to that the additional data synchronization time during backpropagation has become a bottleneck in data parallelism (even 100X cost than a forward matrix multiplication according to [50]). A great tutorial on data parallelism using Tensorflow can be found at [51], where it also explained clearly about the difference between asynchronous and synchronous data parallelism (Fig. 2.6): "Naively employing asynchronous updates of model parameters leads to sub-optimal training performance because an individual model replica might be trained on a stale copy of the model parameters. Conversely, employing fully synchronous updates will be as slow as the slowest model replica. [51]" In most cases, running two different jobs (models, specifically) on the same GPU drastically slows down GPU training. It is even much slower than training a single model on one GPU at one time. However, model parallelism indicates that running multiple models (or split models) simultaneously on the same number of GPUs with the same input to each GPU.

[4] Backpropagation does twice matrix multiplications, see the equations of backpropagation.

A typical example is that given an extremely large weight matrix that can hardly be held in the memory of a single GPU[5], splitting a weight matrix either horizontally or vertically, and then concatenate or stack the calculated matrices into a big one and finally return the result. It is obvious that we must guarantee the data are exactly the same to all splitting models across GPUs in order to get the correct answer. Details about model parallelism can be found at [52].

It it worth noting that NVIDIA SLI technology can sometimes be misclassified as a data parallelism solver for deep learning. In fact, SLI only scales graphics performance, and is not used within the CUDA programming. Since each GPU is individually addressable in a CUDA application, adding a second GPU allows for the distribution of workload freely with respect to both data and model parallelism. Aside from SLI, an exemplary framework that can be utilized in deep neural network modeling is the NVIDIA's Deep Learning GPU Training System (DIGITS)[6], which powers both Caffe and Torch to a great extent.

2.4 Deep vs. Shallow

Why do we prefer a deep model than a shallow one? Since VGG [7] was firstly introduced in 2014 with 19 layers (compared to 7 layers used in AlexNet [10]), researchers tend to design deeper networks with small-sized conv layers: GoogLeNet v1 [53] is constructed with 100 layers. The winner in ILSVRC 2015 challenge (ResNet [28] by Microsoft) proposed a network composed of 110 layers, and they even attempted the deepest networks of up to 1202 layers! Slide in `http://cs231n.stanford.edu/slides/2016/winter1516_lecture11.pdf` elucidated clearly on why we should pre-

[5] It is arguable that we can never have such a large weight matrix (i.e. more than 12GB), but such problem indeed exists in some unsupervised learning tasks.

[6] `https://developer.nvidia.com/digits`

fer using small filter instead of large one. Considering a conv layer with 3 × 3 small-sized filter in Fig. 2.7, it is easy to observe that one neuron in the second conv layer maps to a 5 × 5 region in the input layer (i.e., its receptive field). Thereafter, at the third conv layer, its receptive filed will be a 7×7 region. That being said, with respect to the size of receptive field, a single conv layer with 7 × 7 filter will be tantamount to a stack of three conv layers with 3 × 3 filters.

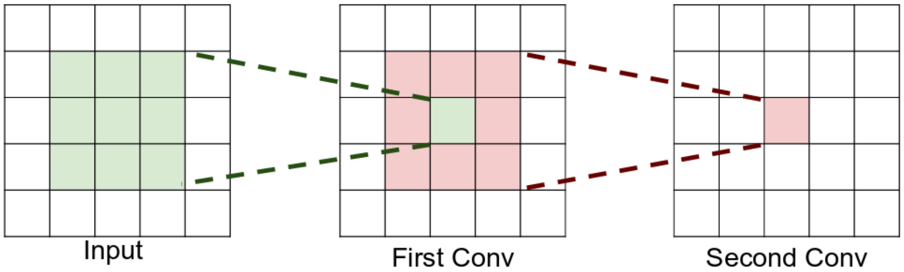
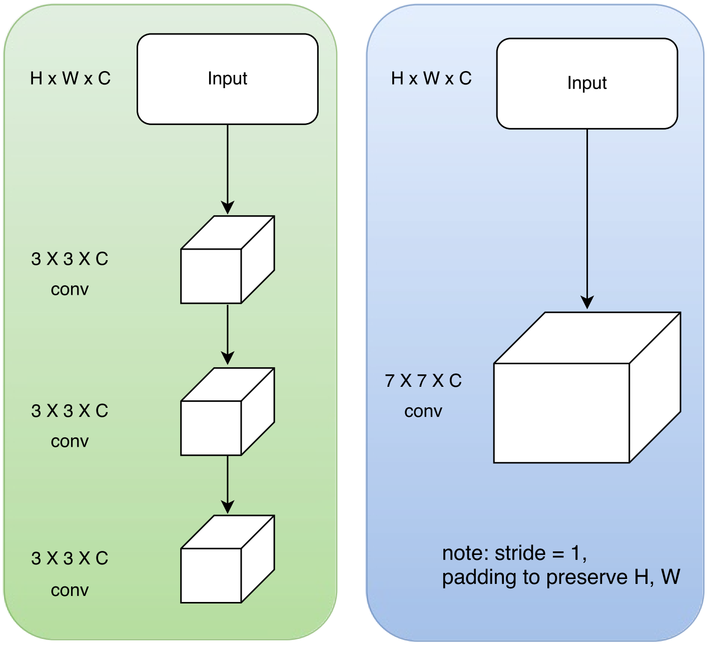


Figure 2.7: A demonstration of how conv layer works by mapping one neuron to its previous layer. In this example, two conv layers follow the input layer. At first conv layer, one neuron maps to a 3 × 3 region in input, while at second conv layer, the mapped region is enlarged to 5 × 5. Source by `http://cs231n.stanford.edu/slides/2016/winter1516_lecture11.pdf`.

As shown in Fig. 2.8, stacking more conv layers with smaller filter has less number of weights to store, as well as less number of multi-adds to compute than using less conv layers with larger filter. Since non-linearity is what we want to achieve in the neural networks, we should always avoid using conv layers with very large filter, for it downgrades heavily the model's capability to capture further details. But in order to get a computationally feasible model (regarding to both storage and training time), most likely we design the fist conv layer with a large filter[7].

[7] It also depends on the size of input image, a 'brute' filter of size 10 × 10 should never be used for a dataset like CIFAR-10, however, it is often utilized for ImageNet.



# of weights	3X [(3 x 3 X C) X C] = 27 C^2	<	(7 X 7 X C) X C = 49 C^2
# of multi-adds	3X [(H x W X C) X (3 X 3 X C)] = 27 H X W X C^2	<	(H x W X C) X (7 X 7 X C) = 49 H X W X C^2
Non-linearity	X3	>	X1

Figure 2.8: A comparison study of using smaller filter $(3 \times 3 \times C)$ with 3 conv layers, versus using larger filter $(7 \times 7 \times C)$ but with a single conv layer. Both schemes have the same-sized receptive field $(7 \times 7 \times C)$ for a single neuron in the last conv layer, however, the former one achieves much more non-linearity in addition to less computation and storage cost compared the latter one.

So, how about 1×1 conv layer? Will it improve the performance further? Using an individual 1×1 con layer theoretically only learns a linear combination of input channels at its output. However, State-of-the-art methods [54, 53, 28] utilize a technique in their design of networks by adding a 1×1 conv layer before a 3 conv layer, and thus the *embeddings* can be achieved after a 1×1 conv layer with dimension reduction. Besides, the non-linearity can be further strengthened. Furthermore, 1×1 conv layers can also be positioned right after max pooling layer and thus play a role as rectified linear activation, as depicted in Fig. 2.9. Szegedy et al. in their work [55] fully discovered the possibilities of using such technique named *bottleneck module*, which has been successfully utilized in the newer GoogLeNet models (since 2014).

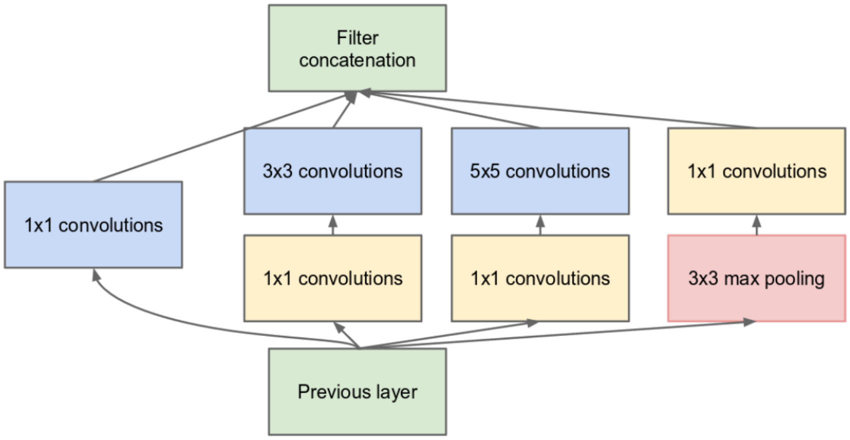


Figure 2.9: An example of Inception module. Using the 1×1 conv layer achieves the dual-purpose of 1) reducing dimensionality (i.e., embeddings) before a small-sized conv layer, 2) rectifying linear activations after a max pooling layer. Source by [53].

Then, how do we define *deep* in deep neural network? Schmidhuber [56] in his review gives a subjective definition that "Instead of committing myself to a precise answer, let me just define for the purposes of this overview: problems of depth > 10 require *Very Deep Learning.*" As for the precise definition of depth, it is still a controversial topic in academia. Schmidhuber firstly introduced in his work [56] a new concept named the Credit Assignment Paths (CAPs), which models the chains of possible causal links between two events. The author further introduces the CAP's depth that follows the path composed of layers with modifiable weights[8]. *Problem*

[8] By doing so, CAP's path defines how far the backwards credit assignment can move down to a causal chain and thus find a modifiable weight [56].

depth and *solution depth* were firstly introduced to distinguish between depth of the deepest CAP and smallest CAP.

Compared to the acyclic feedforward neural networks, *Recurrent Neural Networks* (RNNs) have much larger depth because of its cyclic nature. In this sense, RNNs and its variants are the deepest among all.

Notably, going deeper does not necessarily indicates learning better. Ba et al. [57] figured out in an empirical approach that shallow models can achieve the same performance than deep models, by using a novel technique called *model compression*. The basic idea is to pass the large, unlabeled data into deep model (assumed to be larger and more accurate than the mimic one), and then collect and feed the scores into a much smaller, mimic model. Therefore, since the mimic model is not trained with the original labels, the problem now lies in how to design an effective and efficient algorithm leading the mimic model to learn the function that was learned by the larger model. For anyone interested in the relevant work, more details can be found at [57]. In addition, given a long temporal window (often seen at some long video sequences), shallow models are more capable of capturing the event evolution than deep models. Our empirical experiments suggest that when designing a new neural network model, at most times, the model does not have to fully capture the finest-grained details.

2.5 Spatial, Scaling and Rotation Invariance

When we develop a pipeline for certain computer vision system as in a traditional approach, it becomes crucial that the delicately designed features are invariant to spatial, scaling and rotation changes. Similarly, this general principle applies to the deep learning networks, but in an implicit manner.

One intriguing property of CNNs is their learning capability (to some extent) for spatially invariant features by introducing spatial max-pooling layers. However, the spatial invariance is only attainable after using a very deep network, so that the receptive fields of neurons cover the whole target image. To this end, some work [58, 59, 60, 61, 62, 63, 64] studied CNN representations in the hope that, the model is capable to learn input image transformations including spatial translation, rotation, etc. Among them, *hard attention*-based models [61] modulate responses of neuronal activations for a subsequent forward pass of input, in order to *attend* to specific convolutional feature maps. However, it is nearly impossible to perform gradient descent in hard attention-based methods due to largely entangled networks, and hence workarounds are required, for e.g. using reinforcement learning [61]. As its counterpart, *soft attention*-based models [63] group all locations in input image into fixed grids and come up with nice derivative solutions.

Nonetheless, grid generation in soft attention-based methods constrains the appearance of attentions to fixed grid positions and thus is not able to *attend* to arbitrary regions within the input image. To overcome this issue, Jaderberg et al. [62] proposed spatial transformer networks (STN) [62]: "Unlike max-pooling layers, where the receptive fields are fixed and local, STN can spatially transform an image or its feature maps by learning an affine transformation matrix capable of producing scaling, cropping, rotations, as well as non-rigid deformations." Similarly, a very recent work of [64] introduced four basic operations as layers in the network, in order to build CNNs that are partially or fully rotation equivariant.

2.6 Model Compression

Model compression aims to reduce the number of parameters given a deep neural network, while achieving comparable accuracies against the original model. Due to that many researchers have realized the gap between prototyping deep neural networks on a desktop-level machine and deploying them onto various embedded platforms, model compression has become a very hot topic in these two years. This section surveys and summarizes recent model compression methods in the literature, whose main ideas can be used as general principles in designing our models for many computer vision tasks. For example, the proposal of ThunderNet (Chapter 4) used for semantic segmentation, was inspired by several model compression technologies mentioned in this section.

2.6.1 Related Work

One of the pioneering work about model compression proposed by Han et al. [65] compressed AlexNet by a factor of 9 using network pruning. Later, Han et al. again proposed a compression framework called *Deep Compression* [1] which compressed the model size further down to X35, by extending their previous work using a combination of weight pruning, network quantization and Huffman coding. Both methods achieve the similar accuracy to original AlexNet. In 2016, not only Han et al. [66] successfully implement Deep Compression in hardware and filled the gap of model compression between the desktop and embedded systems, but also *SqueezeNet* was proposed [2] as a new architecture which followed several critical design principles. The compression rate of SqueezeNet was reported to be X50. Till today, SqueezeNet has become quite popular mainly because we can apply various engineered techniques within its delicately designed architecture and compress further our models. As reported by [2], applying Deep Compression on SqueezeNet with 6-bit float resulted in a compression rate of 510! However, high compression rate does not indicate large speedup factors. *XNOR-Net* [67] was proposed and claimed that their compression rate is X34. XNOR-Net introduced a novel technique to binarize weights and activations, so that conv operation which consists of multiplications plus adds, can be implemented with pure adds, and even with the bit shifts! Notably, the gradients in XNOR-Net still need to be in full precision. Afterwards, DoReFa-Net [68] further exploited the network binarization problem, and was capable of using arbitrary lower-bit data type (bandwidth, they call) in all parameters in network, including weights, activations and gradients. According to [68], with an extreme setting of using 1-bit weights, 1-bit activations and 6-bit gradients, the top-1 accuracy on ImageNet (ILSVRC-2012) can achieve 46.1%, compared to 57.1% for the original AlexNet.

As shown in Table. 2.2 and 2.3, state-of-the-art models we have surveyed include Deep Compression [1], SqueezeNet [2], XNOR-Net [67] and DoReFa-Net [68].

Model	Open Source	Platform	Performance on AlexNet		
			Accuracy (Top-5)	Speed-up (GPU)	Storage (DRAM)
SqueezeNet [2]	No	Caffe(M)	80.3%	N/A	X510[9]
Deep Compression [1]	No[10]	Caffe(M)	80.3%	X4	X35
XNOR-Net [67]	Yes [69]	Torch(MC)[11]	80.2%	X58[12]	X34
DoReFa-Net [68]	Yes [70]	TensorFlow(MC)	N/A[13]	N/A	X32

Table 2.2: Comparisons of four state-of-the-art model compression methods–Part1. M = Model, C = Code. The accuracy was tested on ImageNet ILSVRC 2012 challenge.

Note that in Table. 2.2, the compression rates for all methods can not be compared directly, since the numbers were obtained either from different measurement

[9] However, it is the result from 6-bit, which is very hard to implement. For 32-bit, the factor drops down to X50.

[10] Ruiwei has implemented the first step of this model.

[11] XNOR-Net has been implemented in both TensorFlow and Torch, however, its implementation under TensorFlow is slow and incomplete. See discussions at `https://github.com/tensorflow/tensorflow/issues/1592`.

[12] A theoretical factor inferred regarding to one conv operation excluding time of memory allocation and memory access, and for small filter size only. The overall performance in AlexNet is not reported.

[13] Top-1 accuracy is reported as 46.1%, compared to 57.1% for the original AlexNet (BLVC).

approaches (empirically vs. theoretically), or from different targets (conv vs. FC vs. conv + FC, etc.).

Model	Pros	Cons
SqueezeNet	1. Both micro- and macro-architecture are taken into account by following some design principles. 2. Capable of wrapping existing engineered, compression techniques (like Deep Compression) in the networks. 3. No/Replaced FC layers.	1. Bottlenecked architectures can not be easily extended or combined with other meta-architectures. 2. Extensive use of bottlenecks will most likely lead to underfitting.
Deep Compression	1. Both conv and FC layers are highly compressed. 2. Has already been implemented in hardware [66]. 3. Very complete benchmark.	1. Can't process batch input. 2. Not robust to new models and datasets. Parameters were engineered after a great number of experiments. 3. Performance looks nice, but it is difficult to have a fine-tuned implementation since no public code is available and the work seems too *engineered*.
XNOR-Net	1. Works extremely well on large dataset. 2. Approximates normal conv with bit conv. 3. No/Replaced FC layers.	1. Compression rate is low given small filters and few # of channels. 2. FC layers need to be replaced with conv layers in order to be fully compressed. 3. Gradients are still in full precision, where most training time are spent in backward pass. 4. 1-bit conv layer is not implemented using bit-shifts, they still used full precision operation in their code.
DoReFa-Net	1. Low-bit approximations available on all parameters: weights, activations and gradients.	1. Configuration space of bitwidth for weights, activations and gradient need to be tuned for a specific application. 2. Accuracy decreases a lot when compressing the model to an extreme setting.

Table 2.3: Comparisons of four state-of-the-art model compression methods–Part2.

2.6.2 Evaluation

There are only a few works in the literature that provide a complete benchmark regarding to the performance of compression methods. We think model compression methods need to be evaluated in terms of:

- **Accuracy**: Typically, we use classification accuracy for CIFAR-10 and MNIST, as well as top-1 and top-5 classification error for ImageNet (ILSVRC-2012 challenge).

- **Test models**: In most work, researchers opt to test on LeNet, AlexNet, VGG, ResNet and GoogLeNet, or subset of them. However, one problem is that all adhoc parameters must have been fine-tuned given all those existing networks (like in Deep Compression, they have many such parameters). Once we are given a completely new network, we need to run a great number of experiments, in order to fully compress the model while maintaining the same accuracy (which seems a very difficult task).
- **Model size**: number of parameters we need to store on chip. Regarding to the parameters in neural networks, mainly we talk about weights, activations and gradients.
  - *Weights*: These are the *parameters for network*, which decides the learning capability of our model. As for model compression, we want to prune weights and count the number of effective weights instead, cause in most time our models are over-parameterized that results in over-fitting. See Sect. 2.6.6 for a relevant discussion.
  - *Activations*: During training we need to store activations for each layer, cause they will be needed in back propagation. However, at test time, those activations can be reduced by a huge amount, by only storing the activations at current layer, and getting rid of the activations of the layers below, as the network infers. Since most of the memory is usually consumed by the activations, finding a lower-bit representation for activations has been utilized in many methods. In addition, when we chose optimization momentum, Adagrad, or RMSProp, cache needs to be stored. Therefore, the memory to store the activations alone must usually be multiplied by a factor of at least 3 or so (the factor differs a lot for the compressed models).

  - *Gradients*: Memory required in back propagation is about 2 times compared to forward pass, it is because we need double computations to update the gradients and keep the calculated numbers. The problem becomes worse if we have an extremely long Recurrent Neural Network (RNN). However, improvement is possible, see the newest work at [71].

  Usually, at bottom layers (especially the first conv layer) we need to store more activations, while at top layers (especially the first fully connected (FC) layer right after the last conv layer) the model will learn many more weights. Inevitably, one CNN implementation has to maintain memory cache for miscellaneous uses, such as image data batches, perhaps with their augmented copies, etc. in order to feed into the network. Regarding to the optimization of suc, we still have a lot of space to discover regarding to the model compression within data parallelism and model parallelism. Another thing we should pay attention to is that we should distinguish between *training model size* vs. *test model size*. Model size referred to in most work, is essentially the training model size. However, the test model size is also important as we want to know how many space needed for our system purely in testing phrase, when only weights are stored during inference.

- **Running time**: Most of the existing work do not evaluate running performance for their compressed models compared to their counterparts. Other than that, both training and inference time need to be tested for a compressed model. Some non-trivial questions we should take into account include:

  - *No-batch vs. batch*: We should also test the running time both with single image and batch of images. For some models that show no advantages in batch processing (like Deep Compression [1]), its drawback is very obvious (which shows no speed-up over original models on CPU), i.e., no *block-*

ing technique can be possibly used in matrix-matrix (MM) multiplication. When blocking, the amount of memory access is $O(n^2)$, and that of computation is $O(n^3)$, the ratio between memory access and computation is in the order of $\frac{1}{n}$. However, given single input image, since it is a matrix-vector (MV) multiplication, in this case, the amount of memory access is $O(n^2)$ and the computation is $O(n^2)$, memory access and computation are of the same magnitude (as opposed to $\frac{1}{n}$). So MV multiplications are more memory-access-bounded, especially when we access a very large weight matrix (for e.g. FC6 in VGG, which takes 392MB). In CPU with high performance computing library for MM multiplication (like Intel MKL), the speed-up of compressed model with batch input must be very little, even slower, than its counterpart. We should not say batch processing will never be used in the embedded systems (asserted in the work of [1]), so giving a fair comparisons in both settings will suffice while taking the aforementioned factors into account.

– *On single conv layer vs. over whole network*: When evaluating XNOR-Net [67], the authors only infers the maximum speed-up on single conv layer, rather than over the whole network. Yet it shows a speed-up factor of X58, the number cannot fully demonstrate its running time performance. Good things is that testing on single conv layer will help us understand the model's running time on compressed conv layers, similarly to those methods specifically designed for FC layers. If we find it has obvious shortcoming in certain type of layers, then we can integrate other techniques to solve this problem. Overall, testing running performance on both single type of layer and whole network, is necessary for a comprehensive evaluation.

- **Compatibility to other compression techniques/models**: Some compression techniques/models can be combined, for e.g., SqueezeNet [2] plus Deep Compression, or you can use bottleneck module together with weight pruning, weight sharing after binarization etc. Therefore, how to fully discover the configuration space from all existing work remains an open question.
- **Easy implementation on software/hardware**: SqueezeNet supports 6-bit data type for its conv layers, where the implementation seems very difficult, and will be even more troublesome than 1-bit in XNOR-Net.
- **Energy consumption**: Currently, Deep Compression is one of the few work that reported energy consumption. They used *Intel pcm-power* for CPU socket and DRAM power. For GPU, they used *nvidia-smi* and for mobile GPU, they used a Jetson TK1 development board and measured the total power consumption with a power-meter. All other details can be found at their paper [1].

2.6.3 Ways to Compress Neural Networks

As the core idea of model compression is to speed-up running time with smaller model while maintaining the comparable accuracies, therefore we categorize model compression methods into three categories:

- **Mimic deep models using shallow networks**: Learn shallow networks to mimic the deep networks. Methods include [72, 57, 73].
- **Prune and quantize existing models**: Prune redundant, non-informative weights and quantize both weights and gradients using minimum-cost optimization, codebook and other coding technique (for e.g. Huffman coding) to largely reduce the number of parameters to store. Besides, the sparse connection matrix can be stored using format of compressed sparse row (CSR) or compressed sparse column (CSC). The advantage is obvious: we can prune large matri-

ces that are unable to fit into the caches. For e.g., the FC6 in VGG requires $102,760,448$ weights $\times$ 4Byte/2^{20} = 392MB memory to store (far from the capacity of L3 cache, which must in turn affect negatively on processing time). Once pruned with compression rate 1.10% as reported in Deep Compression, the memory storage for effective weights now ideally becomes 4MB! One disadvantage of such methods is that they all contain critical parameters including number of bits to store weights, number of bins used in quantization, etc., which must be fine-tuned on existing networks (in most work, they prefer testing on LeNet, AlexNet, VGG, ResNet, GoogLeNet). Given a new network model, no matter shallow or deep, those parameters need to be tuned again so that neither they will be too extreme to compress, nor they are too small to fit that model. Methods include [74, 75, 76, 77, 78, 65, 1, 79].

- **Replace layers using bottleneck module**: *bottleneck module* (like *Inception* Module in GoogLeNet [53]) has been shown to reduce the parameters to a great extent, while achieving higher non-linearity. An extreme case is SqueezeNet [2], where the parameters were surprisingly compressed by a factor of 50! Therefore, as long as the "squeezed" modules capture the intrinsic features against various variances, bottleneck module is preferred in model compression. Methods include *Network in Network* [54], GoogLeNet v1-v4 [53, 80, 55, 81], ResNet [82] and more recent SqueezeNet [2], Xception [83], ShuffleNet [84] and MobileNet v1-v2[85, 86].
- **Approximate parameters using lower-bit or quantitative vectorization**: Many other approaches have been proposed either to quantitatively vectorize parameters or approximate networks using $\mathcal{L}_2$ error minimization, in the hope that, the accuracy will not drop much when approximated (vectorized) parameters/networks are used. Methods include [87, 88, 89]. In addition, since

high precision storage of parameters including weights and gradients (and even the input as done by [67] are not necessary in training the neural networks, therefore, all parameters can be further compressed by using a lower-bit storage format. 8-bit [90] and 3-bits activations [91] have already explored. Now it is the time for binary bit networks!

