Light-weight CNN Model Design for Faster and Better Object Detector

Jing Luo

Advisor

Xiangyu Zhang

Megvii Inc (Face++)

Beijing, P.R.China

Prof. Cewu Lu

Department of Computer Science and Engineering

Shanghai Jiao Tong University

Shanghai, P.R.China

Light-weight CNN Model Design for Faster and Better Object Detector

ABSTRACT

With the development of artificial intelligence, computer vision has been one of the hottest field powered by deep learning. In this thesis, we propose a light-weight CNN architecture design for faster and better object detector by combining ShuffleNet v2 [1](an inference-efficient backbone) with Light-Head R-CNN [2] (a fast and light-weight detection framework). We also propose a novel multitask learning framework, **distributed multitask learning**, to train this detection architecture.

For the backbone network part, our team [1] concludes four heuristic principles for fast inference on the target platforms by analyzing and benchmarking various key operators. Guided by the heuristics, our team proposes an inference-efficient architecture named ShuffleNet v2, which has significantly improved accuracy/speed trade-offs over previous state-of-the-arts like MobileNet v2 and ShuffleNet v1 on both GPU and ARM platforms, and we use it as our light-weight backbone network. For the detection framework part, our team [2] investigates why typical two-stage methods are not as fast as single-stage detectors like YOLO and SSD, and proposes a new two-stage detector, Light-Head R-CNN, to address thea shortcoming in current two-stage approaches. We follow its design to make the head of detector as light as possible by using a thin feature map and a cheap R-CNN subnet (pooling and single fully-connected layer).

The ultimate ShuffleNet v2 [1] + Light-Head R-CNN [2] architecture, trained under our distributed multitask learning framework, outperforms state-of-art object detectors on COCO while keeping time efficiency. It achieves 32.8 mmAP at 109 FPS on COCO, significantly outperforming the single-stage, fast detectors like YOLO and SSD on both speed and accuracy.

Keywords: Light-weight CNN, Object Detection, Multitask Learning.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Over the past years deep learning methods have been shown to outperform previous state-of-the-art machine learning techniques in several fields, with computer vision being one of the most prominent cases. The year of 2012 witnessed the birth of deep learning era, when Alex Krizhevsky used AlexNet [7] to win that years ImageNet competition, dropping the classification error record from 26% to 15%, an astounding improvement at that time. Since then, the landscape of computer vision community has completely changed. Convolutional neural networks (CNNs) started to dominate the field of computer vision, achieving top scores on many tasks and their related competitions.

A noteworthy trait in the deep learning era is that there has been an exponential growth in the number of CNN based solutions to computer vision tasks in the past years, and this trend seems to be continuing. The evolution of object detection is a good example. It is a heavily researched topic in computer vision. The main aim of object detection task is to detect the object or a set of objects from a predefined set of classes, as well as the minimal sufficient bounding box around each object instance. Traditional detectors extract image features by using hand-engineered object component descriptors, such as HOG [8], SIFT [9], Selective Search [10], Edge Box [11]. During a long time, DPM [12] and its variants are the dominant methods among traditional object detectors. With the rapid progress of deep convolutional neural networks, CNN based object detectors have yielded a remarkable result and become a new trend in detection literature.

Benefited from the rapid development of deep convolutional networks [1,3,6,7,13–20], a great progress has been made for the object detection problem. Currently, the leading region-based object detection paradigm [2, 4, 5, 21–24] consists of two parts, namely, backbone network for image feature generation, and detection head for region proposal generation, region recognition and duplicate removal. We briefly introduce these two parts in the following section.

Chapter 2 Related Works

2.1 Backbone Network

2.1.1 Accuracy Perspective

The backbone networks for object detection are usually borrowed from the ImageNet [25]classification. In last few years, ImageNet has been regarded as a most authoritative datasets to evaluate the capability of deep CNNs. Many novel networks are designed to get higher performance for ImageNet.

AlexNet [7] is among the first to try to increase the depth of CNN. In order to reduce the network computation and increase the valid receptive field, AlexNet down-samples the feature map with 32 strides which is a standard setting for the following works. VGGNet [13] stacks 3x3 convolution operations to build a deeper network, and it still involves 32 strides in feature maps. Most of the following researches adopt VGG-like structure, and design a better component in each stage (split by stride). GoogleNet [14, 26] proposes a novel inception block to involve more diversity features. ResNet [15] adopts bottleneck design with residual sum operation in each stage, which