- **Binarize networks: 1-bit weights and activations**: Using binary parameters in network is believed to have poor performance due to the limited information it can represent. In such an extreme scenario, *BinaryConnect* [92] and *BinaryNet* [93] were proposed, where the former was reported to perform very well (state-of-the-art) only on small datasets like CIFAR-10 and MNIST but is not ideal for ImageNet (according to [67]), while the latter method is an extension of the former one. The state-of-the-art binarization model on ImageNet, including XNOR-Net [67] and DoReFa-Net [68] are proposed recently. Furthermore, it is noteworthy that Expectation BackPropagation (EBP) [94] and its variants [95, 92], provide a good lead on binarizing neural networks in a probabilistic approach, i.e., within the variational Bayesian framework.

2.6.4 Ways to Reduce Parameters

Some principles were discussed in [2] about designing a network with fewer parameters. The main idea is to use:

- **Smaller filter size**: Resize filters into 1 × 1 as best as we can, so that we can have 9X fewer parameters, compared to 3 × 3 filters used in VGG networks.
- **Smaller number of input channels and number of filters.**: Thinking about how we calculate storage of parameters in the conv layers:

$$\#\text{ of param. to store} = \#\text{ of filters} \times w_h \times w_w \times c, \qquad (2.1)$$

where w_h and w_w are the height and width of filters. Normally, w_h, w_w are fixed: we use 7×7 against input layer (for ImageNet), 5×5 at early layers, and 3×3 even 1×1 within the large body of network (middle layers). So, as for model compression the task now becomes reducing # of filters and c while maintaining the accuracies.

- **Larger activation map**: Especially when we have larger activation maps at most early layers that are believed to achieve higher accuracy (i.e., delayed downsampling as shown in both [82] and [2]).

2.6.5 Avoid FC Layers

Fully connected (FC) layers are critical in training a deep learning model, due to the high linear transformation they can make. However, when we calculate the number of weights learned, FC layers (especially the first FC layer after the last conv layer, which in this thesis we call *FC6*[14]) often needs much larger storage than its counterparts (conv layers, pooling, etc.).

Take FC6 in AlexNet as an example, according to [1], it learns **38M** weights, which in turn requires extreme large amount of storage. Inspired by Deep Compression, we can prune the weights (without losing much representation power) to have a sparse connectivity matrix to calculate $\mathbf{XW}$. Deep Compression has pruned the number of weights in FC6 of AlexNet to 9% in order to accelerate the computations!

Otherwise, we may consider to avoid using FC layers by replacing them with the conv layers, which can be further binarized and calculated using only add operations instead of multiplications [67]. In [54], *global average pooling* layer was used to replace FC and directly mapped from feature maps to category scores. As a more native implementation, FC layers can be easily replaced by conv layers, where a great

[14] Conventionally, in the pioneering work of AlexNet, the first FC layer is the 6th layer.

example was given at the tutorial [96]. Take AlexNet again as an example, using a 1×4096 FC layer right after the output of pooling layer (of size $7 \times 7 \times 512$), is equivalent to going through a conv layer with 7×7-sized filters of depth 4096[15], so that the output is a volume of size $1 \times 1 \times 4096$. As for the next FC layer, the conv layer will be with 1×1 filters of depth 4096 again, and so on so forth.

According to [96], there exists another benefit by applying this technique: "this conversion allows us to slide the original ConvNet very efficiently across many spatial positions in a larger image, in a single forward pass. Evaluating the original ConvNet (with FC layers) independently across 224x224 crops of the 384x384 image in strides of 32 pixels gives an identical result to forwarding the converted ConvNet one time." It means that we can resize an input image into larger size, feed it into a converted neural network to have class scores at many spatial positions and then average the class scores. It is indeed what we have seen in feature augmentation, i.e. crop images and feed them into original networks, and then average the class scores to get the final scores. But this time, after we convert our neural network, the efficiency has been improved quite a lot (feeding smaller images for multiple times vs. feeding larger image into converted network for just one time).

2.6.6 Placement and Tuning of Normalization Layer

In SqueezeNet, they only placed one *Dropout* layer (with ratio 50%) after its fire9 module, which is very close to the classification layer. The normalization technique needs to be paid great attentions in a compressed model. By following the principles in Sect. 2.6.4, and especially after we prune our weights (which acts as sort

[15] Here, depth means the number of filters.

of normalization), normalization layers like Dropout [97] and *Batch Normalization* [80], need to be placed at very late positions[16].

Also, their parameters need to be fine-tuned, in terms of how many effective weights have been *picked* before the normalization layer. Take SqueezeNet as an exmaple, a ratio 50% at fire9, indicates that the number of effective weights are much less than what we expected on original models (for which we need to place Dropout right after first conv layers, and afterwards every three conv layers like what has been done in VGG Net).

However, how to define *effective weights* is still arguable and remains as open question. Han et al. [1] used KNN to cluster weights and assign each weight a specific index of the nearest centroid, in order to reduce the space to store those weight in an offset manner. Three initialization methods were examined: Forgy (random), density-based, and linear. Since we have few large weights when training the network, they tend to be clustered to small centroids, if we initialize centroids in either Density or uniform based approach. In contrast, linear initialization allows large weights a better chance to form a large centroid, thus perform better than other methods as described in [1].

2.6.7 Improvement With Model Compression

As has been shown in the literature of model compression, the compressed models sometimes outperforms the original model by a small margin. For e.g., in Deep Compression [1], its top-1 classification error with VGG-16 on ImageNet is 0.33% less than the original model, and 0.41% smaller in terms of top-5 classification error. The technique used in Deep Compression including pruning plus quantization,

[16] By *late*, we mean the layers are close to the classification. Similarly, those close to input layer are referred to as *early* layers.

plays sort of a role as the network normalization, where such an argument by using the similar technique (*DropConnect*) can be found at [98]. According to [67],“the noise introduced by weight binarization provides a form of regularization, which could help to improve test accuracy.”

It is not always true that when compressing network models, we must consider the trade-off between memory compression rate and classification accuracy. We can even think about to *improve* the accuracy by compressing models with some normalization techniques, one great example would be using the bottleneck module as in GoogLeNet [53].

CHAPTER 3

OBJECT DETECTION

This chapter studies particularly the application of CNNs in object detection. Starting from a discussion about the important role that contextual information plays for object detection, this chapter firstly analyzes the empirical receptive fields of a classic VGG16 network. Afterwards, in light of our findings with SSD network, this chapter further proposes a novel Context-aware Single-shot Detector, that has been shown to outperform SSD to a great extent [3].

3.1 Introduction

Deep learning approaches have shown some impressive results in general object detection. However, there still remain fundamental challenges to be addressed, particularly in detecting objects of dramatically different scales. In the literature, many attempts have been made to overcome this issue: from *image pyramids*-based approaches which have often been combined with hand-crafted features [99, 100] to *feature map pyramid*-based approaches [29, 30] within a deep learning framework. In addition, the state of the art has moved from the *sliding-window* paradigm to the much more efficient alternative of *feature maps scanning*, thanks to the representation and learning capabilities of Convolutional Neural Networks (CNNs).

Within the deep learning paradigm, methods predicting proposals from feature maps of the single highest-level scale [13, 14, 29] enable the most variance of scales, due to the shared semantics. However, such methods also suffer from slow inference time, given that the hypothesized proposals at all scales need to go through the whole CNN, leading to large computational cost and memory usage. Single Shot Detector (SSD) [30] mitigated this issue by making predictions from feature maps of multiple scales in a hierarchical approach. This way, when hypothesizing proposals at increasing scales, SSD goes deeper in CNNs with more learnable parameters and thus takes longer time. Due to this *bottom-up* design, SSD assumes that small object detection only relies on fine-grained local features, and ignores context outside those local features. In Fig. 3.1, we show, for an SSD detector, the only *conv4_3* Receptive Field (RF) that was used to detect the smallest object. Within that RF, we can hardly recognize the object of interest (green box). After expanding the *conv4_3* RF into multiple higher scales, more visual cues (marked by red ellipses) become available. With that contextual information, it is possible not only to recognize the object as a sheep, but also to detect the presence of a herd in the given picture.

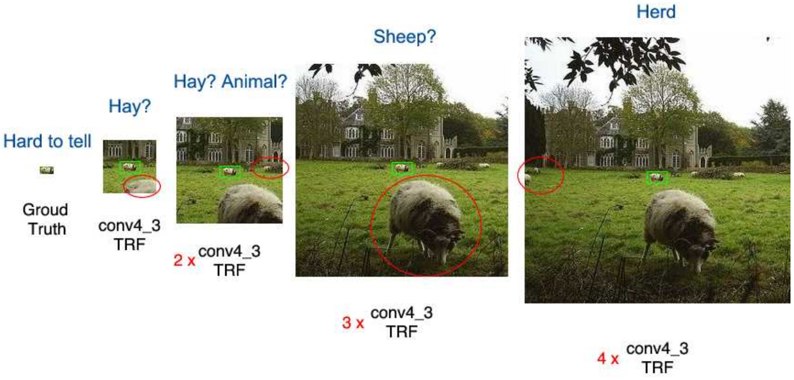


Figure 3.1: In SSD [30], *conv4_3* of the VGGNet was used to detect the smallest object. As the TRF of *conv4_3* includes limited visual cues, integrating informative context from different scales can help us detect very small objects.

Both two dominant deep learning object detection methods: Faster R-CNN [29] and SSD [30], require pre-computed grids (named anchors in [29] and default boxes in [30])[1] to either generate proposals or regress and classify directly upon them. Most of the previous work in deep learning object detection focuses on architecture design of networks, but there is relatively little work studying the underlying instrument of generating grids and proposals. Consequently, the hierarchical structures in networks and grid scales have to be fine-tuned exhaustively in order to obtain satisfactory results.

In the VGGNet version[2] of SSD, prescribed grid scales associated with each RF size were specifically designed at different prediction layers, to ensure that every point on the feature maps of prediction layers *sees* a sufficiently large area from input image. Take as an example a 300×300 input: the grid scales versus RF sizes at all SSD's 6 prediction layers are 30/92, 60/420, 114/452, 168/516, 222/644 and 276/644 respectively. However, according to [101, 102], the effective receptive fields (ERFs) are 2D-gaussian distributed and proved to be significantly smaller than the corresponding theoretical receptive fields (TRFs).

Following [101], our analysis (Fig. 3.5(c)) shows that the ERF of *conv4_3* in SSD is only ×1.9 larger than the corresponding grid scale θ_p (not exceeding ×2.5 across all prediction layers), which motivates the need for more contexts to be integrated into the existing framework. In this chapter, we present a context-aware framework for SSD (Fig. 4.3), and give two different implementations, which either expands RF sizes with multiple scales and then fuses to form new prediction layers, or reuses directly feature maps from higher layers to integrate complex structure features. Because the scale parameters are automatically learned during fusion of contextual feature maps,

[1] We use grid and default box interchangeably throughout this chapter.

[2] We compare the performance of ResNet50/101/152 versions of SSD as well in our experiments. Considering that the runtime performance of SSD is dramatically reduced with ResNet101/152, in this chapter we mainly investigate context layers for the VGGNet version of SSD.

the proposed network enables the trade-off between fine-grained features and richer features encompassing more context.

Regarding to object detection, in this chapter, we make the following main contributions:

(1) To our best knowledge, we are the first to analyze ERFs within the framework of object detection. Using our analysis, we provide the ERF sizes of a standard VGG16 network [7]. These sizes can be utilized as a reference to design more effective CNNs for example in object detection.

(2) We present a new framework based on SSD, by introducing multi-scale dilated convolution layers in a hierarchical approach, named Dilation-based Context-aware SSD (DiCSSD). Moreover, we alternatively propose a VGGNet-based, deconvolutional version of context-aware SSD (DeCSSD). Both were designed specifically for small-scale object detection with scaling parameters learned automatically during feature map fusions. Our experimental results on VOC [103], MS-COCO [104] and DETRAC [105] show that both DiCSSD and DeCSSD outperform SSD while maintaining real-time speed, in addition to producing promising detection results on small objects.

3.2 Related Work

Early object detectors in CNNs. Until a few years ago, the sliding-window paradigm [106, 107] was commonly used. With the emerging technology of deep learning, G. Ross et al. [12] proposed an object detection framework based on deep neural networks named R-CNN, which performs a forward pass for every object proposal and then classifies. Its followers, SPPnet [14] and Fast R-CNN [13] have been shown to largely reduce the training and inference time by sharing computation upon convolutional feature maps for the entire input image.

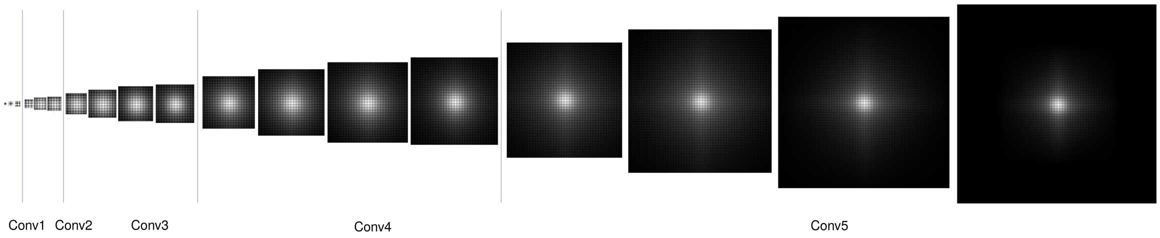


Figure 3.2: Visualization of ERFs vs. TRFs in a VGG16 network.

Faster R-CNN. With the introduction of Region Proposal Network (RPN), the seminal work Faster R-CNN [29] unblocked the bottleneck imposed by hypothesizing region proposals that took a large amount of processing time in Fast R-CNN. Despite their crucial dependency on the proposal generation methods/networks, variants of Fast(er) R-CNN[3][108, 109] advanced the performance to a great extent in a majority of open datasets like vehicle detection datasets KITTI [110], DETRAC [105], and some general object detection datasets like PASCAL VOC [103] and MS-COCO [104].

More recent methods based on Faster R-CNN framework include Feature Pyramid Network (FPN) [4] which designs a generic architecture with lateral connections between low-resolution feature maps and higher, and Region-based Fully Convolutional Networks (R-FCN) [109] which focuses on translation-invariance detections.

Single-Shot Detectors. Most Fast(er) R-CNN methods are able to classify ROIs in 200ms[4], however, proposal generation takes much longer than that and becomes a major bottleneck in deploying the trained model into real-time systems. There-

[3] In this chapter, we refer to *Fast(er) R-CNN* as the CNNs that require region proposals using either RPN or external proposal methods.

[4] A standard Faster R-CNN achieves 5fps on general image datasets.

fore, by directly sampling grids upon the input image, and then training a CNN to directly regress and classify (namely single-shot prediction), method You Look Only Once (YOLO) [111] accelerated the runtime performance to 45fps. However, YOLO incurred a non-negligible penalty in accuracy, due to the coarseness of the features that were learned. One of the state-of-the-art single-shot prediction methods, SSD [30], alleviated this problem with an extension of VGG16 network used to predict multi-scale grids (default boxes named in [30]) in a hierarchical structure. Despite having distinct advantages over Fast(er) R-CNN, SSD suffers in accuracy at the task of detecting small objects, because it cannot integrate context for the features learned in bottom layers. We note that context has been shown crucial in recognizing tiny objects [112]. Furthermore, as the input size of SSD is limited below ∼512 (otherwise its number of default boxes grows exponentially, incurring overwhelming computational and memory cost), its performance on small object detection is greatly constrained.

More recently, YOLO9000 [113] improves accuracy by integrating dataset-specific data augmentation techniques, which brings the need for more hyper-parameters. DSSD [114] is also built directly on the existing SSD framework, and therefore is the most similar to our proposed method. However, DSSD opts for ResNet101 as its base network, and therefore its runtime performance has dropped from 46fps to 10fps. Both the proposed CSSD and DSSD were designed to improve small object detection. The proposed CSSD method achieves real-time speed and has been shown to perform significantly better than SSD, in both MS-COCO [104] and vehicle detection dataset DETRAC [105].

Effective Receptive Fields. Receptive field (RF), also called *field of view*, refers to a region that a unit neuron in a certain layer of the network sees/depends on in the original input image. Most previous work utilized the theoretical receptive field sizes to guide their network designs [55, 115]. However, those designs have not considered whether each pixel in the TRF contributes equally to the output of a certain unit neuron. Zhou et al. [101] were the first to introduce the concept of *empirical receptive fields*, and showed via a data-driven approach that the actual size of RF is much smaller than TRF. Nevertheless, a solid mathematical model of how the empirical receptive fields relate to their theoretical counterparts was offered only recently, by Luo et al. [102], who pointed out that not all pixels in the *effective receptive fields*[5] contributes equally to a unit neuron's response. Instead, the centers of RFs have much larger impacts on the output leading to an obvious 2D-gaussian shape. This is because in the forward pass the central pixels can reach the output with many more different paths than the pixels in outer area, while in the backward pass gradients are back-propagated across all paths equally [102].

Inspired by [101], we calculate the ERF sizes across all VGGNet layers in SSD with associated TRF sizes (Fig. 3.2). To our best knowledge, we are the first to provide the ERF sizes of a standard VGG16 Network [7]. In addition, we compare our computed results against the fitted ERF sizes (Table 3.1) with $\sqrt{\mathrm{TRF}}$, whose finding suggests us to introduce the proposed context layers, which are fused so as to enable the trade-off between fine-grained features and richer features encompassing more context.

instead of $\sqrt{\mathrm{N}}$ (number of convolution layers) used in [102].As the number of convolution layers is linearly correlated with TRF size, our finding in Fig. 3.5(b) is in agreement with [102]. Furthermore, after analyzing the ERF sizes of SSD, we found that the context learned by the original SSD was insufficient in terms of its coverage, with respect to the grids at each prediction layer (Fig. 3.5(c)). This finding inspired us

[5] In this chapter, we use empirical receptive fields and effective receptive fields interchangeably and both of them are shortened for ERFs.

to introduce the proposed context layers, which are fused so as to enable the trade-off between fine-grained features and richer features encompassing more context.

3.3 Context-Aware Single-Shot Detector

We show the framework of our proposed CSSD in Fig. 4.3. Both CSSD and SSD[6] utilize a standard VGGNet [7] as its base network[7]. However, CSSD uses the feature maps of prediction layers after fusion of context layers, rather than using them directly as in SSD.

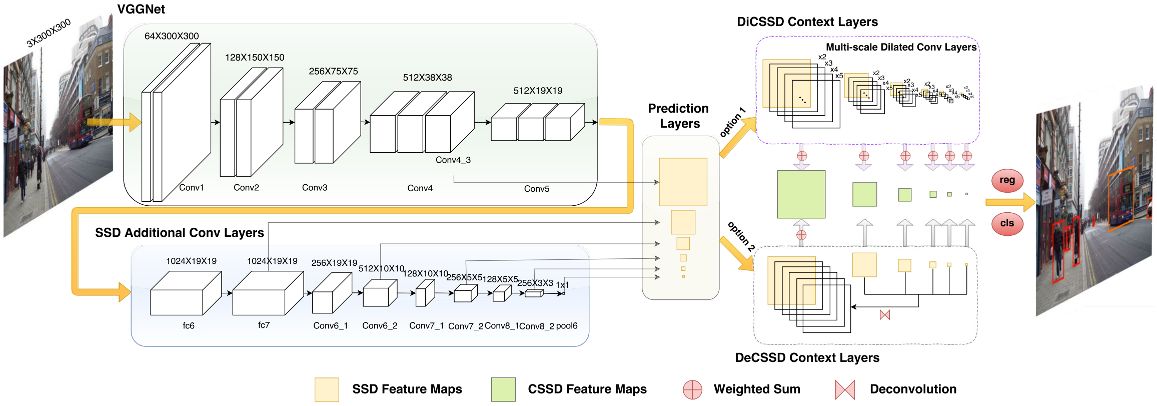


Figure 3.3: Flow diagram of CSSD. We built CSSD directly upon SSD with two implementations of context layers using 1) multi-scale dilated convolution layers, 2) deconvolution layers. The former has an adjustable parameter controlling the number of context layers, while the latter directly reuses the feature maps from upper layers fused specifically for smallest object detection. Both implementations learn associated scaling parameters during fusion of feature maps.

To integrate context for small object detection, intuitively we reuse the feature maps from top layers and merge them into the bottom one. However, as the feature

[6] In the newest implementation of SSD [116], its additional conv layers have been extended further with more parameters, but the number of prediction layers is unchanged. Also note that any similar extensions to SSD can easily be incorporated into CSSD.

[7] In this paper, *VGGNet* refers to the VGG16 version of [7].

map sizes differ a lot (for e.g. by a factor of 1/2 after every pooling layer with stride = 2), we need to *upsample* maps from top layers first and ensure all sizes are the same. We can consider the convolutional part of the network as an *encoding* step, and deconvolution layers can be treated as a *decoding* processing step. The encode-decode structure is known as an *hourglass* and has been shown to be particularly useful in segmentation [117]. As SSD branches out into different prediction layers, in our design of DeCSSD (option 2 in Fig. 4.3), we make hourglasses from all prediction layers except the first one (where the hourglass is essentially the layer itself) and fuse them with learnable scaling parameters across all maps. Subsequently, the fused feature map becomes the new prediction layer used for small object detection only.

Using deconvolution layers to integrate multi-scale contexts has one obvious drawback, namely that the memory usage of network increases significantly, because the coefficients of bilinear filters and the following conv layers (which have been verified to be especially useful in our experiments) take many more weight parameters. Besides, DeSSD does not always work *out of the box*. Due to that, the training error during fusion of *conv4_3's* context layers is prone to drifting. To address that issue, we additionally add one batch normalization layer that allows for more stable learning after each context layer.

We alternatively propose a more lightweight, finetuning-friendly method to integrate contextual information into SSD's framework, i.e., DiCSSD which uses multi-scale dilated convolution layers [118] shown as option 1 in Fig. 4.3. In DiCSSD, the context layers are fused via weighted sums (implemented as element-wise summations with learnable scaling parameters following a batch normalization layer) of multi-scale dilated convolution layers, where every original prediction layer is used individually, unlike DeCSSD that has explicit messaging between them.

DiCSSD rapidly expands the TRF sizes of each prediction layer, thus ensuring that every feature point within *sees* sufficiently large areas. If we choose a setting of 4 context layers in total for every prediction layer, then the TRF size in individual prediction layers will be ×2, ×3, ×4 and ×5 larger (Fig. 3.1). Intuitively, during feature map fusion the network collects a full set of visual cues that performs best in recognizing the object of interest. Note that the number of context layers is a hyper-parameter. In Sec. 3.6.1. we have set the value of that hyper-parameter to 4, based on cross-validation experiments.

3.4 Empirical Receptive Fields

3.4.1 A Data-Driven Approach

	conv2_1	conv2_2	pool2	conv3_1	conv3_2	conv3_3	pool3	conv4_1	conv4_2	conv4_3	pool4	conv5_1	conv5_2	conv5_3
TRF	10	14	16	24	32	40	44	60	76	92	100	132	164	196
ERF (Fitted [102])	5.1	9.9	12.0	19.5	25.8	31.4	34.0	43.3	51.4	58.6	62.1	74.5	85.4	95.4
ERF (Data-Driven)	6.9±0.1	9.8±0.2	11.1±0.2	16.4±0.3	21.4±0.4	25.4±0.6	27.2±0.8	36.8±0.8	45.7±1.0	54.1±1.6	56.9±1.8	77.6±2.9	94.4±4.4	104.2±7.8

Table 3.1: Comparison of sizes of ERFs and TRFs in a VGG16 network. Results of both a data-driven approach inspired by [101] and the fitted values based on theoretic $O(\sqrt{\text{TRF}})$ [102] are given.

We show how to obtain ERFs in Alg. 1 and Alg. 2 respectively. While inspired by [101], our algorithm differs from that [101] in that: 1) We evaluate our algorithms within an object detection framework, instead of a classification framework in [101]. Therefore, our algorithm selects ROIs with the highest activations over K images for the given neuron, rather than top-K images in [101]; 2) We visualize and validate ERF sizes of all conv layers in a VGGNet (16 layers in total), which has been widely used in object detection methods, while [101] only provides ERF sizes of Places-CNN and ImageNet-CNN (5 layers in total), which have not been used much in object detection; 3) We further calculate ERF sizes of SSD additional conv layers using

[102], based on the curve fitted from values of VGGNet. Our result justifies why SSD has to extent its network from VGGNet with many conv layers and even more in its newest implementation [116]: If we take SSD300 as an example, the TRF size of *pool5* (VGGNet's last layer) is 196 but its ERF is only about 104, which fills merely about 1/9 area of the original image. Therefore, the network is unable to learn fully the complex structures of very large objects.

Before running Alg. 1, we need to calculate the TRF size at the i-th layer, denoted as $H_t \times W_t$. After assigning $H^i = 1$ and $W^i = 1$ (according to the definition of RF), TRF size is calculated in reverse order of feature map size , i.e. recursively from i-th layer to the 1st layer:

$$\begin{aligned} H^{i-1} &= (H^i - 1) \cdot s^i + l^i \cdot (k^i - 1) + 1 \\ W^{i-1} &= (W^i - 1) \cdot s^i + l^i \cdot (k^i - 1) + 1, \end{aligned} \tag{3.1}$$

where s^i, l^i, k^i denote convolution stride, dilation stride and kernel size at the i-th layer respectively. Finally, we have $H_t = H^1$, $W_t = W^1$.

The ERF sizes of VGGNet are shown in Table 3.1. We compute ERF (Alg. 2) with a total of $K = 300$ images and threshold $\sigma = 1.0$. With $K = 100$, the computed ERF values across all layers show at most ± 0.1 difference from the corresponding values computed with $K = 300$, therefore it is safe to use only 100 images.

3.4.2 Analysis and Visualization

We show some example discrepancy maps of *conv4_3* in Fig. 3.4. Each discrepancy map shows regions to which certain neuron are most responsive. Those regions are semantic (car roofs/shields, highway shrubs, lanes, etc.) and thus can offer us more insights into how neurons respond to our input image (similar to *emergence of detectors* in [101]).

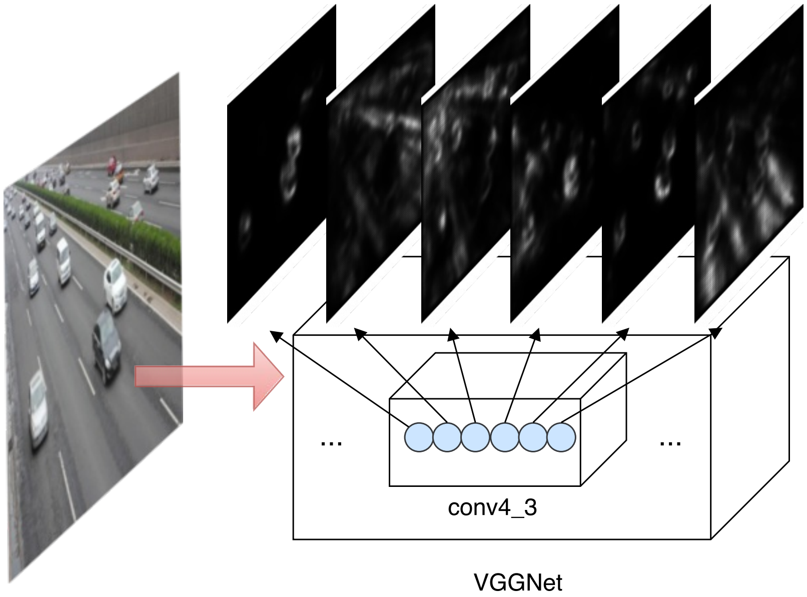
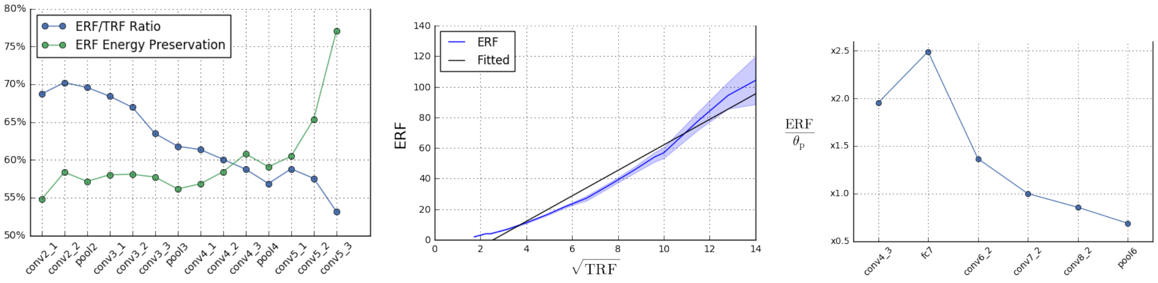


Figure 3.4: Example discrepancy maps of *conv4_3*. Each neuron responses differently with semantics from input image. The raster effects on discrepancy maps are due to *conv4_3*'s limited ERF size around 54.

After obtaining K discrepancy maps, for all neurons in a certain layer we extract ROIs centered at the points with max activation (named *Calibration* in Alg. 2). Subsequently, ROIs are averaged over all neurons, after which we will see a typical 2D-Gaussian shape in accordance with [102], as shown in Fig. 3.2.

Fig. 3.5 demonstrates our findings about ERF: In Fig. 3.5(a), $\frac{\text{ERF}}{\text{TRF}}$ becomes smaller at top layers which is again in accordance to [102]. In addition, we fit a linear curve of ERF vs. $\sqrt{\text{TRF}}$ sizes in Fig. 3.5(b), where the variations of two lines are mainly due to the pooling layers interleaved before *conv5_3* in VGG16 network, as well as the data noise introduced when calculating ERFs. Last but not least, we compare ERF sizes against grid scales of SSD at different prediction layers in Fig. 3.5(c), from which we realize that the sizes of SSD's default boxes are not sufficiently large to include context, especially for small objects (*conv4_3*).



(a) (b) (c)

Figure 3.5: (a) As the network goes deeper, ERF retains more intensity, and surprisingly has lower ratio against its TRF counterpart. (b) A likely-linear relationship between ERF and $\sqrt{\mathrm{TRF}}$ sizes. (c) Fitted ERF sizes to corresponding grid scales θ_p at different prediction layers.

3.5 Training

We follow the same training process as [30]: After generating default boxes, we match each of the ground truth boxes to the best overlapped default boxes with Jaccard overlap higher than 0.5. For the rest of default boxes, which have been left unmatched, we select a subset of them based on confidence loss while keeping the ratio of matched to unmatched boxes to 1:3, which keeps a balance between number of positive and negative proposals. Afterwards, our objective function minimizes regression loss using Smooth ℓ_1 and classification loss using Softmax. Note that SSD was updated with a new expansion data augmentation trick that has been shown to boost the performance a lot in small object detection [116]. In this chapter, we follow the newest data augmentation expansion trick in our experiments on VOC, while keep using the original data augmentation technique [30] on DETRAC.

Both implementations [30, 116] use the entire original image as input, and then randomly sample patches that have the *minimum* Jaccard overlap with one of the ground truth objects. Multiple minimum Jaccard overlap thresholds have been used, including 0.1, 0.3, 0.5, 0.7 and 0.9, each with 50 maximum trials. After doing so, the statistical distribution of training sample scales fed into the network is expected to be more *equalized*.

3.6 Experiments

3.6.1 Ablation Study

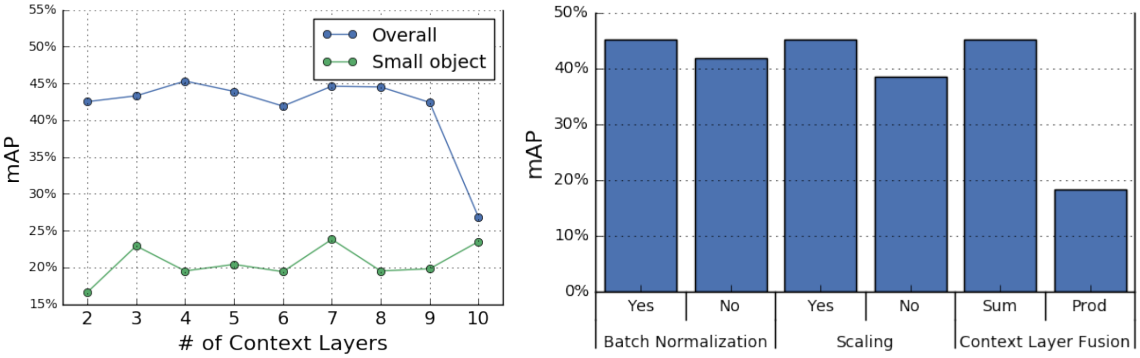
In Fig. 3.6(a), we study how mAP changes with different number of context layers for DiCSSD, as measured on the DETRAC dataset. Note that the overall mAP score is not necessarily indicative of the mAP value for small object detection. We note that with higher number of context layers, the overall mAP score of DiCSSD drops dramatically while memory requirements increase. We set the number of context layers equal to 4, based on cross-validation on the DETRAC dataset, and we used the same number (4 context layers) for the VOC 2007 dataset. In addition, we conducted experiments under different settings of batch normalization, scaling, and context layer fusion method and show the results in Fig. 3.6(b). The *golden* model compared in Fig. 3.6(b) has been trained with a full combination of batch normalization + scaling + sum.

3.6.2 Convergence Speed

In Fig. 3.7, we demonstrate the superior convergence speed of our proposed method DiCSSD against SSD. Due to that the learning rate was set to decay at 60k, 80k, 100k iteration for VOC and 280k, 360k iteration for MS-COCO respectively, we can find noticeable mAP improvements at those iterations in Fig. 3.7.

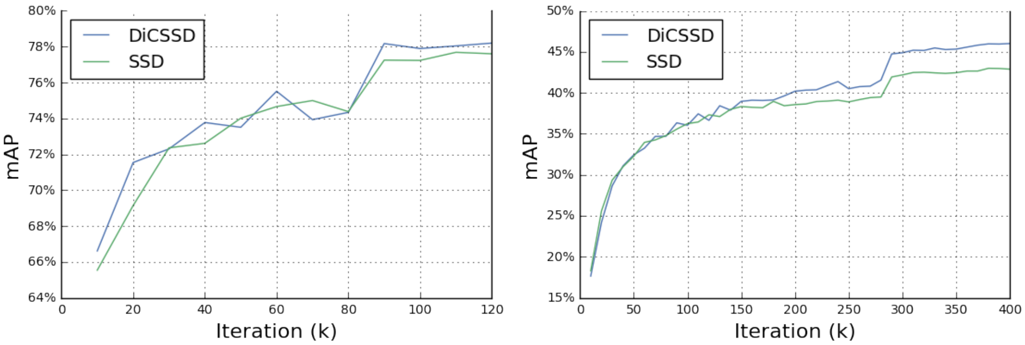
3.6.3 Sensitivity and Impact Analysis on PASCAL VOC 2007

Using [119], we perform sensitivity and impact analysis of different object characteristics on PASCAL VOC 2007 in Fig. 3.8, where less values of sensitivity indicate



(a) (b)

Figure 3.6: We perform our ablation study on DETRAC (Sec. 3.6.4) to test, for our DiCSSD method: (a) mAP under different number of context layers. (b) Effects of different settings with batch normalization, scaling and context layers fusion.



(a) VOC 2007 *test* set (b) MS-COCO *minival* set

Figure 3.7: Comparison of convergence speed.

more robustness to certain characteristic. We note that the sensitivity of DiCSSD in terms of the *BBox area* is lower than that of SSD (0.421 vs. 0.453). Besides, due to the additional context modeling, DiCSSD has been shown to perform better than SSD under various *occlusion* conditions (0.327 vs. 0.324 in sensitivity analysis).

In Fig. 3.9, we give visualization results of calculating ERF for *conv4_1* as described in Alg. 2 of our submission. Though ROIs whereby each neuron (right part) are mostly responsive show different activation shapes, their averaged result

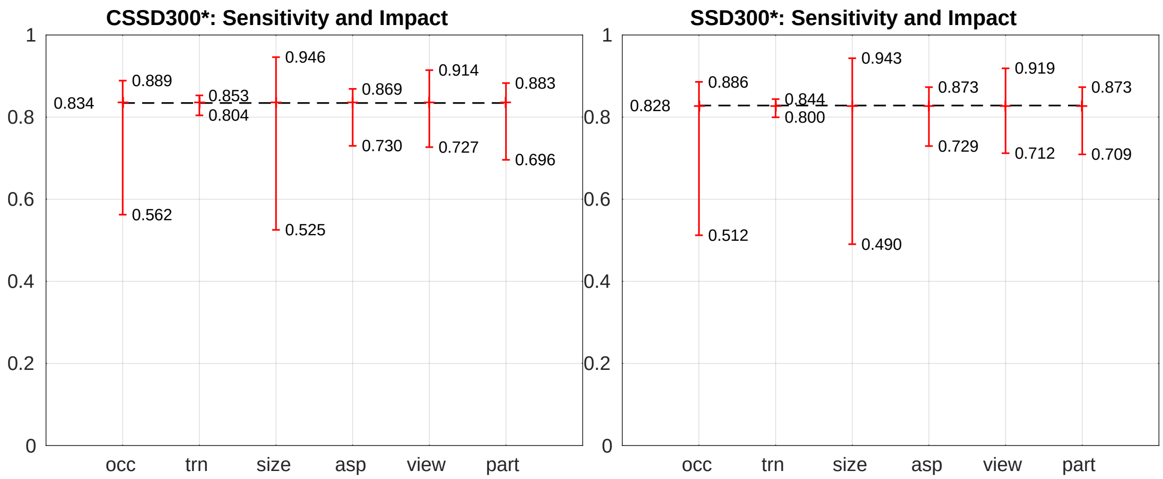


Figure 3.8: Sensitivity and impact analysis of different object characteristics [119]. We show the average mAP over categories within each characteristic (occlusion, truncation, bounding box area, aspect ratio, viewpoint, part visibility), with the highest performing and lowest performing shown in single column. Overall mAP across characteristics is indicated by the black dashed line. The difference between max and min indicates sensitivity while the difference between max and overall indicates the impact.

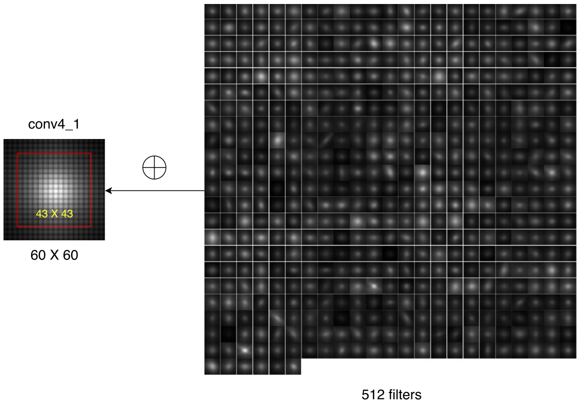
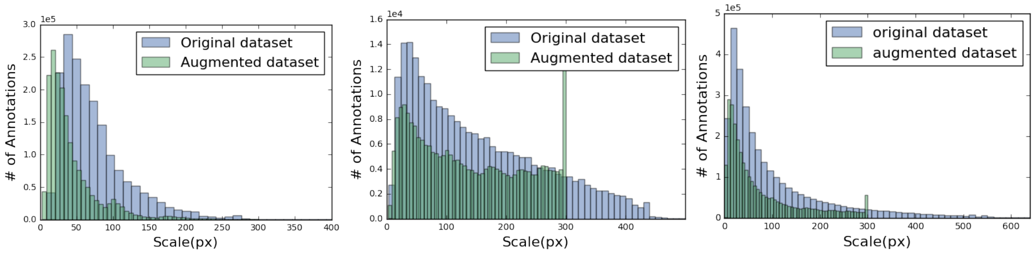


Figure 3.9: Exemplified ERF computation for *conv4_1*. All 512 ROIs in right part are calibrated and averaged over $K = 300$ input images. Both TRF (cropped patch) and ERF (red box) are given.

(left part) is a typical 2D-Gaussian. The results are in accordance with the findings from [102].

We review several important properties of SSD that have not been fully studied in [30].



(a) DETRAC (b) VOC0712 (c) MS-COCO

Figure 3.10: Distribution of ground truth annotation scales in training sets of DETRAC, VOC0712 and MS-COCO, versus the same datasets after data augmentation with SSD300.

Effects of data augmentation. Fig. 3.10 demonstrates that all three object detection datasets have an decreasing number of annotations as the objects become larger, due to which the SSD data augmentation technique polarizes the scale distribution especially on VOC0712 and MS-COCO. For DETRAC, however, as the number of relatively small objects dominates, in each trial it can hardly reach higher jaccard thresholds of 0.7 and 0.9, resulting in a *half-polarized* distribution. Our statistical analysis of DETRAC in Fig. 3.10(a) ($\mu \pm \sigma$): 74.6 ± 48.9 vs. 43.5 ± 37.7, indicate that the data augmentation technique from SSD [30, 116] tends to shrink the ground truth annotations to $\times 0.6$ its original size. The scale coming with the largest number of annotations has been shifted from 32.3px to 16.5px after augmentation.

Algorithm 1 Discrepancy maps

```
Input: A pre-trained model $Z$, input image set $U \in \{I_n | n = 1, \dots, N\}$, $I_n \in \mathcal{R}^{H_I \times W_I}$,
    occluder $o \in \mathcal{R}^{d \times d}$, stride $s$, layer $x$ with output size $H_f \times W_f \times l$
Output: Discrepancy maps $D \in \mathcal{R}^{H_I \times W_I \times l}$
Compute TRF size $H_t \times W_t$ at layer $x$ using Eqn. (3.1)
Locate only fully responsive ROI $Q \in \mathcal{R}^{H_Q \times W_Q}$ within area of rows $\in [\frac{H_t}{2}, H_I - \frac{H_t}{2}]$,
    cols $\in [\frac{W_t}{2}, W_I - \frac{W_t}{2}]$ /*To speed-up computation*/
Create empty discrepancy maps $D$
while $i \leq N$ do
    In $Q \in U_i$, create occluded image set $V \in \{I_m | m = 1, \dots, M\}$, $I_m \in \mathcal{R}^{H_I \times W_I}$
    using $o$ with stride $s$
    Record coordinates of each occluded area $P \in \{P_m | m = 1, \dots, M\}$, $P_m \in \mathcal{R}^{d \times d}$
    while $j \leq M$ do
        Feedforward $U_i$ and $V_j$ through $Z$. At layer $x$, obtain activation maps re-
    spectively $A^x_{U_i}$ and $A^x_{V_j}$, both $\in \mathcal{R}^{H_f \times W_f \times l}$
        while $k \leq l$ do
            $D^k_{P_j} \leftarrow D^k_{P_j} + \sum |A^{x_k}_{U_i} - A^{x_k}_{V_j}|$
        end while
    end while
end while
```

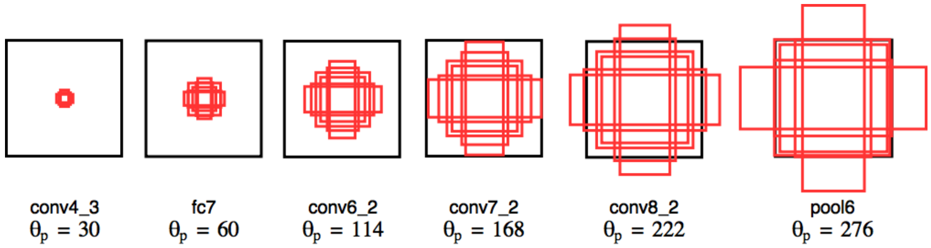


Figure 3.11: At different prediction layers, SSD [30] generates default boxes with various but uniformly sampled scales θ_p. All outbound default boxes will be clipped.

Algorithm 2 ERF

```
Input: Discrepancy maps $D \in \mathcal{R}^{H_I \times W_I \times l}$, TRF size $H_t \times W_t$ at layer $x$, threshold $\sigma$
Output: ERF size $H_e \times W_e$
while $i \leq l$ do
    $P_i \leftarrow \operatorname*{argmax}_{(m,n)} D^i$ /*Calibration*/
    $E_i \leftarrow$ ROI centered at $P_i$ in $D^i$, $E_i \in \mathcal{R}^{H_f \times W_f}$
    $S_i \leftarrow \sqrt{\sum [E_i \geq \sigma * \mu(\text{vec}(E_i))]}$ /*[...] are the Iverson brackets*/
end while
$H_e \leftarrow \mu(S)$
$W_e \leftarrow \mu(S)$
```

Offsets between RF centers and default boxes. We denote the feature map size at certain prediction layer as $H_f \times W_f$. SSD uniformly generates a dot matrix (blue dots in Fig. 3.13) of size $H_f \times W_f$ given input image $I_m \in \mathcal{R}^{H_I \times W_I}$, using stride along x, y axes as $\frac{H_I}{H_f}$, $\frac{W_I}{W_f}$ respectively. Subsequently, those dots are treated as the centers of a group of default boxes (Fig. 3.11). There exist, however, obvious offsets between the centers of default boxes and the centers of RFs that has not been mentioned in [30, 116].

Using above calculation, Fig. 3.12 demonstrates a simplified example for a 5×5 input image being convoluted by 3×3 filters, wherein there exist noticeable offsets in-between. In Fig. 3.13, we further calculate and visualize the offsets at layers before (and including) each pooling layer or conv layer with stride ≥ 2, given input size 300×300. As the network goes deeper, the offsets visualized in Fig. 3.13 become much larger, thus leading to inaccurate estimation of proposal locations which in turn negatively affects regression.

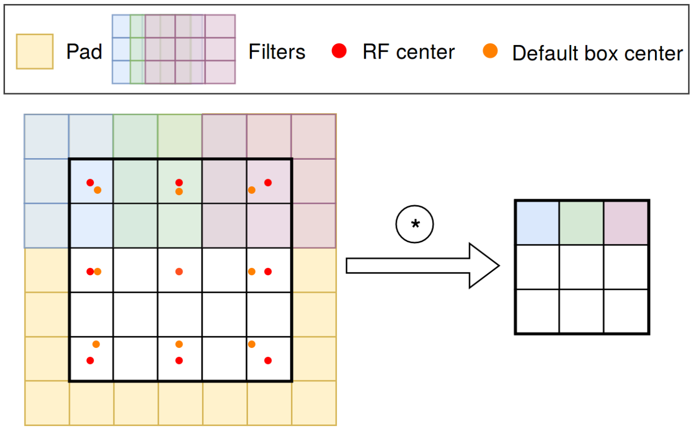


Figure 3.12: Illustration of the offsets between centers of default boxes and centers of receptive field. This simple example displays a 5×5 input image convoluted by 3×3 filters (only top three are shown) with stride $= 2$, pad $= 1$.

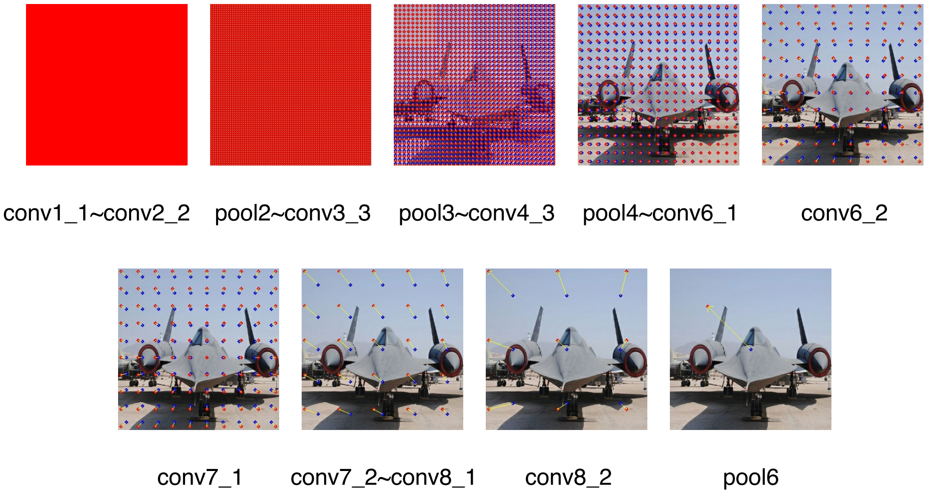
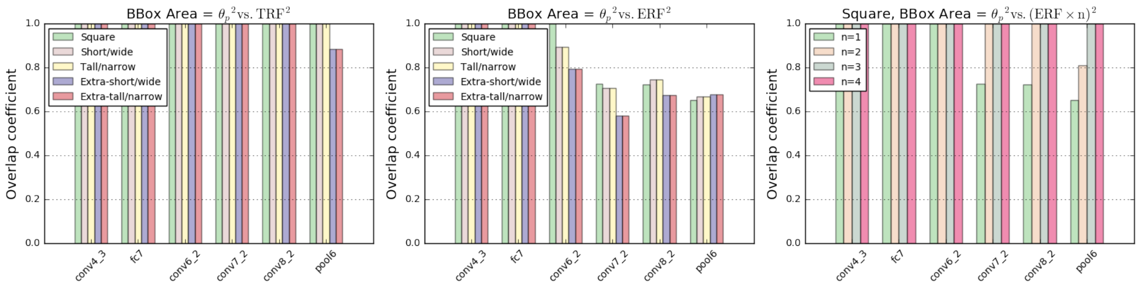


Figure 3.13: In original SSD framework [30], we observe obvious offsets (yellow arrows) between rf centers (red dots) and default box centers (blue dots).



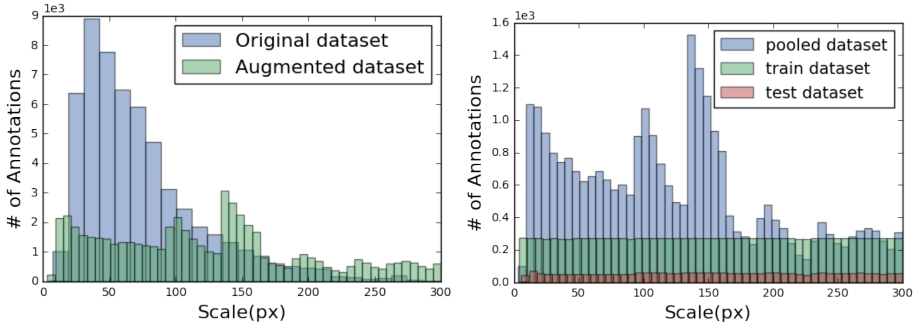
(a) (b) (c)

Figure 3.14: At each prediction layer of SSD [30], we quantitatively evaluate the degree of inaccurate estimation of proposal locations due to the offsets between boxes centered at RF centers (named square *RF boxes* with side length TRF/ERF) and default boxes. In **(a)** we assume RF boxes to have side length of TRF and calculate the overlap coefficients, and **(b)** with side length of ERF. We define the aspect ratios for group of default boxes at individual prediction layers as (also see Fig. 3.11): *Square* $(\theta_p \times \theta_p)$, *Tall/narrow* $(\sqrt{2}\theta_p \times \frac{\sqrt{2}}{2}\theta_p)$, *Short/wide* $(\frac{\sqrt{2}}{2}\theta_p \times \sqrt{2}\theta_p)$, *Extra-tall/narrow* $(\sqrt{3}\theta_p \times \frac{\sqrt{3}}{3}\theta_p)$, *Extra-short/wide* $(\frac{\sqrt{3}}{3}\theta_p \times \sqrt{3}\theta_p)$. In **(c)**, we expand ERF to its ×2, ×3 and ×4 original size, and calculate the overlap coefficient of two boxes respectively.

In order to have a quantitative measurement of this inaccuracy, in Fig. 3.14 we calculate the overlap coefficient between boxes centered at RF centers (named *RF boxes*) and default boxes. By using overlap coefficient c, we can see whether TRF/ERF covers the proposals. Otherwise, proposal location is considered inaccurate when $c < 1$. If we assume the side length of RF boxes is its corresponding TRF size, then this inaccuracy between RF boxes and default boxes associated with various aspect ratios is considered to be insignificant, as shwon in Fig. 3.14(a). However, when we use ERF size as the side length of RF boxes, then at prediction layers after *fc7* most RF boxes cannot fully cover ERFs which indicates inaccurate estimation of proposal locations, as shown in Fig. 3.14(b).

After introducing context layers in our proposed DiCSSD, at each prediction layer the ERF expands to its ×2, ×3 and ×4 (and even more depending on the number of context layers used) original size and thus ensures sufficient coverage of RF boxes against default boxes. Fig. 3.14(c) demonstrates that with the help of context layers, the overlap coefficients in last three prediction layers have now become sufficiently large with $c = 1$.

Coverage degree of grids to ground truth boxes. We investigate on the problem that at which scale the pre-computed grids[8] cover most to the ground truth boxes (i.e. with higher chances of hit). In order to do so, we firstly pool our dataset[9] with a new data augmentation technique aiming to return more widely ranging annotations, and then sample the pooled dataset so as to obtain a dataset that consists of annotations with uniformly distributed scales (Fig. 3.15). Next, we remove respectively prediction layers used to hypothesize small, medium and large-scale proposals,



(a) Pooled dataset (b) Uniformly sampled dataset

Figure 3.15: Distribution of ground truth annotation scales in DETRAC versus the same dataset after data augmentation.

[8] Also called default boxes in SSD, we use them interchangeably.

[9] To ensure a rapid prototyping capability, we use a random subset of DETRAC which has been split carefully into different scenes with various environment conditions (described in Sec. 6.2 of our submission).

Model	Description	mAP
SML	original SSD model [30]	64.5
ML	remove (a), (b)	67.3
SL	remove (c), (d)	61.7
SM	remove (e), (f)	65.8

Table 3.2: We evaluate above models in our ablation study to test the coverage degree of SSD's default boxes to the ground truth boxes. Prediction layers numbered by (a)-(f) are illustrated in Fig. 3.11. Keys: S=Small, M=Medium, L=Large.

and subsequently compute its mAP.

Our results in Table 3.2 indicate that prediction layers for medium objects are the most crucial, and removing prediction layers for small/large objects surprisingly improves mAP. Our finding suggests that adding more prediction layers with increasing number of default boxes does not necessarily improve mAP. Instead, we speculate about the detection performance in SSD to be more related to the coverage degree of grids to ground truth boxes. Therefore, given a completely new dataset, the network design of SSD should be the most effective and efficient via targeting the maximum coverage degree with the minimum number of prediction layers.

3.6.4 Our Results

Similar to SSD, we load a pre-trained model of VGGNet on the ILSVRC CLS-LOC dataset [20] and use the átrous algorithm [121] to convert *fc6* and *fc7* into conv layers. In addition, we use SGD with initial learning rate 10^{-3}, 0.9 momentum, 0.0005 weight decay and 32 batch size. The learning rate decay policy, however, is different depending on the dataset used in experiments: for DETRAC, we use an initial learning rate 10^{-3} at the first 40k iterations, then continue training with learning rate 10^{-5}

until the model is fully converged at iteration 60k. For PASCAL VOC 2007, we use the same training policy with the newest implementation of SSD [116], which uses initial learning rate 10^{-3} at first 80k iterations and continues training two rounds of 20k iterations with learning rate 10^{-4}, 10^{-5} respectively. For MS-COCO, we train our model with learning rate 10^{-3} for the first 160k iterations, followed by two rounds of 40k iterations with learning rate 10^{-4} and 10^{-5} respectively.

DETRAC. DETRAC is a challenging real-world vehicle detection dataset [105], with a total of 10 hours of videos. It consists of 60 sequences in *train* set and 40 in *test* set, which include significant differences in vehicle categories, weather, scales, occlusion ratios, and truncation ratios.

We note that ground truth for the DETRAC *test* set has not been released yet. Since we needed ground truth for our quantitative experiments, we opted to split the original 60 sequences of the DETRAC *train* set into 48 sequences that we used for training, and 12 sequences that we used as our test set. Our test set consisted of the 12 sequences numbered by *20034*, *20063*, *39851*, *40131*, *40191*, *40243*, *40871*, *40962*, *40992*, *41063*, *63521*, *63562*. Moreover, we subsampled frames with step size 10 in all sequences, leading to a total of 6349 training images and 1880 test images. To have a comprehensive evaluation of the proposed networks, we also assign to each ground truth bounding box new annotations for scale, occlusion and three difficulty levels, as shown in Table 3.3.

We compare the performance of CSSD with SSD in Table 3.4. In that table, (C)SSD533 indicate models trained with input size 533 × 300, which keeps the same aspect ratio with the original input size 940 × 540, and has been found to greatly boost the mAP of (C)SSD300 (50.3% vs. 35.1% for SSD300). Our results show that both DiCSSD and DeCSSD outperform SSD with two different input resolutions,

Scale	**Small** 0-50 pixels	**Medium** 50-150 pixels	**Large** > 150 pixels
Occlusion	**No** $< 1\%$	**Partial** $1\% - 50\%$	**Heavy** $> 50\%$
Difficulty	**Easy** Medium or large object, no occlusion, $0\% \leq$ truncation $< 15\%$	**Normal** Partial occlusion, or $15\% \leq$ truncation $< 30\%$	**Hard** Heavy occlusion, or truncation $\geq 50\%$.

Table 3.3: New annotation labels created for DETRAC, where each of them has been evaluated in Table 3.4. Bounding boxes with undefined conditions in difficulty are defaulted to *Normal*.

while DiCSSD significantly improves SSD with higher mAP than DeCSSD (+10.2% vs. +5.1% to SSD300, +5.3% vs. +1.2% to SSD533). In small object detection, DiCSSD300 has been found particularly effective, with mAP increase $> 5\%$ over both DeSSD300 and SSD300. Note that, although DeCSSD533 achieves the highest mAP in small object detection among the three, its performance on large object is down to the lowest 68.5%. This happens because, given high resolution images, top prediction layers may contain much richer feature maps, but directly reusing and fusing those maps into a single map may overweigh the training loss at the first prediction layer. Consequently, while DeCSSD outperforms the other methods in small object detection, its discriminative capabilities for large objects have been greatly constrained.

Method	Network	Overall	Category				Difficulty			Occlusion			Scale			Weather			
			Car	Van	Bus	Others	Easy	Normal	Hard	No	Partial	Heavy	Small	Medium	Large	Sunny	Rainy	Night	Cloudy
DiCSSD533	VGGNet	**55.6**	63.2	**52.1**	**81.2**	**46.3**	**63.2**	**41.0**	**12.2**	57.5	**47.4**	**26.7**	35.6	**79.9**	**84.4**	61.8	**70.6**	**53.9**	69.8
DeCSSD533	VGGNet	51.5	**64.3**	47.9	71.9	14.3	60.6	40.9	10.2	**59.2**	44.2	24.3	**39.2**	76.5	68.5	**64.0**	64.8	52.8	**72.2**
SSD533 [30]	VGGNet	50.3	58.7	45.6	75.8	37.0	61.3	34.1	7.1	56.5	38.9	17.4	31.1	74.2	81.7	55.6	64.0	52.5	63.8
DiCSSD300	VGGNet	**45.3**	**53.0**	**39.5**	**71.5**	19.4	**61.9**	**25.0**	**9.8**	**50.9**	32.5	**24.0**	**19.6**	**72.5**	**78.7**	**48.5**	**60.7**	**48.1**	**54.6**
DeCSSD300	VGGNet	40.2	47.8	37.4	64.6	11.4	57.3	21.1	7.2	47.8	30.3	17.6	14.7	67.8	63.3	41.2	55.1	45.7	53.0
SSD300 [30]	VGGNet	35.1	43.5	24.8	50.2	4.5	53.2	19.0	6.5	42.8	26.6	16.2	14.2	60.2	59.4	42.0	42.3	39.4	51.6
SSD300 [30]	ResNet152	35.2	42.3	33.5	54.3	4.5	52.8	17.5	7.1	41.1	**34.7**	17.6	6.8	62.9	60.2	36.8	50.8	30.5	44.4
SSD300 [30]	ResNet101	39.1	42.8	32.6	69.5	**30.2**	55.4	16.4	7.1	43.2	30.4	17.5	8.1	63.1	69.5	40.0	48.0	44.2	46.5
SSD300 [30]	ResNet50	25.1	42.4	30.5	59.8	26.9	46.1	4.8	0.0	35.9	0.4	0.0	7.4	63.2	63.8	44.7	0.0	0.0	0.0

Table 3.4: Evaluation of the proposed networks on DETRAC dataset.

In summary, DiCSSD was found to be the most effective compared to others in DETRAC, with both low and high-resolution input size. Later on, we will see that the run-time speed of DiCSSD is comparable to SSD, which proves the efficiency of our proposed method. Visualization results on DETRAC can be found at Fig. 3.18.

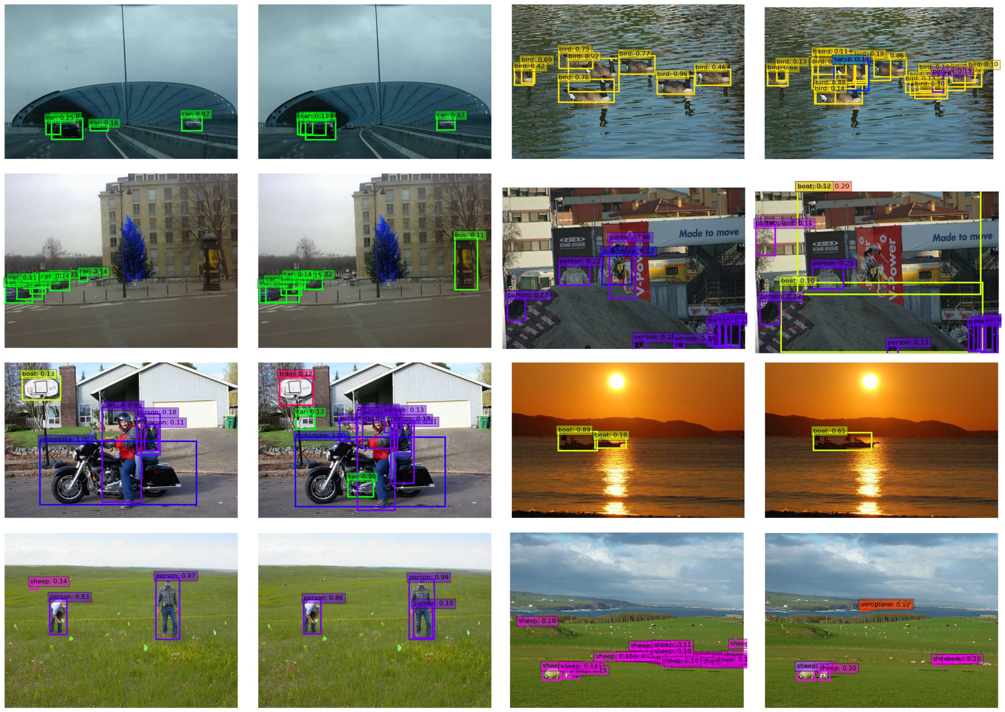


Figure 3.16: Curated examples of DiCSSD (left) and SSD (right) on VOC07 *test* set.

PASCAL VOC 2007. In this dataset, we train each model with a union of VOC2007 *trainval* set and VOC 2012 *trainval* sets, and evaluate on VOC 2007 *test* set. Again, our results in Table 3.5 indicate that both DiCSSD and DeCSSD outperform SSD (+0.6% vs. +0.1%). We note that the performance of SSD has been improved greatly with its new expansion data augmentation trick [116]. Still, our proposed context layers, applied on top of this improved SSD, further boost performance and take little memory consumption. It is noteworthy that among all the 20 categories evaluated, mAPs of DiCSSD are higher than those of SSD in 15 categories. We also evaluate the performance of DiCSSD in Fig. 3.17 using [119]. Again DiCSSD is better than SSD for categories that give the most false positives including airplane, bird, cat and chair. Visualization results can be found at Fig. 3.16.

Method	Network	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
DiCSSD300*	VGGNet	**78.1**	**82.2**	**85.4**	**76.5**	69.8	**51.1**	86.4	**86.4**	88.0	61.6	82.7	76.4	**86.5**	**87.9**	**85.7**	78.8	**54.2**	76.9	77.6	**88.9**	**78.2**
DeCSSD300*	VGGNet	77.6	79.9	84.7	76.4	**70.2**	48.2	86.5	86.1	**88.9**	**61.7**	**83.1**	76.8	86.1	87.4	85.3	78.8	52.0	77.0	79.1	87.0	77.2
SSD300* [116]	VGGNet	77.5	79.5	83.9	76.0	69.6	50.5	**87.0**	85.7	88.1	60.3	81.5	**77.0**	86.1	87.5	84.0	**79.4**	52.3	**77.9**	**79.5**	87.6	76.8
DSSD321 [114]	ResNet101	78.6	81.9	84.9	80.5	68.4	53.9	85.6	86.2	88.9	61.1	83.5	78.7	86.7	88.7	86.7	79.7	51.7	78.0	80.9	87.2	79.4
ION [120]	VGGNet	75.6	79.2	83.1	77.6	65.6	54.9	85.4	85.1	87.0	54.4	80.6	73.8	85.3	82.2	82.2	74.4	47.1	75.8	72.7	84.2	80.4
R-FCN [109]	ResNet101	80.5	79.9	87.2	81.5	72.0	69.8	86.8	88.5	89.8	67.0	88.1	74.5	89.8	90.6	79.9	81.2	53.7	81.8	81.5	85.9	79.9
Faster [28]	ResNet101	76.4	79.8	80.7	76.2	68.3	55.9	85.1	85.3	89.8	56.7	87.8	69.4	88.3	88.9	80.9	78.4	41.7	78.6	79.8	85.3	72.0
Fast [13]	VGGNet	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
Faster [29]	VGGNet	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6

Table 3.5: Detection results on PASCAL VOC2007 *test* set. All models were trained with VOC2007 *trainval* set + VOC2012 *trainval* set. (C)SSD300* and SSD512* are the latest SSD models with the new expansion data augmentation trick [116].

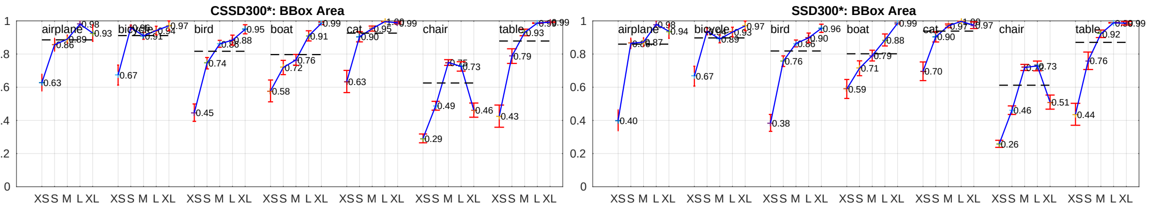


Figure 3.17: Sensitivity analysis of bounding box area using [119]. Keys in *BBox Area*: XS=extra-small, S=small, M=medium, L=large, XL=extra-large.

MS-COCO 2015. In order to evaluate CSSD on a more general, large-scale object detection dataset, we compare our proposed model with both SSD , DSSD and many others on MS-COCO 2015 dataset. To directly compare CSSD with SSD, we use the same *trainval35k* training set [120] and follow the same training policy to SSD300* [116]. In Table 3.6, we show our detection results on MS-COCO [104] *test-dev2015* set. Both DiSSD300* and SSD300* use the new expansion data augmentation trick [116], while SSD300 is with the original SSD implementation [30].

Method	Train Set	Network	FPS	Avg. Precision, IoU:			Avg. Precision, Area:			Avg. Recall, #Dets:			Avg. Recall, Area:		
				0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
DiCSSD300*	trainval35k	VGGNet	40.8	**26.9**	**46.3**	**27.7**	**8.2**	**27.5**	**43.4**	**25.0**	**37.3**	**39.8**	**15.4**	**43.1**	**60.0**
SSD300* [116]	trainval35k	VGGNet	46	25.1	43.1	25.8	6.6	25.9	41.4	23.7	35.1	37.2	11.2	40.4	58.4
SSD300 [30]	trainval35k	VGGNet	46	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
DSSD321 [114]	trainval35k	ResNet101	9.5	28.0	46.1	29.2	7.4	28.1	47.6	25.5	37.1	39.4	12.7	42.0	62.6
R-FCN [109]	trainval	ResNet101	9	29.9	51.9	-	10.8	32.8	45.0	-	-	-	-	-	-
ION [120]	train	VGGNet	-	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster [29]	trainval	VGGNet	7	21.9	42.7	-	-	-	-	-	-	-	-	-	-
Fast [13]	train	VGGNet	-	19.7	35.9	-	-	-	-	-	-	-	-	-	-

Table 3.6: MS-COCO *test-dev2015* detection results.

Our proposed network DiCSSD has been shown to greatly boost the performance of SSD (+3.2% given IoU=0.5), and even better than DSSD (+0.1% given IoU=0.5) which uses ResNet101 as its base network but only achieves 9.5fps (compared to 40.8 fps of ours). We also note that for small object, DiCSSD achieves better precision than the newest SSD (8.2% vs. 6.6%), in addition to the highest recall (15.4%) among all the benchmarked methods. Visualization results can be found at Fig. 3.19.

Inference time. In Table 3.7, we compare the speed and accuracy of benchmarked models on PASCAL VOC 2007. The proposed DiCSSD network achieves the highest mAP, while maintaining a real-time speed of 40.8 fps. Therefore, DiCSSD has been the most effective model among all models compared. Note that we calculate fps with batch size = 1, preprocessing time counted and cuDNN [122] enabled (v5.1 for Titan X, v4 for K40 and GTX 980).

Method	mAP	Network	FPS			Input Resolution
			K40	GTX 980	Titan X (Pascal)	
SSD300* [116]	77.5	VGGNet	16.2	36.2	46	300x300
DiSSD300*	78.1	VGGNet	12.2	24.3	40.8	300x300
DeSSD300*	77.6	VGGNet	8.7	18.4	39.8	300x300
DSSD [114]	78.6	ResNet101	-	-	9.5	321x321
YOLO [111]	66.4	VGGNet	-	-	21	448x448
Faster R-CNN [13]	73.2	VGGNet	-	-	7	∼1000x600
Faster R-CNN [28]	76.4	ResNet101	-	-	2.4	∼1000x600
R-FCN [109]	80.5	ResNet101	-	-	9	∼1000x600

Table 3.7: Speed and accuracy comparisons on VOC2007.

3.6.5 CSSD Curations

We show in Fig. 3.18 and Fig. 3.19 some visualization results of CSSD (particularly, DiCSSD) comparing to SSD.

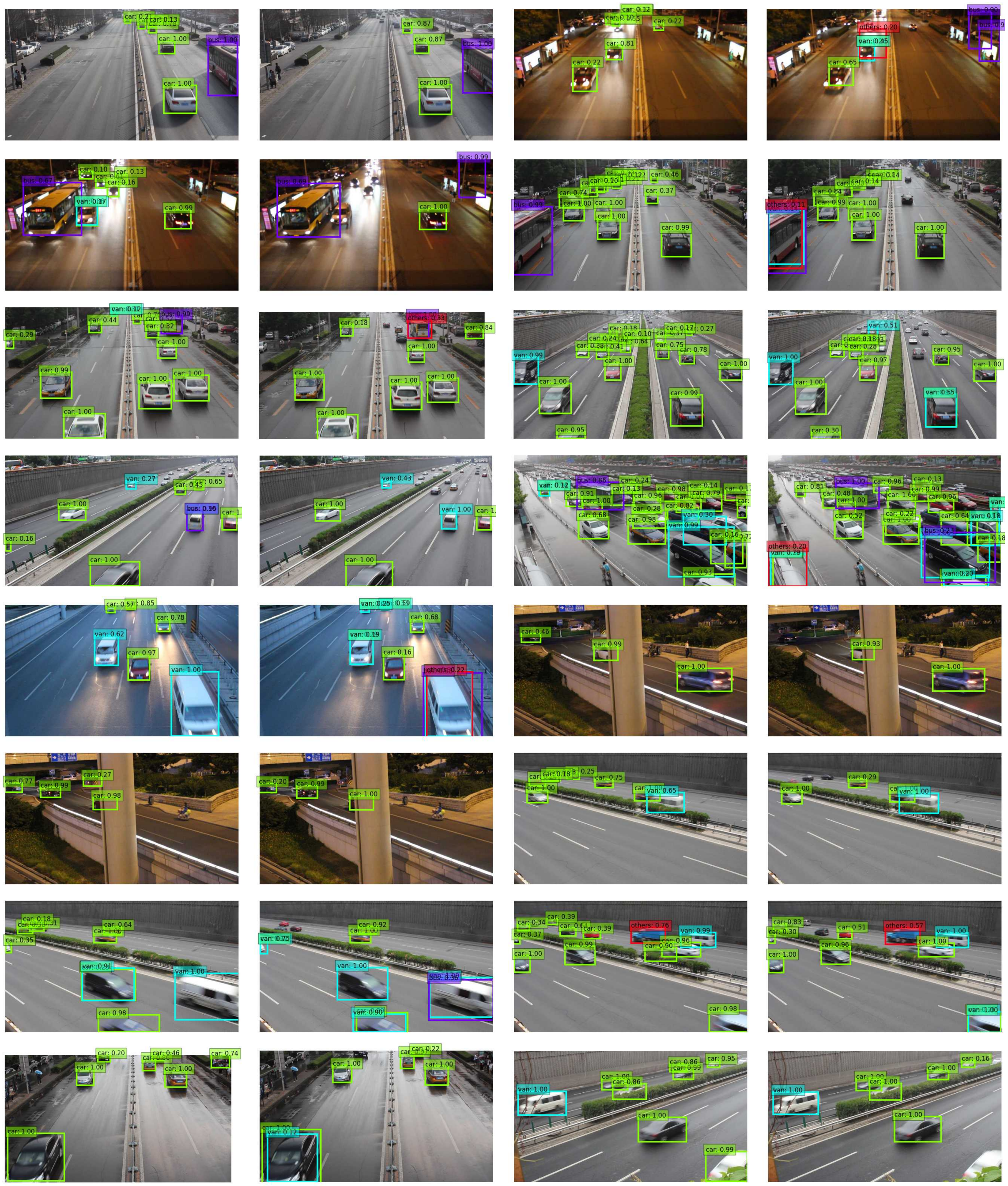


Figure 3.18: Curated examples of DiCSSD (left) and SSD (right) on DETRAC *test* set.

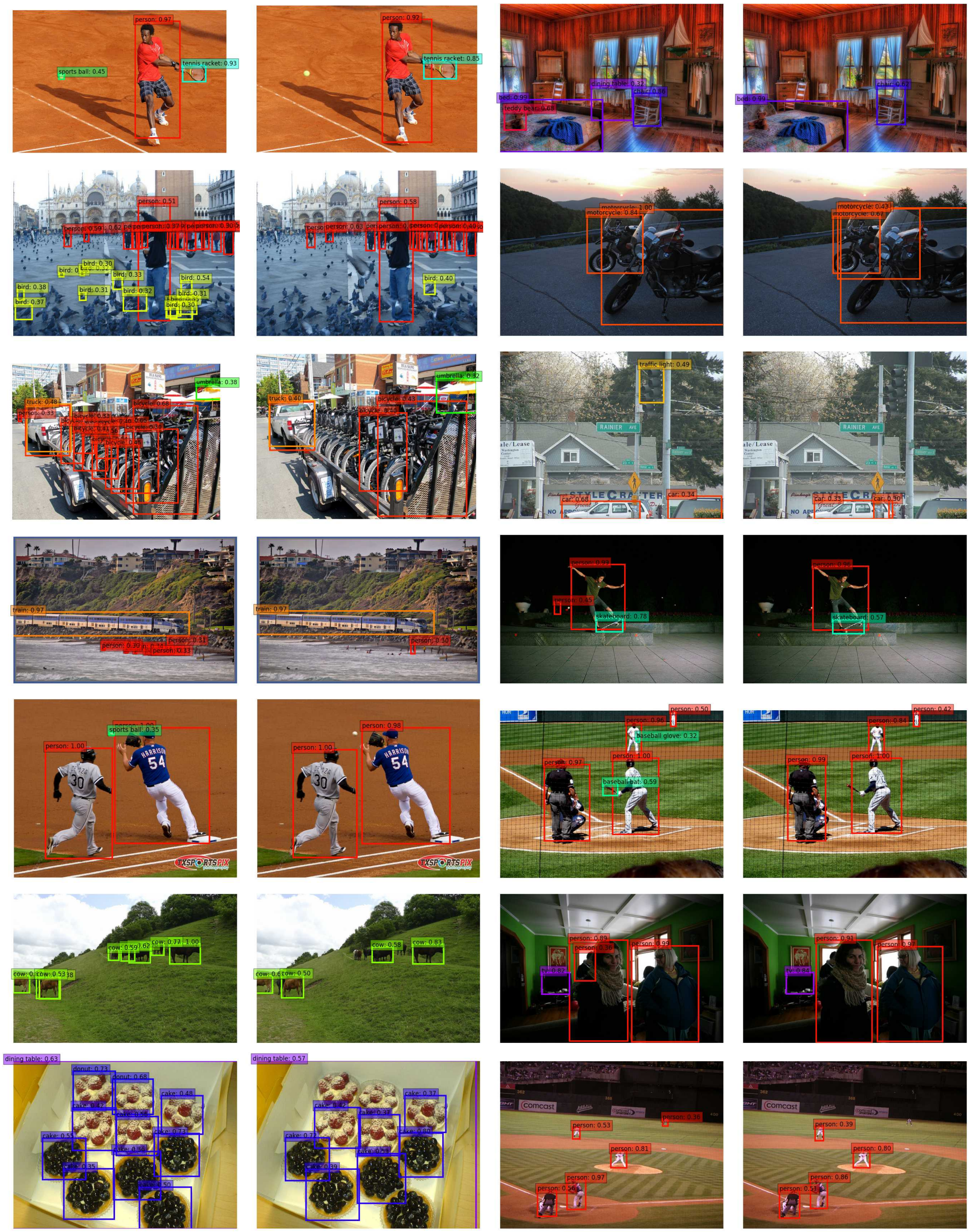


Figure 3.19: Curated examples of DiCSSD (left) and SSD (right) on MS-COCO *test-dev2015* set.

CHAPTER 4

SEMANTIC SEGMENTATION

This chapter gives an overview of deep learning based semantic segmentation methods. To fill the gap between high-quality prediction and real-time performance, this chapter proposes an effective but yet efficient semantic segmentation method called ThunderNet. ThunderNet runs faster than the current fastest model in the literature ENet [8], with up to 1.7x speedup on a desktop machine and 1.2x on an embedded system, while performing significantly better than ENet.

4.1 Introduction

Image semantic segmentation is a fundamental problem in computer vision. Its main task is to perform dense predictions over all pixels and output categories belonging to each. Semantic segmentation has been in the long run treated as a crucial part in achieving deep understandings of the image, with topics include, but not limited to instance segmentation, scene parsing, and human object interactions etc. [123, 124, 125]. In the past years, deep learning approaches [15, 17, 16, 126, 127, 128] have made fruitful progress in semantic segmentation, with the development of Convolutional Neural Networks (CNNs) and many emerging technologies related to it.

However, recent advances in semantic segmentation with CNNs largely depend on those deep and wild backbones, with dedicated designs of various bottlenecks and many other meta-architectures. Taking those sophisticated designs results in a large amount of redundant overhead, regarding to the number of operations in order to make dense predictions. In Fig. 4.1, we show the accuracy of state-of-the-art methods on Cityscapes dataset [129], in addition to their runtime speed. By increasing network complexity, most research [16, 15, 127, 31, 17, 130, 128, 126] in the past few year has focused on generating high-quality predictions, and thus inevitably slower the inference speed to a great extent. Several works in the literature have been proposed to overcome the speed issue, including SQ [131], ENet [8], ICNet [132] and the most recent ERFNet [133]. They aim at improving the inference speed while attempting to maintain comparable accuracy. Nonetheless, as shown in Fig. 4.1, few of these methods have achieved real-time speed with satisfactory performance (i.e. above 65% mIoU on Cityscapes). Under such circumstances, real-time semantic segmentation on embedded systems (mobile platforms, NVIDIA Jetson TX1/TX2, etc.) has become a fundamentally crucial, but very challenging task.

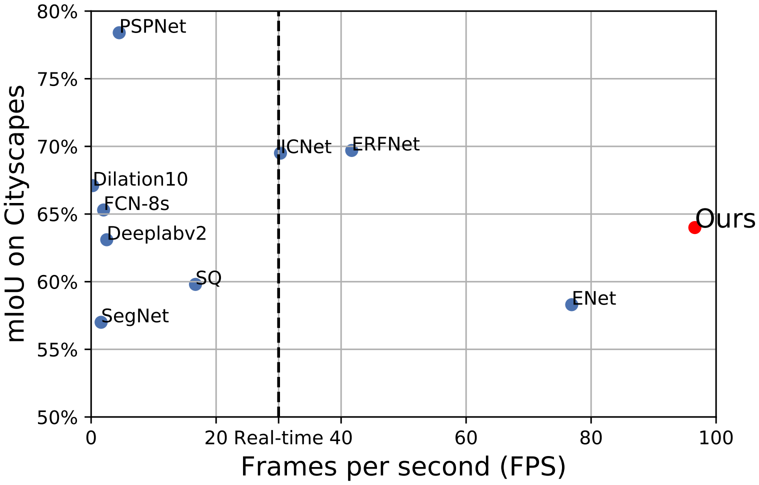


Figure 4.1: Accuracy (mIoU %) vs. inference speed (fps) for the state-of-the-art methods on Cityscapes, including SegNet [16], FCN-8s [15], Dilation10 [127], SQ [131], Deeplabv2 [17], PSPNet [31], ENet [8], ICNet [132], ERFNet [133]. Our proposed ThuderNet is fastest and has obtained comparable accuracy to a majority of those methods.

In this chapter, we focus on this challenging task and propose a fast and efficient lightweight network that achieves 64% mIoU on Cityscapes with 96.2 fps, which greatly boosts the previous methods regarding to the trade-off between accuracy and runtime speed. Our method, called Turbo Unified Network (ThunderNet), unifies both pyramid pooling module [31] and decoder structure, and builds upon a minimum backbone truncated from ResNet18 [28]. In summary, our main contributions are:

(1) We present a novel and extremely lightweight network (ThunderNet) that achieves high accuracy of 64% mIoU on Cityscapes[1] with single-scale test, in addition to significantly faster inference speed (96.2 fps on Titan XP, input size 512x1024).

(2) Comparing to those real-time methods currently available in the literature, ThunderNet does not utilize any bottleneck modules and thus enables easy and flexible combinations with many popular meta-architectures and designs. We showcase that by incorporating those meta-architectures directly, we can improve our model further but with runtime speed penalized to a certain extent, depending on the preferences and choices.

(3) ThunderNet follows the *encoder-decoder* architecture [16] which allows for training in an end-to-end fashion. Without bells and whistles (those non-trivial training techniques for e.g. transfer learning, alternative training and auxiliary loss), ThunderNet can be trained in a simple setting and converges only in hours.

(4) We test the proposed ThunderNet on NVIDIA Jetson TX2, which achieves 20.9 fps (input size 256x512) with large speedup of inference time on embedded systems– up to 1.2x over ENet [8] and 1.8x over ERFNet [133] respectively.

[1] `https://www.cityscapes-dataset.com/method-details/?submissionID=1073`

4.2 Related Work

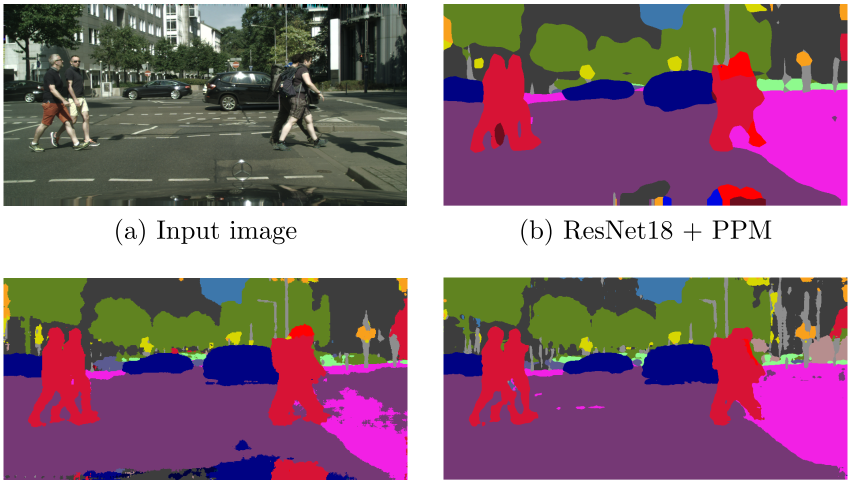
After CNN-based methods [7, 134] made a significant breakthrough in image classification [135], Long et al. [15] pioneered the use of CNNs in semantic segmentation. The proposed FCN firstly perform end-to-end semantic segmentation by replacing last fully-connected layers with convolution layers. FCN's output were very coarse due to the successive use of pooling layers that had been previously found useful in image classification for the integrated context. In semantic segmentation, however, the lost of fine details with pooled feature maps will generally result in coarse predictions. With skip connections, therefore, FCN refined the outputs by fusing them with the feature maps from shallower layers. Afterwards, SegNet [16] followed *encoder-decoder* architecture of FCN. But different from FCN where upsampling was done using deconvolution, SegNet saved indices from pooling layers in order to upsample feature maps at its decoder. As the progressively enlarged feature maps learn both coarse geometries and fine boundaries, encoder-decoder architecture has become very popular thereafter [136, 8, 137, 133, 138, 139]. In object detection, the encoder-decoder architecture has also gained its popularity (for e.g. FPN [140]).

There also exists many other meta-architectures and designs that have been found particularly useful in the literature of semantic segmentation. Deeplab family (v1 to v3+) [141, 17, 130, 139] used atrous spatial pyramid pooling (ASPP) which captured image context at multiple scales. In ASPP, dilated convolution plays a key role to remedy the shortcoming of fine features and structures due to the progressive downscaling of feature maps at classic networks like VGG and ResNet [28], where more related studies can be found at [127] and [138]. In addition, RefineNet [142] consists of a multi-path framework that runs pyramid of input image at different scales. In RefineNet, both short-rang and long-range shortcut connections were exploited in order to enhance the fusion results from low and high-resolution feature maps. Recently, Zhao et al. [31] propose PSPNet that has become one of the state-of-the-art models in semantic segmentation. In PSPNet, a pyramid pooling module (PPM) is applied to generate representations at different levels, whose output will be upsampled and concatenated along feature map channels to form the final representation. By doing so, the final predictions will carry both local and global contextual information at various levels. However, due to the pooling layers and strided convolution[2] layers within the backbone network, detailed boundaries are still missing. After realizing this problem, the most recent work Deeplabv3+ [139] chooses Xception [83] as its backbone, and involves both ASPP to extract enriched semantic information and a simple yet decoder to recover the detailed object boundaries.

Aforementioned methods opt to use heavy backbones and thus require more computing resources. In Fig. 4.1, we demonstrate that only few of them has reached the real-time inference speed. On mobile platforms like smart phones in addition to those GPU-powered embedded systems like NVIDIA Jetson TX1/TX2 etc., applying above methods becomes unfeasible. As there exists a big gap towards real-time segmentation, some recent works try to alleviate the problem by developing a simplified, lightweight network with bottleneck modules. For instance, ENet [8] was proposed using an extreme setting of bottlenecks which results in underfit to the dataset (it only achieves 58.3% on Cityscapes). ICNet [132] used input from multi-scales and trained with each scale different number of convolution layers. ICNet achieves high accuracy on Cityscapes (69.5%) with over 30 fps on a Desktop-level GPU. Implementation of ICNet, however, is non-trivial because of the auxiliary losses used at all scales and the dedicated design of sharing weights between the smallest and middle scale. The newest work ERFNet [133], on the other hand, achieves much better results (69.7%)

[2] In this chapter, the strided convolution denotes the convolution operations with stride size greater than or equal to 2.

than ENet with special bottlenecks composed of factorized convolutions. It is because more context are captured in ERFNet's encoder with convolutions dilated by the increasing rates, meanwhile ERFNet's three-level decoder itself is capable to learn more boundary details. However, the alternative training used in ERFNet is not easy to realize. Besides, due to that its backbone is fully assembled with bottlenecks, it becomes very hard to extend this work with other meta-architectures.



(a) Input image

(b) ResNet18 + PPM

(c) ResNet18 + decoder

(d) ResNet18 + PPM + decoder

Figure 4.2: We start with a simplified experiment that by adding the pyramid pooling module (PPM) into a ResNet18 backbone, after convergence the output looks quite smooth for geometry shapes, but losing clear boundaries (b). Instead, by adding a four-level decoder (symmetric to encoder's $2^4 = 16$ total stride) to the same backbone, the output shows more detailed object boundaries, but its shape is hardly preserved (c). Inspired by this finding, we unify the two meta-architectures, after which the final results look much better with both clear geometry shapes and sharp boundaries (d).

Conclusively, both dilated convolution and pyramid pooling module aim at capturing contextual information at various levels, while adding decoder to a classic backbone used for image classification will enhance boundary details. Inspired by this finding and related experiment illustrated in Fig. 4.2, in this chapter, we propose to unify the two meta-architectures–PPM and decoder, into single network. Compared to Deeplabv3+ [139] and ERFNet [133], our proposed ThuderNet builds on a minimum backbone truncated only from a popular ResNet18 network, without using any bottleneck modules. Due to that there are only ∼4.7m parameters in ThunderNet and they are all from standard convolution layers[3], its inference speed can be as fast as 96.2 fps (on a Titan XP), given input size 512x1024.

4.3 Framework

We propose a lightweight network called ThunderNet and illustrate its framework in Fig. 4.3. ThunderNet mainly consists of an encoder, a pyramid pooling module and a customized decoder. As shown in Fig. 4.3, for encoder, we opt to use a ResNet18 [28] network that has been truncated away from its 4th block. For pyramid pooling module, we follow the same setting with PSPNet [31] but further reduce the bottlenect convolution layer (the one after bilinear upsampling) to 1x1 convolution with 256 channels (instead of 3x3 convolution with 512 channels in [31]). For decoder, we use our customized decoder which consists of two consecutive deconvolution upsampling that both appends to and followed by a 1x1 convolution layer.

The proposed ThunderNet is simply, but yet very efficient for the task of image semantic segmentation. Firstly, we realize that, in order to have a network with fast inference speed, we must have a lightweight encoder that consists of fewer 3x3 convolution layers with smaller number of output channels, compared to those networks similar to the VGG16 implementation [7]. According to an online study [4], ResNet18

[3] Compared to standard convolutions, performing convolutions in bottleneck modules will consume much more runtime memory and thus take longer time, due to the largely expanded number of operations.

[4] https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

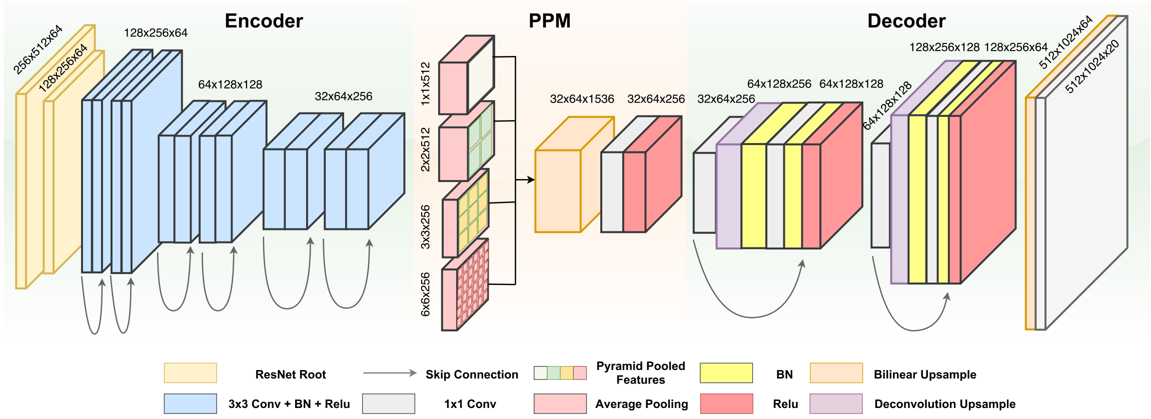


Figure 4.3: Framework of our proposed ThunderNet. ThuderNet follows encoder-decoder architecture, where its encoder mainly consists of a ResNet18 backbone truncated away from its 4th block, and its decoder implemented as a customized, two-level consecutive upsampling network. In between, the pyramid pooling module (PPM) is added in order to capture the contextual information from different levels.

is one of those very efficient backbones regarding to the number of operations it takes versus its performance on ImageNet. Moreover, in Table 4.4, we also compare the performance of our truncated ResNet18 (ResNet18-3b) to the original ResNet18 (ResNet18-4b), which solidifies the reason for our selection of ResNet18-3b model.

Secondly, not only the import of PPM helps to visually enhance the smoothness of object geometries (Fig. 4.2b), but also it boosts our quantitative segmentation results, as shown in Table 4.5. By appending a FPN-alike decoder, the boundary details can be learned more and thus improving our segmentation results further.

Thirdly, all convolution layers used in ThunderNet contain merely standard convolution operations, either 1x1 or 3x3. This easy realization of our network enables fast prototyping, as well as flexible combinations with other meta-architectures and designs, as illustrated in Sect. 3.6.

Last but not least, due to the fully-optimized matrix adds/multiplications with standard convolutions under a Desktop-level GPU like Titan X/XP, ThunderNet's dedicated design of using merely standard convolution layers will benefit significantly more than those networks using bottleneck modules (for e.g. ENet [8] and ERFNet [133]). We show in Sect. 3.6 that such advantage of ThunderNet will make it an extremely fast network. With Titan XP, the inference speedup can be up to x2 compared to the currently fastest network ENet [8].

4.4 Experiments

4.4.1 Implementation

In this chapter, we use PyTorch for easily prototyping our models and measure the inference speed with the exactly same model implemented in Caffe[5]. We follow a similar training policy to ERFNet [133], by using Adam optimizer [144] with learning rate of $5e^{-4}$, momentum of 0.9, and weight decay of $1e^{-4}$. For all experiments, we load a pre-trained ImageNet model of ResNet18 directly into ThunderNet, thus abandoning weights of all layers at its 4th block. In order to overcome class imbalancing problem on Cityscapes, we use the same class weights as [133] when training alternatively ERFNet's decoder part, i.e. weights of $\{2.81, 6.98, 3.78, 9.94, 9.77, 9.51, 10.31, 10.02, 4.63, 9.56, 7.86, 9.51, 10.37, 6.66, 10.26, 10.28, 10.28, 10.40, 10.13, 0\}$ for all 19 classes. Besides, the deconvolution layers adopted in ThunderNet's decoder are implemented with kernel size of 3, stride 1 and padding 1, followed by an explicit padding of 1 around feature map outputs. Finally, we note that all our results use single-scale test, which makes our evaluation unbiased in real scenario. In addition, we report only the best result for all training models, with batch size 48 and maximum training epoch of 150 on Cityscapes.

[5] In Caffe, we use a commonly adopted technique to accelerate the inference speed, by merging batch normalization, scale and relu layers into their preceding convolution layer [143].

Method	Sub	mIoU (%)	Time (ms)	Frame (fps)
SegNet [16]	4	57.0	60	16.7
CRF-RNN [145]	2	62.5	700	1.4
DeepLabv2 [17]	2	63.1	400	2.5
FCN-8S [15]	no	65.3	500	2
Adelaide [128]	no	66.4	35000	0.03
Dilation10 [127]	no	67.1	4000	0.25
SQ [131]	no	59.8	60	16.7
ICNet [132]	no	69.5	33	30.3
ENet [8]	2	58.3	13	76.9
ERFNet [133]	2	69.7	24	41.7
ThunderNet	2	64.0	10.4	96.2

Table 4.1: Final accuracy (mIoU) and speed comparisons on Cityscapes test set.

4.4.2 Results on Cityscapes

We evaluate our proposed ThunderNet on Cityscapes dataset [129], which has a total of 19 classes and contains a train set of 2975 images, a validation set of 500 images, both with ground truth publicly available to download, in addition to a test set of 1525 images whose ground truth data are not available. In order to have our model evaluated on the test set, we will have to submit our results to an online testing server. Therefore, by comparing our results on both val and test set, it allows for the models present in this chapter to show clear signs of overfit/underfit. Note that for all the experiments present in this chapter, we do not use any additional coarse annotations from Cityscapes for our training.

All accuracies reported in this chapter use the common *Intersection-Over-Union (IoU)* metric:

$$\mathrm{IoU} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}, \tag{4.1}$$

where TP, FP, FN denotes respectively the number of true positive, false positive and false negative pixel-wise predictions. The IoU metric is designed for a specific class. After averaging IoUs over all classes, we will have a fair evaluation metric namely the *mean of class-wise IoU (mIoU)*, which indicates the overall performance of our model. Another metric that appears in this chapter is *pixel-wise accuracy (Pixel Acc)*, which takes additional TN (true negative) into account with all pixels:

$$\text{Pixel Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{4.2}$$

We show our final results on Cityscapes test set in Table 4.1. Our proposed ThunderNet has become the fastest network with comparable accuracy to a majority of the methods benchmarked. Compared to the currently fastest network in the literature ENet [8], our model has achieved much better results with more than 5.7% mIoU. When benchmarked with those methods aiming at high-quality predictions including SegNet [16], CRF-RNN [145] and DeepLabv2 [17], our method not only shows better accuracy, but also runs at significantly faster inference speed.

4.4.3 Performance Analysis

In Table. 4.2 and Table. 4.3, we compare ThunderNet with those fastest networks currently available in the literature. Comparing to [133], we have enriched the experiments by additionally testing ENet, ICNet and our ThunderNet on both NVIDIA Jetson TX2 platform and Titan XP GPU. We conducted experiments for all above three methods in Caffe, with CUDA 8.0 and cuDNN 7.0 under Titan XP, CUDA 9.0 and cuDNN 7.0 under Jetson TX2 (using JetPack 3.2, L4T R28.2).

Same with [132], in order to have a fair evaluation of the inference speed testings under Jetson TX2 and Titan XP, we use Caffe's time measure tool *Caffe time* and repeat the forward-backward operations for 100 times to reduce variances during testing. It is noteworthy that as ENet published their code originally in PyTorch, we used its caffe version publicly available at [146], which has been marked as ENet*.

As illustrated in Table. 4.2 and Table. 4.3, our experimental results show that ThunderNet outperforms ENet at various input resolutions (except at the resolution

Model	NVIDIA TEGRA TX1 (Jetson)						NVIDIA Titan X					
	480x320		640x360		1280x720		480x320		640x360		1280x720	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
SegNet [16]	757	1.3	1251	0.8	-	-	69	14.6	289	3.5	637	1.6
ENet [8]	47	21.1	69	14.6	262	3.8	7	135.4	21	46.8	46	21.6
SQ [131]	60	16.7	86	11.6	389	2.6	N/A					
ERFNet [133]	93	10.8	141	7.1	535	1.9	12	83.3	41	24.4	88	11.4
	NVIDIA TEGRA TX2 (Jetson)						NVIDIA Titan XP					
ENet* [8]	62.9	15.9	87.4	11.4	**298.6**	**3.3**	8.2	122	9.5	105.3	23.1	43.3
Ours	**56.8**	**17.6**	**82.9**	**12.1**	304.0	3.3	**4.8**	**208**	**6.0**	**166.7**	**15.7**	**63.7**

Table 4.2: Comparison of inference speed for fastest models currently available, given different input sizes of 480x320 (HVGA), 640x360 (nHD) and 1280x720 (HD) respectively. ENet indicates its original performance analysis reported in [8], while ENet* indicates its Caffe implementation that has been tested in the same environment with our proposed ThunderNet.

Model	NVIDIA TEGRA TX1 (Jetson)						NVIDIA Titan X					
	512x256		1024x512		2048x1024		512x256		1024x512		2048x1024	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
ENet [8]	41	24.4	145	6.9	660	1.5	7	142.9	13	76.9	49	20.4
ERFNet [133]	85	11.8	310	3.2	1240	0.8	8	125	24	41.7	89	11.2
	NVIDIA TEGRA TX2 (Jetson)						NVIDIA Titan XP					
ICNet [132]	327.2			3.1			45.5			25.3		
ENet* [8]	56.3	17.8	175.3	5.7	891.9	1.1	8.0	125.0	15.3	65.3	46.8	21.4
Ours	**47.9**	**20.9**	**171.0**	**5.8**	**682.7**	**1.5**	**4.7**	**212.8**	**10.4**	**96.2**	**30.2**	**33.1**

Table 4.3: Comparison of inference speed for ENet, ERFNet, ICNet and our ThunderNet, given different input sizes of 512x256, 1024x512 and 2048x1024 respectively.

of 1280x720, where ThunderNet is only 6.6 ms slower than ENet). ThunderNet runs significantly faster than ENet (about ∼x1.5 faster across all resolutions), particularly under Titan XP. This is due to that ThunderNet performs merely standard convolution operations that have been fully optimized with a Desktop-level GPU like Titan XP, instead of bottlenecks used in ENet that will have to be expanded rapidly during forwarding (thus benefit less regarding to those matrix optimizations). Therefore, the optimization adopted in those powerful GPUs will have to be downgraded under embedded platforms like NVIDIA Jetson TX2. With Jetson TX2, Our proposed ThunderNet achieves much better results (64 % vs. 58.7%), and is still x1 to x1.2 faster than ENet.

Regarding to the real-time applications of deep learning based models for semantic segmentation, under embedded systems ThunderNet can be as fast as 20.9 fps with input size 256x512. With 640x360 (nHD) resolution, ThunderNet runs at 12.1 fps which is sufficiently fast for most applications even in autonomous driving. Furthermore, even given the full-size (1024x2048) Cityscapes input images, by utilizing GPU like Titan XP, ThunderNet has already passed the real-time requirement with 33.3 fps.

4.4.4 Ablation Studies

Backbone Selection. We started our work with ResNet18 backbone tests in Table 4.4. These results show that given input size 512x512, ResNet18-3b backbone achieves the best speed and accuracy trade-off, compared to all other options. Different from other experiments, in this experiment we measured the fps using Tensorflow and adopt a similar data augmentation method to PSPNet [31], i.e. a random mirror and a fixed-size crop after aspect ratio preserving resize with smaller side length randomly sampled between the crop size (384/512/783) and 1.5x the smaller size of Cityscapes input images ($\sim$1500). To ensure the convergence of our tests models, we used Stochastic Gradient Descent (SGD) with learning rate 0.01 and batch size 8. All results are reported with models converged at sufficiently large training iteration $\sim$100k.

PPM and Decoder Unification. Following the training configurations described in Sect. 3.6, we perform the basic ablation studies by adding PPM and decoder, respectively. Table 4.5 demonstrates that adding either PPM and decoder improves our ResNet18-3b backbone to a certain extent (1.1% and 2.3% respectively), while unifying the two meta-architectures significantly boosts the performance by 4.75%.

Backbone	Input Size 384x384			Input Size 512x512			Input Size 783x783		
	FPS	Pixel Acc % (val)	mIoU % (val)	FPS	Pixel Acc % (val)	mIoU % (val)	FPS	Pixel Acc % (val)	mIoU % (val)
ResNet18-3b	147.5	74.30	43.39	**94.0**	**76.80**	**46.48**	50.8	78.59	48.01
ResNet18-4b	104.5	74.94	45.33	71.5	76.37	46.73	39.0	78.82	48.16

Table 4.4: Experimental results of using ResNet18 backbones with different input sizes on Cityscapes val set. We validate the effectiveness of our backbone by simply appending bilinear upsampling layer to our ResNet18 truncated backbone (ResNet18-3b) and the original backbone (ResNet18-4b), whose output will be finally converted to logits.

Model	ResNet18-3b	ResNet18-3b + PPM	ResNet18-3b + decoder	ResNet18-3b + PPM + decoder
mIoU %	60.02	61.18	62.31	64.77

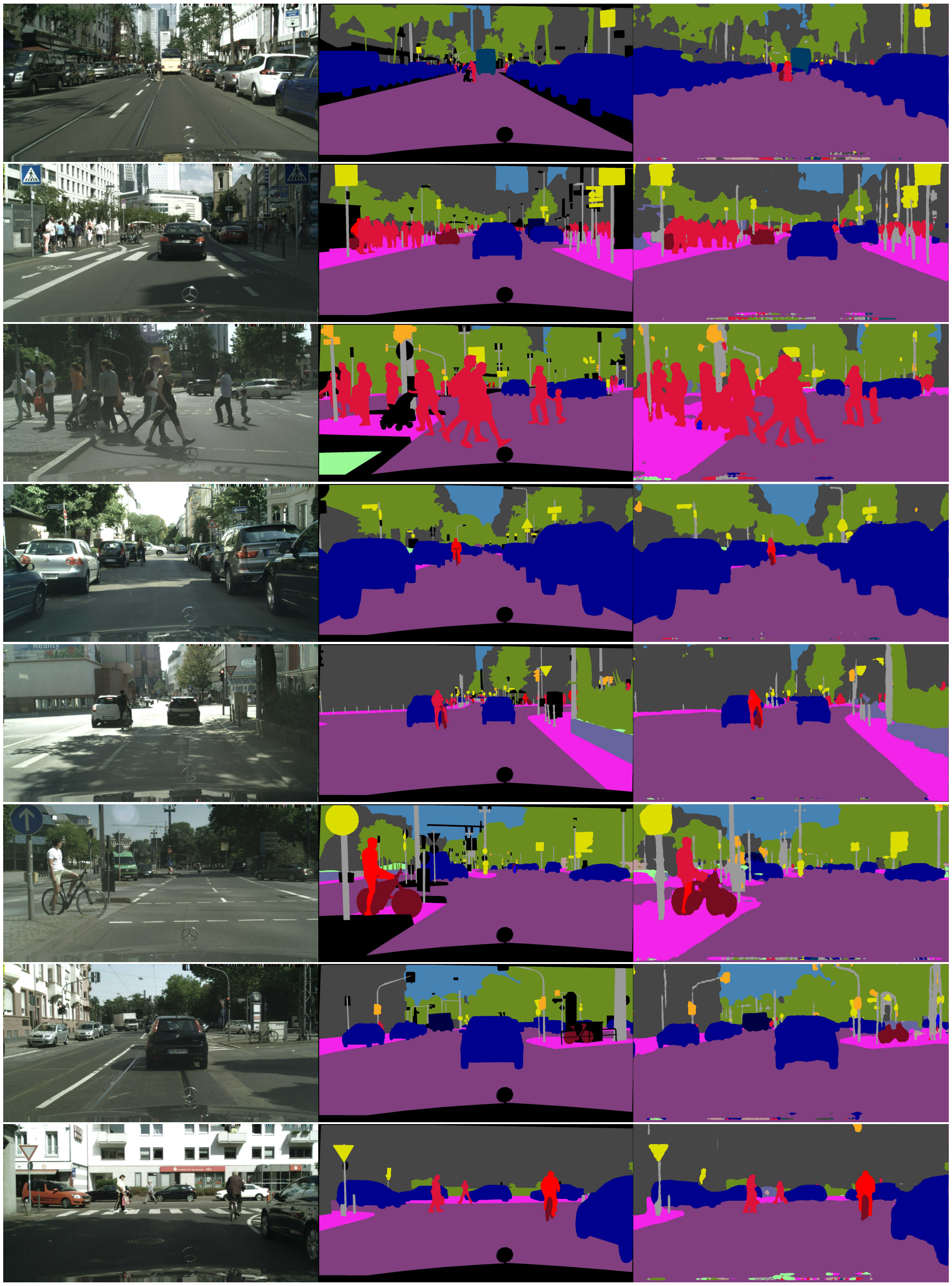
Table 4.5: Ablation study for adding PPM and decoder respectively, on Cityscapes val set.

4.4.5 Visualizations

Fig. 4.4 shows our qualitative segmentation results using ThunderNet. We can see from those visualization results that although ThunderNet is composed of an extremely small backbone and decoder structure, it can still make good predictions for those objects far away. Despite of its lower accuracies on unbalanced classes for instance *wall*, *pole* and *truck*, the network makes accurate pixel-wise predictions for those commonly-seen classes including *road*, *pedestrians* and *vehicles*, which is sufficient for the applications of autonomous driving regarding to the trade-off between speed and accuracy it has already achieved.

4.5 Conclusion

In this chapter, we propose a fast and efficient network for semantic segmentation called ThunderNet–a shorthand for the Turbo Unified Network. ThunderNet builds on an extremely lightweight backbone truncated from the popular ResNet18 model, and unifies both the pyramid pooling module and a customized, two-level



(a) Input image (b) Ground truth (c) ThunderNet

Figure 4.4: Curated examples of the segmentation results output by ThunderNet.

consecutive upsampling decoder structure. Our experimental results on Cityscapes show that ThunderNet has significantly surpassed the currently fastest network in the literature ENet, regarding to both accuracy and inference speed. Even with GPU-powered embedded systems like NVIDIA Jetson TX2, ThunderNet still achieves up to 1.2x speedup. When compared to other methods proposed to achieve the better trade-off between speed and accuracy, ThunderNet still shows its advantages in speed due to its dedicated design of using merely standard convolutions. Without any other non-trivial implementation details and training policies, ThunderNet is easy to train and converges within only a few hours.

CHAPTER 5

3D SENSOR CALIBRATION

This chapter reviews the joint calibration methods for RGB-D sensor, where a quantitative evaluation and comparison is present with our own datasets [147]. This chapter also examines both advantages and disadvantages of current joint calibration methods, from the perspective of practical use.

5.1 Introduction

RGB-Depth (RGB-D) cameras, such as the Microsoft Kinect and Asus Xtion PRO, have become very widely used in perceptual computing applications. Human gesture recognition [148, 149], tracking of facial expressions [150], image segmentation, [151] and 3D reconstruction [152] can be accomplished using the fused color and depth measurements provided by the sensors. In order for this sensor data to be transformed into 3D spatial information, the intrinsic and extrinsic camera parameters must be known or computed using a camera calibration process.

Calibration data is essential for sensing accuracy, and can vary somewhat between devices due to manufacturing variance. Device manufacturers generally provide default calibration values, but these values may not give the best possible results. Additionally, the manufacturer calibration does not take into account depth distortion, referred to as *myopic* property in [153]: if the depth increases, the error increases as well. Depth distortion can be alleviated by using an *undistortion map* that consists of coefficients compensating for distortion on a per-pixel basis. The effectiveness of depth distortion correction has been studied in various work [154, 155, 156].

Most calibration methods proposed for RGB-D devices utilize the color and depth cameras jointly to estimate the relative pose between the two sensors [157, 158, 154, 159, 156]. In this chapter, we present a review and quantitative evaluation of popular joint calibration methods (in particular, [157, 154, 153], where [157, 154] belong to supervised method, and [153] is treated as an unsupervised method) on Kinect V1. In this chapter, we answer some interesting and non-trivial questions regarding the use of RGB-D sensors in practical applications:

- For a specific method, how many images do we need to obtain a stable and satisfactory calibration performance?
- Do more images always result in better calibration?
- Do high-resolution color images help to achieve higher accuracy for a joint calibration method?
- For calibration methods using different sources (IR, disparity, depth), what are the differences and key features of them?

5.2 Related Work

5.2.1 Joint calibration vs non-joint calibration

Calibration for color cameras has been extensively studied in the literature [160]. Most calibration methods proposed for Kinect-like devices utilize the color and depth cameras jointly to estimate the relative pose between the two sensors [157, 158, 154, 159, 156]. Unfortunately, joint calibration requires human interactions to mark plane of the calibration target (checkerboard etc.).

Joint calibration of RGB-D sensors presents challenging problems: 1) it is not easy to obtain correspondences between RGB points and the point cloud captured by the depth sensor. These correspondences must come from structural corners and edges that are difficult to find; 2) the point clouds generated from the depth sensor are quite noisy and incomplete at such corners and edges, due to the technology used (i.e. structured infrared light).

For reference, when using Herrera's method [154], if the input size of images is 30, it took about 3 min to segment images and the calibration took about 2 min. The whole procedure therefore took about 5.5 min [161]. It seems that skipping human interventions will be of great interest and importance in the future development of joint RGB-D calibration methods.

Calibration for color cameras has been extensively studied in literature [160]. As a depth-only calibration method, the work of Jin et al. [155] utilizes cuboids instead of the traditional checkerboard pattern to calibrate Kinect merely in depth domain for the purpose of getting rid of undesired interactions. Their method relies on the precise measurement of angle difference and length, as well as a sophisticated arrangement of cuboids. This adds complexity to the process and decreases ease-of-use.

Calibration methods can be categorized into two classes: 1) supervised calibration and 2) unsupervised calibration, decided upon by whether the calibration target parameters such as shape, size, and color are known in advance.

5.2.2 Supervised calibration

Early work in supervised calibration include Burrus's rgbdemo toolbox [157], Smisek's method [156] and the pioneering study from Herrera et al. [154], with its enhanced model [162], etc. Among these methods, Burrus's and Smisek's work use IR images while Herrera's method use disparity images to jointly calibrate the Kinect.

To minimize human intervention during calibration, some recent works [161, 163] have proposed methods that are able to detect a known pattern automatically.

Specifically, with [161] using a fixed checkerboard and [163] using a moving sphere, both of which rely on ad-hoc parameters: Ilya et al. [161] presented an automatic algorithm which detects the corners of a checkerboard in depth image. The algorithm uses prior knowledge about checkerboard such as side length, diagonal length. Their claimed performance improvement over Herrera's method [154] is trivial, due to the light-weight parameters used in the optimization with respect to the degradation of performance, whereas the number of parameters decreased by $640 \times 480 = 307200$ in per-pixel level. Staranowicz et al. [163] also attempted to find correspondences between the color and depth images. In the RGB image, they detected a sphere using image processing techniques, while in the depth image, RANSAC was used to distinguish between inliers (point cloud of sphere) and outliers (hand, etc.) so that an ellipse of the sphere was fitted. Performance of this method is highly dependent on the accuracy of image processing techniques, as well as the distribution of noise that RANSAC was trying to minimize.

It is worth mentioning that recent work like [154, 164] estimate depth distortion in a per-pixel basis, that is, given a pixel and the corresponding depth/disparity value d, the real depth z_d is estimated as $z_d = f_{(u,v)}(d)$ [165]. Basso et al. [165] proposed a novel supervised method which kept alternating *local* (error related to object shape) and *global* (systematic wrong estimation of the average depth) optimization during calibration process. In their study, depth distortion was imputed to an incorrect parameter set resulting in an absolute error that increases with distances [165], and undistortion map was obtained by applying the local undistortion function to a synthetic point cloud at a defined distance.

5.2.3 Unsupervised calibration

In unsupervised methods, calibration can be performed without a prefabricated rig. Such methods are convenient, but vulnerable to many sources of noise. While the performance of unsupervised methods are generally inferior to supervised methods, their ease-of-use and potential for improvement make them a topic of research interest.

Recently, some unsupervised methods have been proposed [166, 153, 167] which aim to remove prior knowledge of the target. Kummerle et al. [166] utilized simultaneous localization and mapping (SLAM) to perform online calibration on a moving robot. Teichman et al. [153] proposed a generic approach to calibrate Kinect with CLAMS: Calibrating, Localizing, and Mapping Simultaneously. Method of CLAMS firstly reconstructed a scenario by storing trajectories of a moving camera and by building pointcloud from close range data (less than 2m in order to obtain relatively accurate training samples). Then camera parameters were calculated based on maximum likelihood estimation.

5.2.4 Kinect library

As shown in Table. 5.1, three libraries are available for capturing data from Kinect (V1). It should be noted that for use of calibration and other stereo applications, *Libfreenect* is a better choice than others, since Libfreenect captures original disparity data (sometime also called raw depth data), which enables calibration methods to have more accuracy evaluations by calculating reprojection errors of disparities. Furthermore, both libraries, i.e. OpenNI and Kinect for Windows SDK, have automatically transformed disparity to depth for ease of use, however, simply applying inverse transform to obtain disparity by using estimation function like Eq. 5.5 will make calibration results even worse. As we will explain later, Eq. 5.5 is used as an approximation from disparity to depth, which follows distortion correction that plays

Library	OpenNI	Libfreenect	Kinect for Windows SDK
High Resolution Color Image Support	1280 × 1024	1280 × 1024	1280 × 960 [2]
Format of Depth Data	Depth in m	Disparity in kdu	Depth in mm
Operation Range	Default: 0.5m to 4m Near: 0.4m to 3m	0.5m to 4m	Default: 0.8m to 4m Near: 0.5m to 3m
Wrapper Available	OpenCV	Online source [3]	Matlab Image Acquisition Toolbox
Depth Field of View (Horizontal, Vertical, Diagonal)	58°H, 45°V, 70°D [168]		

Table 5.1: Comparison of three different libraries available for Kinect (V1).

an important role in calibration. Therefore only Libfreenect captures raw disparity that can be used in disparity even though other libraries/wrappers available may support generation of disparity that is not original. For instance, OpenCV[1] supports generation of disparity data using OpenNI which essentially performs $d = b * \frac{f}{z_d}$, where b, f indicate horizontal baseline between the cameras (in meters) and focal length (in pixels) respectively, z_d is depth data measured by OpenNI and d is the transformed disparity.

5.3 Calibration Methods

All methods reviewed in this chapter use the pinhole camera model. As in the work of Herrera et al. [154], a 3D point in color camera coordinates $\mathrm{x}_c = [x_c, y_c, z_c]^T$ is projected to the color image plane $\mathrm{p}_c = [u_c, v_c]^T$ as follows:

(1) The point is normalized by the z coordinate: $\mathrm{x}_n = [x_n, y_n]^T = [x_c/z_c, y_c/z_c]^T$.

[1] Relevant API can be found at: `http://docs.opencv.org/doc/user_guide/ug_highgui.html`.

[2] Since SDK v1.0, the high-res RGB color mode of 1280x1024 was replaced by 1280x960 mode, which is the mode supported by the official Kinect for Windows hardware.

[3] For e.g., Matlab wrapper for Libfreenect is available at `http://acberg.com/kinect/`.

(2) Geometric distortion is performed:

$$\mathrm{x}_g = \begin{bmatrix} 2k_3 x_n y_n + k_4(r^2 + 2x_n^2) \\ k_3(r^2 + 2y_n^2) + 2k_4 x_n y_n \end{bmatrix} \tag{5.1}$$

$$x_{ck} = (1 + k_1 r^2 + k_2 r^4 + k_5 r^6)\mathrm{x_n} + x_g \tag{5.2}$$

where, $r^2 = x_n^2 + y_n^2$ and $\mathrm{k}_c = [k_1, ..., k_5]$ is a vector of the distortion coefficients.

(3) The image coordinates p_c are calculated:

$$\mathrm{p}_c = \begin{bmatrix} u_c \\ v_c \end{bmatrix} = \begin{bmatrix} f_{cx} & 0 \\ 0 & f_{cy} \end{bmatrix} \begin{bmatrix} x_{ck} \\ y_{ck} \end{bmatrix} + \begin{bmatrix} u_{0c} \\ v_{0c} \end{bmatrix} \tag{5.3}$$

where $\mathrm{f}_c = [f_{cx}, f_{cy}]$ are focal lengths in x, y axes, respectively, and $\mathrm{p}_{0c} = [u_{0c}, v_{0c}]$ is the principle point.

The raw depth data obtained from the Kinect are 11-bit numbers from 0-2047, called *disparity*, expressed in Kinect disparity units (*kdu*). Conversion from disparity d to depth z_d occurs in two steps:

Distortion correction: The distortion pattern of the depth camera can be corrected using a multiplier image, obtained by measuring reprojection errors of the wall plane at several distances and then dividing all images by the median values across all distances to normalize. Herrera et al. [154] determined that the resulting normalized error medians for each measured disparity fits well to an *exponential* decay. The distortion model can be constructed with per-pixel coefficients that decay exponentially with increasing disparity:

$$d_k = d + D_\delta(u, v) \cdot \exp(\alpha_0 - \alpha_1 d), \tag{5.4}$$

where d denotes distorted disparity returned by Kinect, D_δ denotes spatial distortion pattern, and $\alpha = [\alpha_0, \alpha_1]$ models decay of the distortion effect.

Scaled inverse: There are several equations we can use to estimate depth values [168] from disparity, of which the most commonly used is

$$z_d = \frac{1}{c_1 d_k + c_0}, \tag{5.5}$$

where c_0 and c_1 are depth camera intrinsic parameters to be calibrated and d_k is the corrected disparity.

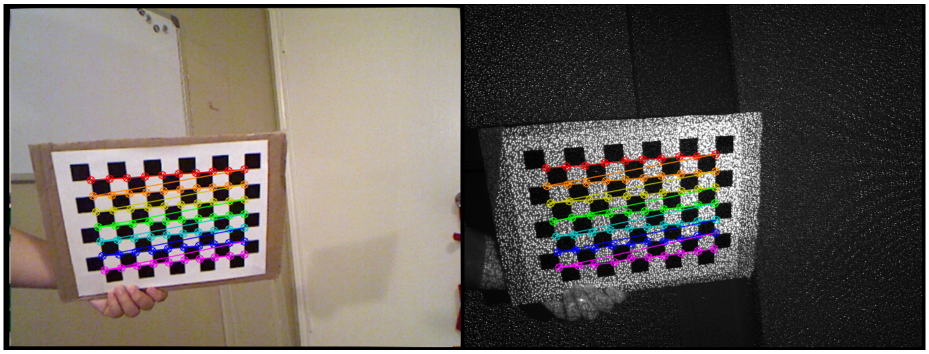
Transformation between depth camera coordinates $\mathrm{x}_d = [x_d, y_d, z_d]^T$ and depth image coordinates $\mathrm{p}_d = [u_d, v_d]^T$ uses a similar model to the color camera:

$$\mathrm{p}_d = \begin{bmatrix} u_d \\ v_d \end{bmatrix} = \begin{bmatrix} f_{dx} & 0 \\ 0 & f_{dy} \end{bmatrix} \begin{bmatrix} x_{dk} \\ y_{dk} \end{bmatrix} + \begin{bmatrix} u_{0d} \\ v_{0d} \end{bmatrix} \tag{5.6}$$

where $[x_{dk}, y_{dk}]^T$ are the coordinates of x_d after normalization of geometric distortion.

As does Herrera et al. [154], we denote the model consisting of Equations (5.1)-(5.3) as $\mathcal{L}_c = \{\mathrm{f}_c, \mathrm{p}_{0c}, \mathrm{k}_c\}$ for the color camera. Similarly, we use $\mathcal{L}_d = \{\mathrm{f}_d, \mathrm{p}_{0d}, \mathrm{k}_d, c_0, c_1, D_\delta, \alpha\}$ to denote the intrinsic parameters of the depth camera.

To perform Kinect calibration, correspondences between color and depth cameras frames are found and used to estimate sensor extrinsic parameters and relative poses. We use three reference frames, $\{D\}$ (depth), $\{C\}$ (color) and $\{V_i\}$ (reference frame of the calibration plane in image i to which the checkerboard pattern is attached). We denote the rigid transformation from one reference frame to another as $\mathcal{T} = \{\mathrm{R}, \mathrm{t}\}$ (as specified by Herrera et al. [154]), where R indicates the rotation matrix and t indicates the translation vector. To transform a point x_w from calibration pattern coordinates $\{W\}$ to color camera coordinates $\{C\}$, we use $\mathrm{x}_c = {}^W\mathrm{R}_C \mathrm{x}_w + {}^W\mathrm{t}_C$, where the rotation and translation from $\{W\}$ to $\{C\}$ are denoted ${}^W\mathrm{R}_C$ and ${}^W\mathrm{t}_C$, respectively. Similarly, the relative pose between depth and color cameras is denoted ${}^D\mathcal{T}_C$. For image i, ${}^{V_i}\mathcal{T}_D$ and ${}^{V_i}\mathcal{T}_C$ denote extrinsics from the reference frame to depth and color frames.



(a) RGB image (b) IR image

Figure 5.1: Burrus method: corner detection in RGB and IR images.

In this chapter, we associate depth camera with IR camera, where disparity/depth/IR data are all calculated based on the same intrinsic depth parameters and thus in the same world coordinates. Furthermore, image coordinates of disparity and depth are treated same whereas IR image coordinates can be calculated by using a simple affine transformation from depth image coordinates (using for e.g. Equation (5.7)).

5.3.1 Burrus's Method

Kinect depth data is calculated by triangulation against known IR projection patterns at a known depth [169]. As we can see in Fig. 5.1, only close-range objects are visible, so the checkerboard must be placed near the camera. Burrus's method [157] uses IR images and is more user-friendly, since it requires no manual labeling. However, it ignores depth distortion and results in relatively low calibration performance.

The IR points are first shifted to disparity image coordinates by applying an affine transformation (see [169] for details):

$$\begin{bmatrix} d_x \\ d_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & -4.8s \\ 0 & 1 & -3.9s \end{bmatrix} \begin{bmatrix} u_{\mathrm{ir}} \\ v_{\mathrm{ir}} \\ 1 \end{bmatrix} \tag{5.7}$$

where, $[u_{\mathrm{ir}}, v_{\mathrm{ir}}]^T$ denotes IR coordinates and $[d_x, d_y]^T$ denotes disparity coordinates. Factor s is set to 2 for high resolution (1280×1024) and 1 for low resolution (640×512) images.

Corners are then extracted from the RGB and IR images. Using a similar notation to Herrera et al., intrinsics $\mathcal{L}_c = \{\mathrm{f}_c, \mathrm{p}_{0c}, \mathrm{k}_c\}$ and $\mathcal{L}_d = \{\mathrm{f}_d, \mathrm{p}_{0d}, \mathrm{k}_d\}$ are estimated for the RGB and IR camera, respectively. After stereo calibration between RGB and IR correspondences is applied using $\mathcal{L}_c$ and $\mathcal{L}_d$, the relative pose between IR camera and RGB camera (${}^D\mathcal{T}_C = \{{}^D\mathrm{R}_\mathrm{C}, {}^D\mathrm{t}_\mathrm{C}\}$) and the fundamental matrix $\mathbf{F}$ are computed. Thus, calibration can be evaluated by computing alternatively the corresponding epilines with $\mathbf{F}$ using RGB and depth points. As Burrus's method ignores depth distortion and uses one-step stereo calibration to coarsely estimate the extrinsics between RGB and IR cameras, the approach leads to lower performance compared to Herrera's method, due to the coarser refinement of both intrinsic and extrinsic parameters.

5.3.2 Smisek's Method

Smisek et al. [156] proposed a method that also calibrates with IR images whereas c_0, c_1 were calibrated with depth images. The procedure of Smisek's and Burrus's method are the same at first (i.e. estimate $\mathcal{L}_c, \mathcal{L}_d$ by showing the same

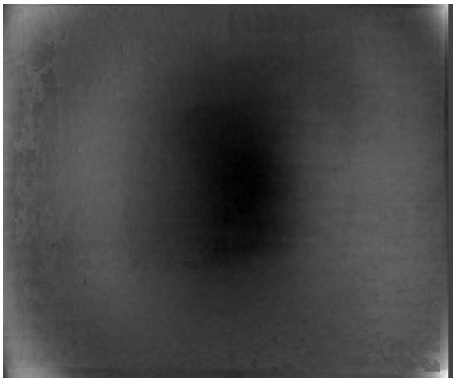


Figure 5.2: Undistortion map visualized by data learned from Herrera's method. Less intense pixels indicate higher values needed to compensate the depth distortion. The horizontal null band of 8 columns on right-most side of image is caused by a 9 x 9 correlation window used by Kinect to compare the local pattern with memorized pattern in order to estimate disparities from IR image [169].

calibration target to both IR and color cameras) except using a slightly different affine transformation:

$$\begin{bmatrix} d_x \\ d_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & -3.0s \\ 0 & 1 & -2.9s \end{bmatrix} \begin{bmatrix} u_{\text{ir}} \\ v_{\text{ir}} \\ 1 \end{bmatrix} \tag{5.8}$$

The shifting offset of $(3.0, 2.9)$ was estimated as the mean value over all four experiments the authors did by calculating misalignment offset in same IR and depth image. After obtaining intrinsics of both cameras, ${}^{D}\mathcal{T}_{C}$ was estimated similarly by a stereo calibration [170]. Instead of fixing $c_0 = 3.3309495161$ and $c_1 = -0.0030711016$ as in Burrus's method, however, Smisek et al. calibrated c_0, c_1 by optimizing them according to

$$\min_{c_0, c_1} \sum_{\text{images}} \sum_{u,v} \left| \hat{d}_k - d_k \right|_2 \tag{5.9}$$

As shown above, d_k denotes measured disparity point within calibration plane, $\hat{d}_k$ denotes disparity data reconstructed by the best plane fit in IR coordinates (see Appendices for details).

Some other contributions should be noted in Smisek's work: first, they found that by blocking the IR projector and illuminating the target with a halogen lamp, a much more clear IR image can be obtained which makes it easier for corner detector to extract corners in IR images. Second, they have observed by fitting a planar target spanning the field of view and by measuring the errors with ground truth distances that small and complex error residuals still exist which varies over distances. As what has been explained early in this chapter, such pattern of residuals is often called depth distortion (see Fig. 5.2). Furthermore, in order to compensate the residual error, they formed a z-correction image (i.e., D_δ) by taking mean of all 18 residual images captured from 0.7 to 1.3 meters. However, the relationship between D_δ and disparity like the one shown in Equation (5.4) was not discovered in Smisek's work.

5.3.3 Herrera's Method

Herrera's method [154] can be divided into two parts: (1) Initialization. (2) Non-linear minimization. The first part involves initializations for three cameras: color camera, depth camera and external camera. Since we are reviewing calibration methods with Kinect only, all content related to external camera in [154] is removed from this chapter. In Herrera's work, Zhang's method [160] is utilized to estimate $\mathcal{L}_c$ and ${}^{W_i}\mathcal{T}_C$ for each image i. The depth camera is then initialized by fixing $\mathrm{f}_d = [590, 590]$, $\mathrm{p}_{0d} = [320, 230]$, $[c_0, c_1] = [3.0938, -0.0028]$ and $\alpha = [1, 1]$, while assigning 0's to f_d and D_δ. As non-linear minimization will be applied later on, the relative pose is also initialized with estimation values: ${}^{D}\mathcal{T}_C = \{{}^{D}\mathrm{R}_C = \mathrm{I}_3, {}^{D}\mathrm{t}_C = [-0.025, 0, 0]^T\}$.

As for the second part, non-linear minimization involves: (1) Sample from disparity images; (2) Refine L_d and ${}^{D}\mathcal{T}_C$; (3) Keep D_δ constant and optimize Equation (5.11); (4) Refine D_δ and k_d independently; (5) Joint minimization by continuing step (3) and (4) iteratively until certain criterion has been met.

The initialization gives a rough estimate of the depth camera parameters and relative pose, whereas the parameters have fairly good initial values for the color camera. Thus, step (2) in non-linear minimization optimizes only L_d and ${}^{D}\mathcal{T}_C$ with all other parameters fixed.

The non-linear minimization relies on using Levenberg-Marquardt algorithm over all related parameters, whereas the cost error is calculated using Euclidean distance. Because the errors have different units, they are weighted by the inverse of the corresponding measurement variance, σ_C^2 and σ_D^2, before calibration:

$$c = \frac{\sum \|\hat{\mathrm{p}}_c - \mathrm{p}_c\|^2}{\sigma_C^2} + \frac{\sum \left(\hat{d} - d\right)^2}{\sigma_D^2}, \tag{5.10}$$

where $\hat{\mathrm{p}}_c$ and $\hat{d}$ indicate the reprojected value for color image point p_c and measured disparity d, respectively. Since the above function has high nonlinearity and depends on many parameters (for instance, computing $\hat{d}_k - d_k$ requires D_δ which contains $640 \times 480 = 307,200$ entries), the authors simplify the cost function by separating the optimization of disparity distortion parameters, i.e. by calculating the residuals in undistorted disparity space instead of in measured disparity space:

$$c = \frac{\sum \|\hat{\mathrm{p}}_c - \mathrm{p}_c\|^2}{\sigma_C^2} + \frac{\sum \left(\hat{d}_k - d_k\right)^2}{\sigma_D^2}. \tag{5.11}$$

Using equation (5.4), residuals in undistorted disparity space can be rewritten as

$$c_d = \sum_{\text{images}} ss\sum_{u,v} \left(d + D_\delta(u,v) \cdot \exp(\alpha_0 - \alpha_1 d) - \hat{d}_k\right)^2. \tag{5.12}$$

It should be noted that Herrera's method requires users to manually select corner points of the calibration plane in disparity images in order to estimate the plane equation and predict disparities. Though the procedure looks tedious , it allows for non-minimization in disparity space where raw data were captured, therefore leading to higher accuracy compared to Burrus's method. Burrus's calibration uses near-range IR data and ignores depth points that can be gathered at various distances, thus missing information on memorized patterns that have been embedded in Kinect. Since Kinect is a closed system, it is just a deduction from the postulates.

5.3.4 Teichman's Method

The unsupervised calibration method of Teichman et al. [153] employs SLAM to capture the camera trajectory and build a map of the environment. After the point cloud map is created, the depth points are then undistorted by adding a Gaussian noise:

$$\mathbb{P}\left(z|z_d, w\right) = \eta \exp\left(\frac{-\left(z - wz_d\right)}{2\delta}\right) \tag{5.13}$$

where w denotes per-pixel multiplier in undistortion map, z_d and z denote measured depth and ground truth depth, respectively. Therefore, in sense of maximum likelihood estimation, Teichman's method optimized D_δ using

$$\max_{w} \prod_{z_d \in \text{map}} \mathbb{P}(z|z_d, w) \tag{5.14}$$

which can be reduced to

$$\min_{w} \prod_{z_d \in \text{map}} \left(z - wz_d\right)^2 \tag{5.15}$$

So far, Teichman's method calibrates only the distortion pattern without considering relative pose between color camera and depth camera, as well as the depth geometric distortion. As shown in Fig. 5.4, undistortion map obtained using Techiman's method exhibits similar pattern with Fig. 5.2.

Though proposed as a pioneering work, we have discovered that when using Teichman's method it is difficult to capture enough samples at all parts of the view frustum, because a sufficient number of training examples at far distances (6m to 10m) is guaranteed only if the scene consists of a planar surface that is long and wide enough.

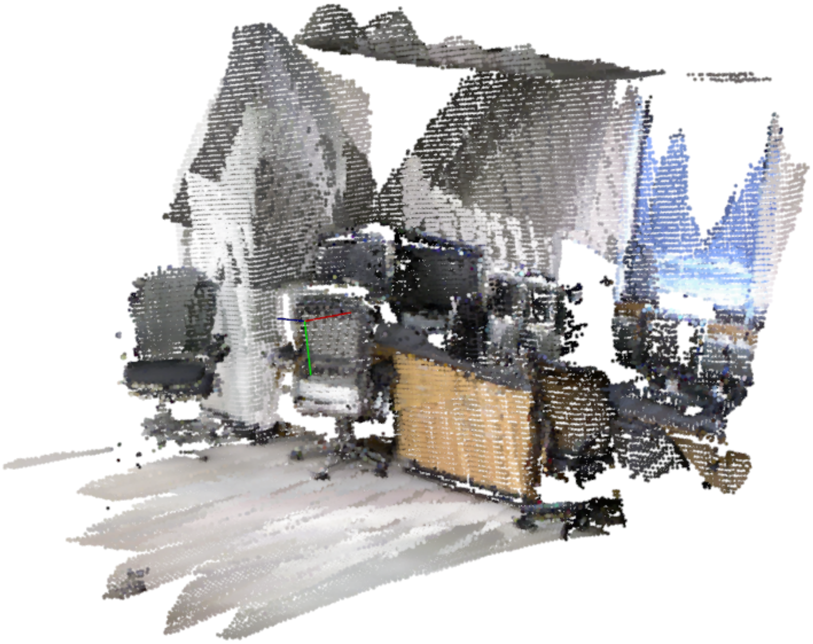


Figure 5.3: Example pointcloud of environment recorded using Techiman's method.

We have discovered that it is difficult to capture enough training data at all parts of the view frustum, because a sufficient number of samples on corners and edges at far distances (6m to 10m) are guaranteed only if the scene consists of a planar surface that is long and wide enough. In addition, it is unavoidable to repeat the procedure of running SLAM and calibration for many times, in order to obtain point clouds with adequate coverage of environment.

5.4 Evaluation

To examine Herrera's method using disparity images, four datasets were captured: two for calibration (*A1*, *A2*) and two for validation (*B1*, *B2*). A1 and B1 consist of 51 and 61 images respectively and both of B1, B2 consist of 15 images. In order to capture the original disparity data, we used the *Libfreenect*[168] driver. The

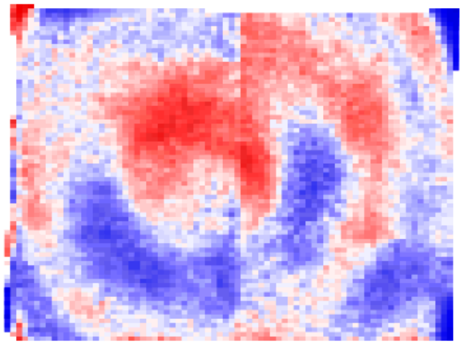
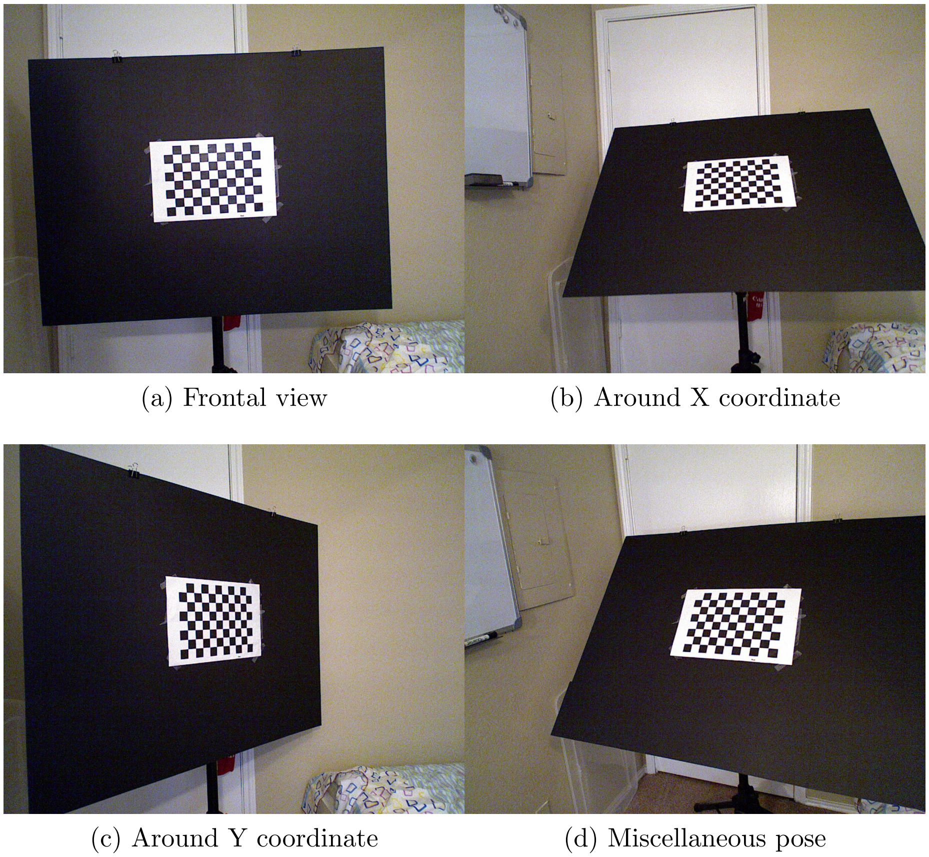


Figure 5.4: Undistortion map visualized by data learned from Teichman's method. Full red indicates multiplier of 1.1, full blue of 0.9 and full white of 1.0. Similar to Fig. 5.2, lower values indicate the data need to be pulled closer to the sensor and thus considered more unstable according to myopic property. Null band is removed intentionally from this image.



(a) Frontal view

(b) Around X coordinate

(c) Around Y coordinate

(d) Miscellaneous pose

Figure 5.5: Examples of dataset used in our evaluation.

RGB data measured in our experiments are in medium resolution (640×480) as well as high resolution (1280×1024), both were paired with 640×480 disparity images. All four datasets were captured with the same Kinect, among which A1 and B1 are in medium resolution, A2 and B2 are in high resolution. Details are given in Table 5.2.

Dataset / Description	A1	A2	B1	B2
color resolution	medium	high	medium	high
# total poses	51	60	15	15
# frontal view	9	10	4	3
# around X coordinate	12	10	3	4
# around Y coordinate	9	10	3	4
# misc. poses	6	15	5	4
# wall images	15	15	0	0

Table 5.2: Description of all four datasets A1, A2, B1 and B2, where A1 and A2 are used for calibration methods using disparity images.

For Burrus's method, we captured an IR image dataset named *A3* that consists of 56 images for evaluation. Since the visible range for capturing IR images is limited, the dataset was created by changing the distances slightly given multiple poses of the calibration target.

For Teichman's method, taking sufficient training examples at all depths (from 0m to 10m) over all areas of the image frame is difficult, and we have only been able to capture a sufficient number of training examples from 0m to 4m. Since we evaluate Teichman's work with respect to depth uncertainty in both A1 and A2, where maximum depths are less than 2.5m and 3m respectively, the performance is therefore guaranteed with full capacity when evaluated in A1 and A2.

In this chapter, validation of Herrera's method follows its original work[154]: After estimating all calibration parameters, validation is performed by fixing intrinsic parameters and estimating the poses of the checkerboard plane (${}^{W}\mathcal{T}_{C}$). The variances

$\mathcal{L}_d$ learned	A1	error↓	A2	error↓
$\mathrm{f}_d, \mathrm{p}_{0d}$	1.75	—	2.14	—
$\mathrm{f}_d, \mathrm{p}_{0d}, \alpha$	1.75	0.00	2.14	0.00
$\mathrm{f}_d, \mathrm{p}_{0d}, \alpha, c_0, c_1$	1.73	0.02	2.13	0.01
$\mathrm{f}_d, \mathrm{p}_{0d}, \alpha, c_0, c_1, \mathrm{k}_d$	1.69	0.04	1.98	0.15
$\mathrm{f}_d, \mathrm{p}_{0d}, \alpha, c_0, c_1, \mathrm{k}_d, D_\delta$	1.08	0.61	1.20	0.78

Table 5.3: Different parameters learned from Herrera's method with corresponding calibration error in A1 and A2. Errors are measured by std. dev. of residuals for reprojected disparities with a 99% confidence interval.

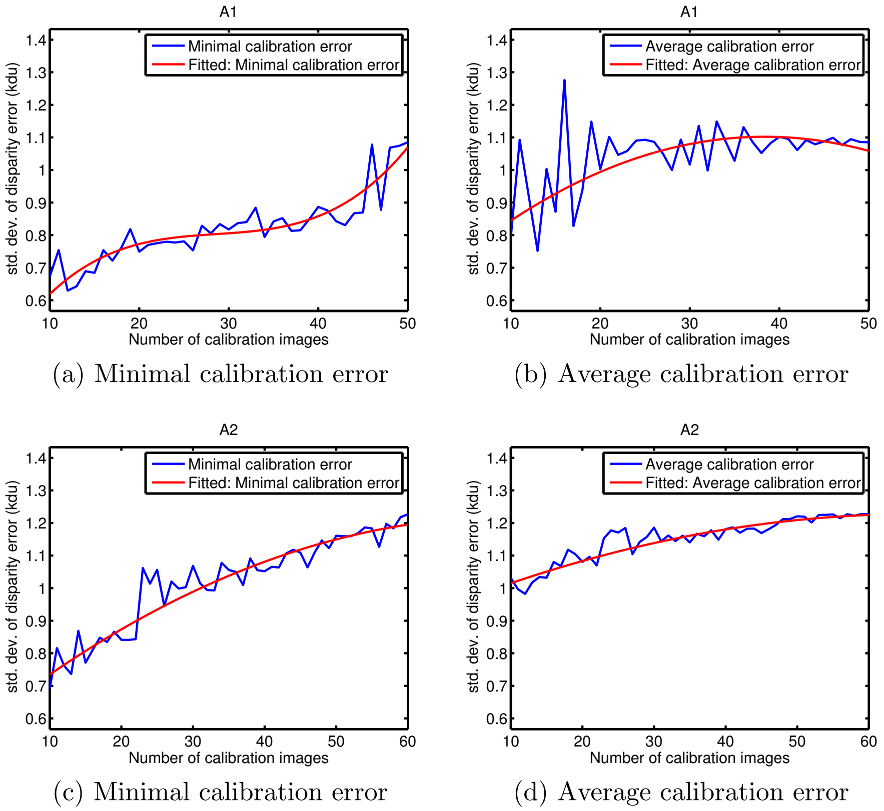
of color and disparity error were kept constant ($\sigma_c = 0.18$px, $\sigma_d = 0.9$kdu) so that we can compare different calibrations in an unbiased manner (see Equation (5.11)).

5.4.1 Calibration error vs. parameters learned

Table 5.3 presents how calibration error decreases as more parameters are learned in Herrera's calibration method. We have found the largest amount of decrease of calibration error after considering depth distortion D_δ, and the second largest one resulted from adding depth geometric distortion k_d, meaning that both parameters, especially D_δ, are deterministic in fitting the model to training examples. For those parameters not learned during calibration, they were initialized as follows:

$$\begin{aligned}
\alpha &= \begin{bmatrix} 1 & 1 \end{bmatrix} \\
(c_0, c_1) &= \begin{bmatrix} 3.3309495161 & -0.0030711016 \end{bmatrix} \\
\mathrm{k}_d &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
D_\delta &= \mathbf{0}_{480\times640}
\end{aligned}$$

Note that as suggested by [171], we will calibrate $\mathcal{L}_d = \{\mathrm{f}_d, \mathrm{p}_{0d}, \alpha, \mathrm{k}_d, D_\delta\}$ in the next two experiments meaning that (c_0, c_1) will be fixed at $c_0 = 3.3309495161$ and $c_1 = -0.0030711016$ respectively.



(a) Minimal calibration error

(b) Average calibration error

(c) Minimal calibration error

(d) Average calibration error

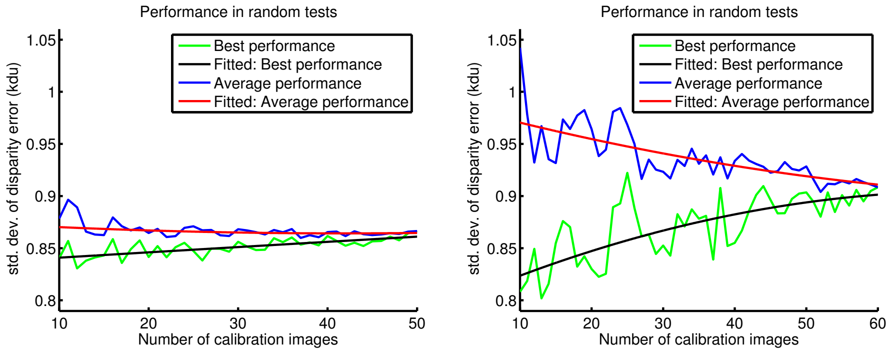
Figure 5.6: Calibration error of Herrera's method on A1 and A2 in random tests: with corrections of both geometric distortion and depth distortion.

5.4.2 Calibration performance vs. number of images

In this experiment, we randomly selected n images 10 times from both A1 and A2 (where $10 \leqslant n \leqslant 51$ in A1 and $10 \leqslant n \leqslant 60$ in A2), and then calculated the calibration error based on disparity reprojection error by std. deviations of residuals with a 99% confidence interval [154]. Validations were then performed in the same manner using the corresponding validation dataset (i.e. from A1 to B1, from A2 to B2). For reference, Fig. 5.6 shows how well Herrera's method fit the calibration datasets with random tests. As we can see from Fig. 5.6, the calibration error increases with the increasing number of images, due to the additional uncertain depth points that are added to the calibration set.

Fig. 5.7 shows the calibration performance of Herrera's method on B1 and B2. The performance on datasets of medium resolution color images (A1 & B1) proved to be better than on those of high resolution color images (A2 & B2). The performance becomes stable in both B1 and B2 as the number of images increases. For both validations, the best performance can be obtained with extensive experiments, which in our case n is around 12. Nevertheless, in order to achieve a stable and satisfactory performance for one-time calibration, users of Herrera's method are suggested to calibrate with 50 images in standard resolution while with 60 images in high resolution. Besides, as the distributions of poses in training sets are almost identical, however, in most of tests the average calibration performance of A1 are better than A2 showing that the higher resolution color images used in Herrera's method do not necessarily improve calibration performance. The results of one-way An Analysis of Variance (ANOVA) test ($p = 0.1297$ for B1 and $p < 0.0001$ for B2) indicate that when using medium-resolution dataset, the random tests suggest a common mean with increasing number of images, while using high-resolution dataset, the mean of random test results differ given different number of images. This is because more details were captured in high-resolution dataset and thus calibration procedure is more vulnerable to different sources of noise.

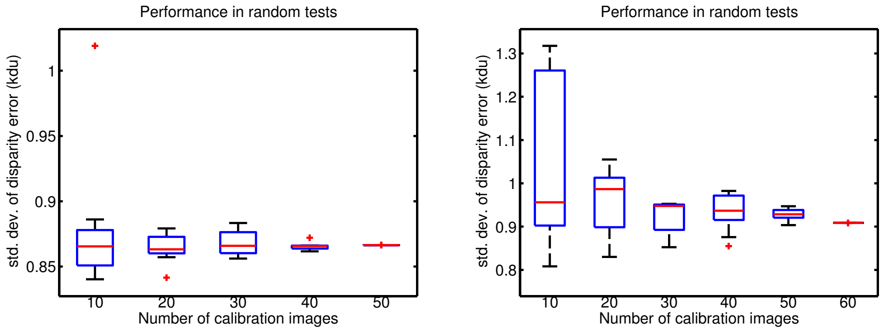
Also, as shown in Fig. 5.6, low calibration errors do not necessarily result in higher calibration performance on validation datasets, especially after calibrating with those images consisting of data of high variance. Most random tests in A1 have average calibration error std. dev. ≥ 1, however, all of them have test results where std. dev. ≤ 0.9. Conclusively, it is not fair to predict the calibration performance merely based on calibration error. When using Herrera's methods, it should be noted that some *restrictions* apply during calibration:



(a) Validation of A1 in B1

(b) Validation of A2 in B2

Figure 5.7: Performance of Herrera's method on B1 and B2 after calibration with A1 and A2 in random tests: with corrections of both geometric distortion and depth distortion.



(a) Validation of A1 in B1

(b) Validation of A2 in B2

Figure 5.8: Boxplot of performance of Herrera's method based on Fig. 5.7.

(1) Image of checkerboard should be attached to a plane big enough. Though you don't have to find a calibration plane which fills the whole space, it is suggested to use a plane four times larger than the checkerboard image. Small calibration plane will lead to unacceptable calibration result due to insufficient depth points sampled and higher chances of sampling more noise.

(2) Users are required to follow Table. 5.2 to take images with various poses as it will aid to get better calibration performance.

5.4.3 Depth uncertainty

Since stereo devices like Kinect are exposed to the problem of myopia as aforementioned, we can compare the depth uncertainty of the three methods with the manufacturer calibration. In Herrera's method, depth uncertainty was measured in A1 and A2 with calibration results from the two datasets respectively. As shown in Fig. 5.9, depth points of A1 are more stable and accurate than those of A2. Herrera's method is clearly better than manufacturer calibration, especially at the camera's near range (up to 1.5m), in accordance with [154].

Depth uncertainty for Burrus's method in Fig. 5.9 is presented as a curve that fits closely to one of the manufacturer calibration. This method shows no improvement over the manufacturer calibration. That being said, depth distortion was not examined in manufacturer calibration, as it is also ignored in Burrus's method.

Calibration performance of Teichman's method is not satisfactory with respect to depth uncertainty because they merely calibrated D_δ and ignored other important parameters in $\mathcal{L}_d = \{\mathrm{f}_d, \mathrm{p}_{0d}, \mathrm{k}_d, c_0, c_1, D_\delta, \alpha\}$. Note that in this experiment, f_d, p_{0d} are fixed at $[590, 590]$ and $[320, 230]$ respectively, in order to correctly compute the depth uncertainty.

Table 5.4 shows a quantitative comparison study for all three methods evaluated in depth uncertainty. Herrera's method clearly outperforms the other two methods and the manufacturer calibration though for depths between 1.5m and 3.0m, the manufacturer calibration can also achieve a satisfactory performance. In addition, due to lack of parameters considered in the calibration procedure, Teichman's method performed worse than other approaches including manufacturer calibration, especially at depth beyond 1m.

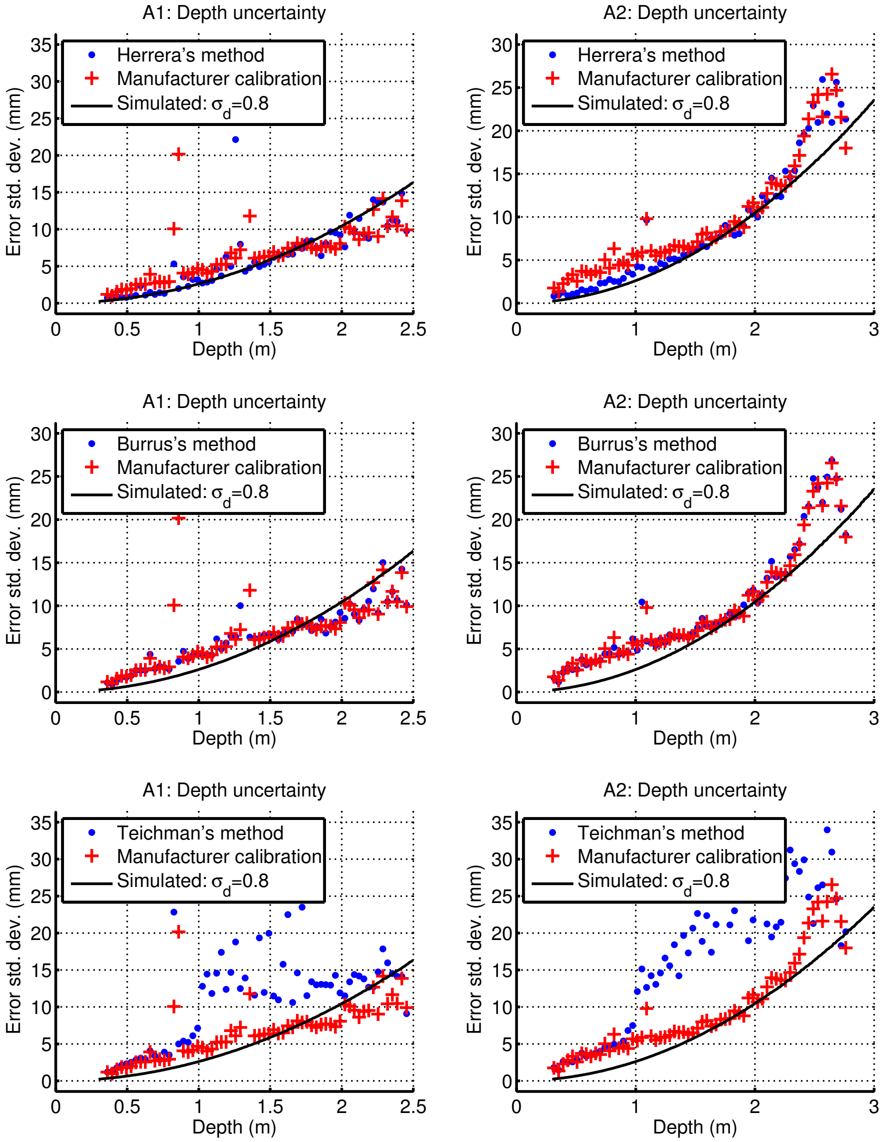


Figure 5.9: All methods: depth uncertainty measured from A1 and A2.

Depth z (m)	$0 \leq z < 0.5$		$0.5 \leq z < 1$		$1 \leq z < 1.5$		$1.5 \leq z < 2$		$2 \leq z < 2.5$		$2.5 \leq z < 3$	
	A1	A2	A1	A2	A1	A2	A1	A2	A1	A2	A1	A2
Manufacturer	1.47	2.30	4.80	4.24	7.60	6.40	**7.44**	8.75	10.70	**14.48**	—	22.98
Burrus's	1.36	2.10	4.74	4.15	7.31	6.40	7.50	8.90	**10.68**	15.92	—	23.10
Herrera's	**0.81**	**0.99**	**4.24**	**2.28**	**5.91**	**5.20**	7.65	**8.14**	11.24	15.40	—	**22.84**
Teichman's	1.73	2.21	5.38	4.60	15.80	15.71	14.17	22.68	13.79	25.73	—	25.81

Table 5.4: Comparison of three different methods evaluated with respect to depth uncertainty. Depths are divided into bins of size 0.5m. For A1, the maximum depth is 2.5m while 3.0m for A2. Depth uncertainty are measured in mm and best results are highlighted in the table.

5.5 Conclusion

In this chapter, we have reviewed and presented a quantitative comparative study of three calibration methods categorized as pioneering work, using different sources: IR, disparity map, and depth image. We have shown that Herrera's calibration method using disparity image outperforms other methods due to more intrinsic depth parameters exploited. We also present an empirical sample complexity analysis for Herrera's method. The unsupervised approach proposed by Teichman et al. is of great potential since it requires no assumption on calibration target or user interactions. However, the performance of such methods is not as good as supervised ones since it learns fewer parameters and is more likely to be implicitly constrained in indoor environment. Future work can combine supervised and unsupervised approach by adding more constraints in unsupervised fashion.

CHAPTER 6

CONCLUSIONS

This thesis aims to develop effective and efficient deep learning based models, for both object detection and semantic segmentation. We investigate the state-of-the-art methods in these two fields by testing on large-scale datasets including ImageNet, MS-COCO, VOC, DETRAC, Cityscapes, etc. We have present, both theoretically and empirically, different solutions for object detection and semantic segmentation in order to develop effective and efficient models.

In object detection, we firstly study the empirical receptive fields and provide the corresponding statistical analyses for a VGG16 network. To apply our findings of ERFs onto SSD, we further propose a novel network architecture called context-aware SSD, by integrating contextual information of different levels into the output of prediction layers of SSD. Our experimental results demonstrated that CSSD not only outperforms SSD with general large-scale object detection datasets like VOC and MS-COCO, but also surpassed SSD in task-specific dataset like DETRAC for surveillance scenario.

In semantic segmentation, we fill the gap in the literature between high-quality prediction and real-time performance, by proposing ThunderNet–an effective and efficient network that consists of a minimum backbone, a pyramid pooling module and a two-level customized decoder. We test ThunderNet on Cityscapes with both desktop-level machine and GPU-powered embedded system, and demonstrate that ThunderNet is able to achieve comparable accuracy to a majority of methods currently available and is extremely fast regarding to the inference speed. Besides, our results on embedded systems indicate that ThunderNet requires much fewer computational resources, and therefore meeting the industrial demands from the perspective of the tradeoff between accuracy and runtime speed.

Finally, this thesis also provides a quantitative evaluation of joint calibration methods for RGB-D sensor. We discuss the practical issues of current methods and compare, both quantitatively and qualitatively, the benchmarked methods with our own datasets.

REFERENCES

[1] 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.

[2] 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¡ 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.

[3] W. Xiang, D.-Q. Zhang, H. Yu, and V. Athitsos, “Context-aware single-shot detector,” *arXiv preprint arXiv:1707.08682*, 2017.

[4] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” *arXiv preprint arXiv:1612.03144*, 2016.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” *arXiv preprint arXiv:1703.06870*, 2017.

[6] J. Ren, X. Chen, J. Liu, W. Sun, J. Pang, Q. Yan, Y.-W. Tai, and L. Xu, “Accurate single stage detector using recurrent rolling convolution,” *arXiv preprint arXiv:1704.05776*, 2017.

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

[8] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, “Enet: A deep neural network architecture for real-time semantic segmentation,” *arXiv preprint arXiv:1606.02147*, 2016.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[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] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[13] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, no. 9, pp. 1904–1916, 2015.

[15] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[16] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv preprint arXiv:1606.00915*, 2016.

[18] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1653–1660.

[19] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun, "Light-head r-cnn: In defense of two-stage object detector," *arXiv preprint arXiv:1711.07264*, 2017.

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

[21] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[22] A. Coates, H. Lee, and A. Y. Ng, "An analysis of single-layer networks in unsupervised feature learning," *Ann Arbor*, vol. 1001, no. 48109, p. 2, 2010.

[23] R. Benenson, "What is the class of this image? discover the current state of the art in objects classification," 2016, [accessed 22-Feb-2016]. [Online]. Available: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[24] B. Graham, "Fractional max-pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[25] A. Karpathy, "Lessons learned from manually classifying cifar-10," 2011, [accessed Apr-2011]. [Online]. Available: http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/

[26] ——, "What i learned from competing against a convnet on imagenet," 2014, [accessed Feb-2015]. [Online]. Available: http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

[27] A. Ray, "What is the state-of-the-art today on imagenet classification?" 2016, [accessed Jul-2016]. [Online]. Available:

https://www.quora.com/What-is-the-state-of-the-art-today-on-ImageNet-classification-In-other-words-has-anybody-beaten-Deep-Residual-Learning

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[29] 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.

[30] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision*. Springer, 2016, pp. 21–37.

[31] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2881–2890.

[32] C. Gulcehre, "Deep learning — software links," 2016, [accessed 23-Aug-2017]. [Online]. Available: http://deeplearning.net/software_links/

[33] 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 preprint arXiv:1408.5093*, 2014.

[34] D. Yu, A. Eversole, M. Seltzer, K. Yao, Z. Huang, B. Guenter, O. Kuchaiev, Y. Zhang, F. Seide, H. Wang, *et al.*, "An introduction to computational networks and the computational network toolkit," Technical report, Tech. Rep. MSR, Microsoft Research, 2014, 2014. research. microsoft. com/apps/pubs, Tech. Rep., 2014.

[35] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[36] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[38] B. Vision and L. Center, "platoon," 2017, [accessed Oct-2017]. [Online]. Available: https://github.com/BVLC/caffe/wiki/Model-Zoo

[39] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[40] M. I. for Learning Algorithms, "platoon," 2015, [accessed Sep-2017]. [Online]. Available: https://github.com/mila-udem/platoon

[41] E. Research, "Recurrent neural network library for torch7's nn," 2016, [accessed Nov-2016]. [Online]. Available: https://github.com/Element-Research/rnn

[42] P. Bhatia, "Neural conversation models," 2016, [accessed Mar-2017]. [Online]. Available: https://github.com/pbhatia243/Neural_Conversation_Models

[43] T. official tutorial, “Loop,” 2017, [accessed Nov-2017]. [Online]. Available: http://deeplearning.net/software/theano/tutorial/loop.html

[44] Wikipedia, “Comparison of deep learning software,” 2016, [accessed 27-Jul-2016]. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

[45] K. Tran, “Evaluation of deep learning toolkits,” 2016, [accessed May-2016]. [Online]. Available: https://github.com/zer0n/deepframeworks

[46] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems,” *arXiv preprint arXiv:1512.01274*, 2015.

[47] Baidu, “Paddlepaddle,” 2017, [accessed Mar-2018]. [Online]. Available: https://github.com/PaddlePaddle/Paddle

[48] Tecent, “ncnn,” 2017, [accessed Mar-2018]. [Online]. Available: https://github.com/Tencent/ncnn

[49] S. Mohammed, “Explaining fp64 performance on gpus,” 2015, [accessed 22-Jun-2015]. [Online]. Available: http://arrayfire.com/explaining-fp64-performance-on-gpus/

[50] T. Dettmers, “How to parallelize deep learning on gpus part 1/2: Data parallelism,” 2014, [accessed 09-Oct-2014]. [Online]. Available: http://timdettmers.com/2014/10/09/deep-learning-data-parallelism/

[51] T. O. Tutorials, “Convolutional neural networks,” 2018, [accessed Jan-2018]. [Online]. Available: https://www.tensorflow.org/tutorials/deep_cnn

[52] T. Dettmers, “How to parallelize deep learning on gpus part 2/2: Model parallelism,” 2014, [accessed 09-Nov-2014]. [Online]. Available: http://timdettmers.com/2014/11/09/model-parallelism-deep-learning/

[53] 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*, 2015, pp. 1–9.

[54] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[55] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

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

[57] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[58] K. Sohn and H. Lee, "Learning invariant representations with local transformations," *arXiv preprint arXiv:1206.6418*, 2012.

[59] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 8, pp. 1872–1886, 2013.

[60] A. Kanazawa, A. Sharma, and D. Jacobs, "Locally scale-invariant convolutional neural networks," *arXiv preprint arXiv:1412.5104*, 2014.

[61] M. F. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber, "Deep networks with internal selective attention through feedback connections," in *Advances in Neural Information Processing Systems*, 2014, pp. 3545–3553.

[62] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2008–2016.

[63] K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” *arXiv preprint arXiv:1502.03044*, 2015.

[64] S. Dieleman, J. De Fauw, and K. Kavukcuoglu, “Exploiting cyclic symmetry in convolutional neural networks,” *arXiv preprint arXiv:1602.02660*, 2016.

[65] 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*, 2015, pp. 1135–1143.

[66] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, “Eie: efficient inference engine on compressed deep neural network,” *arXiv preprint arXiv:1602.01528*, 2016.

[67] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” *arXiv preprint arXiv:1603.05279*, 2016.

[68] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” *arXiv preprint arXiv:1606.06160*, 2016.

[69] AI2, “Xnor-net,” 2016, [accessed Dec-2017]. [Online]. Available: https://github.com/mrastegari/XNOR-Net

[70] Y. Wu, “Dorefa-net,” 2016, [accessed Mar-2018]. [Online]. Available: https://github.com/ppwwyyxx/tensorpack/tree/master/examples/DoReFa-Net

[71] A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves, “Memory-efficient backpropagation through time,” *arXiv preprint arXiv:1606.03401*, 2016.

[72] Y. N. Dauphin and Y. Bengio, “Big neural networks waste capacity,” *arXiv preprint arXiv:1301.3583*, 2013.

[73] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[74] L. Y. Pratt, *Comparing biases for minimal network construction with back-propagation.* Morgan Kaufmann Pub, 1989, vol. 1.

[75] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage." in *NIPs*, vol. 2, 1989, pp. 598–605.

[76] B. Hassibi and D. G. Stork, *Second order derivatives for network pruning: Optimal brain surgeon.* Morgan Kaufmann, 1993.

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

[78] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[79] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *CoRR, abs/1504.04788*, 2015.

[80] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[81] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[82] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5353–5360.

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

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

[85] 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 preprint arXiv:1704.04861*, 2017.

[86] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *arXiv preprint arXiv:1801.04381*, 2018.

[87] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[88] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1131–1135.

[89] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *arXiv preprint arXiv:1510.03009*, 2015.

[90] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, vol. 1. Citeseer, 2011, p. 4.

[91] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2014, pp. 1–6.

[92] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[93] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[94] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Advances in Neural Information Processing Systems*, 2014, pp. 963–971.

[95] J. M. Hernández-Lobato and R. P. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," *arXiv preprint arXiv:1502.05336*, 2015.

[96] Stanford, "Cs231n course materials," 2016, [accessed Nov-2017]. [Online]. Available: http://cs231n.github.io/convolutional-networks

[97] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[98] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

[99] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[100] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[101] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene cnns," *International Conference on Learning Representations*, 2015.

[102] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 4898–4906.

[103] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[104] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision*. Springer, 2014, pp. 740–755.

[105] L. Wen, D. Du, Z. Cai, Z. Lei, M. Chang, H. Qi, J. Lim, M. Yang, and S. Lyu, "DETRAC: A new benchmark and protocol for multi-object detection and tracking," *arXiv CoRR*, vol. abs/1511.04136, 2015.

[106] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1532–1545, 2014.

[107] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[108] S. Zagoruyko, A. Lerer, T.-Y. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár, "A multipath network for object detection," *arXiv preprint arXiv:1604.02135*, 2016.

[109] Y. Li, K. He, J. Sun, *et al.*, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 379–387.

[110] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2012, pp. 3354–3361.

[111] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[112] P. Hu and D. Ramanan, "Finding tiny faces," *arXiv preprint arXiv:1612.04402*, 2016.

[113] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[114] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "Dssd: Deconvolutional single shot detector," *arXiv preprint arXiv:1701.06659*, 2017.

[115] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient deep learning for stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5695–5703.

[116] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[117] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1520–1528.

[118] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[119] D. Hoiem, Y. Chodpathumwan, and Q. Dai, "Diagnosing error in object detectors," in *European conference on computer vision.* Springer, 2012, pp. 340–353.

[120] S. Bell, C. Lawrence Zitnick, K. Bala, and R. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2874–2883.

[121] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," in *Wavelets*. Springer, 1990, pp. 286–297.

[122] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[123] L.-J. Li, R. Socher, and L. Fei-Fei, "Towards total scene understanding: Classification, annotation and segmentation in an automatic framework," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 2036–2043.

[124] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 129–136.

[125] A. Mallya and S. Lazebnik, "Learning models for actions and person-object interactions with transfer to question answering," in *European Conference on Computer Vision*. Springer, 2016, pp. 414–428.

[126] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1520–1528.

[127] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[128] G. Lin, C. Shen, A. Van Den Hengel, and I. Reid, “Efficient piecewise training of deep structured models for semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3194–3203.

[129] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The cityscapes dataset for semantic urban scene understanding,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[130] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, “Rethinking atrous convolution for semantic image segmentation,” *arXiv preprint arXiv:1706.05587*, 2017.

[131] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, *et al.*, “Speeding up semantic segmentation for autonomous driving,” in *MLITS, NIPS Workshop*, 2016.

[132] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, “Icnet for real-time semantic segmentation on high-resolution images,” *arXiv preprint arXiv:1704.08545*, 2017.

[133] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, “Erfnet: Efficient residual factorized convnet for real-time semantic segmentation,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018.

[134] 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.

[135] 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,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[136] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention.* Springer, 2015, pp. 234–241.

[137] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun, “Large kernel matters–improve semantic segmentation by global convolutional network,” *arXiv preprint arXiv:1703.02719*, 2017.

[138] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, “Understanding convolution for semantic segmentation,” *arXiv preprint arXiv:1702.08502*, 2017.

[139] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” *arXiv preprint arXiv:1802.02611*, 2018.

[140] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *CVPR*, vol. 1, no. 2, 2017, p. 4.

[141] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Semantic image segmentation with deep convolutional nets and fully connected crfs,” in *ICLR*, 2015. [Online]. Available: http://arxiv.org/abs/1412.7062

[142] G. Lin, A. Milan, C. Shen, and I. Reid, “Refinenet: Multi-path refinement networks for high-resolution semantic segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[143] K.-H. Kim, S. Hong, B. Roh, Y. Cheon, and M. Park, “Pvanet: deep but lightweight neural networks for real-time object detection,” *arXiv preprint arXiv:1608.08021*, 2016.

[144] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[145] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr, "Conditional random fields as recurrent neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1529–1537.

[146] T. Saemann, "Enet," 2017, [accessed Jul-2017]. [Online]. Available: https://github.com/TimoSaemann/ENet

[147] W. Xiang, C. Conly, C. D. McMurrough, and V. Athitsos, "A review and quantitative comparison of methods for kinect calibration," in *Proceedings of the 2nd international Workshop on Sensor-based Activity Recognition and Interaction.* ACM, 2015, p. 3.

[148] O. Patsadu, C. Nukoolkit, and B. Watanapa, "Human gesture recognition using kinect camera," in *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on.* IEEE, 2012, pp. 28–32.

[149] Z. Ren, J. Yuan, J. Meng, and Z. Zhang, "Robust part-based hand gesture recognition using kinect sensor," *Multimedia, IEEE Transactions on*, vol. 15, no. 5, pp. 1110–1120, 2013.

[150] Q. Cai, D. Gallup, C. Zhang, and Z. Zhang, "3d deformable face tracking with a commodity depth camera," in *Computer Vision–ECCV 2010.* Springer, 2010, pp. 229–242.

[151] S. Nathan, H. Derek, K. Pushmeet, and F. Rob, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[152] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[153] A. Teichman, S. Miller, and S. Thrun, "Unsupervised intrinsic calibration of depth sensors via slam." in *Robotics: Science and Systems*, 2013.

[154] C. Herrera, J. Kannala, J. Heikkilä, *et al.*, "Joint depth and color camera calibration with distortion correction," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 10, pp. 2058–2064, 2012.

[155] B. Jin, H. Lei, and W. Geng, "Accurate intrinsic calibration of depth camera with cuboids," in *Computer Vision–ECCV 2014*. Springer, 2014, pp. 788–803.

[156] J. Smisek, M. Jancosek, and T. Pajdla, "3d with kinect," in *Consumer Depth Cameras for Computer Vision*. Springer, 2013, pp. 3–25.

[157] N. Burrus, "Rgbdemo," June 2013, http://rgbdemo.org/.

[158] C. Zhang and Z. Zhang, "Calibration between depth and color sensors for commodity depth cameras," in *Multimedia and Expo (ICME), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.

[159] W. Liu, Y. Fan, Z. Zhong, and T. Lei, "A new method for calibrating depth and color camera pair based on kinect," in *Audio, Language and Image Processing (ICALIP), 2012 International Conference on*. IEEE, 2012, pp. 212–217.

[160] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1. IEEE, 1999, pp. 666–673.

[161] I. V. Mikhelson, P. G. Lee, A. V. Sahakian, Y. Wu, and A. K. Katsaggelos, "Automatic, fast, online calibration between depth and color cameras," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 218–226, 2014.

[162] C. Raposo, J. P. Barreto, and U. Nunes, "Fast and accurate calibration of a kinect sensor," in *3D Vision-3DV 2013, 2013 International Conference on*. IEEE, 2013, pp. 342–349.

[163] A. Staranowicz, G. R. Brown, F. Morbidi, and G. L. Mariottini, "Easy-to-use and accurate calibration of rgb-d cameras from spheres," in *Image and Video Technology.* Springer, 2014, pp. 265–278.

[164] A. Canessa, M. Chessa, A. Gibaldi, S. P. Sabatini, and F. Solari, "Calibrated depth and color cameras for accurate 3d interaction in a stereoscopic augmented reality environment," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 227–237, 2014.

[165] F. Basso, A. Pretto, and E. Menegatti, "Unsupervised intrinsic and extrinsic calibration of a camera-depth sensor couple," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014, pp. 6244–6249.

[166] R. Kummerle, G. Grisetti, and W. Burgard, "Simultaneous calibration, localization, and mapping," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on.* IEEE, 2011, pp. 3716–3721.

[167] S. Miller, A. Teichman, and S. Thrun, "Unsupervised extrinsic calibration of depth sensors in dynamic scenes," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on.* IEEE, 2013, pp. 2695–2702.

[168] OpenKinect.org, "Imaging information," Nov. 2013, http://openkinect.org/wiki/Imaging_Information/.

[169] K. Konolige and P. Mihelich, "Technical description of kinect calibration," Dec. 2012, http://wiki.ros.org/kinect_calibration/technical/.

[170] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[171] J. Kramer, N. Burrus, F. Echtler, C. Herrera, and M. Parker, *Hacking the Kinect.* Springer, 2012.

BIOGRAPHICAL STATEMENT

Wei Xiang was born in Xi'an, China. He received his B.S. degree from Xidian University, China, in 2012. His research interests centre around computer vision, image processing, data mining and machine learning. Particularly, he is interested in solving traditional computer vision tasks in a large-scale setting with deep learning based methods. During his Ph.D. program, he interned at Futurewei Technologies, NVIDIA and JD.COM.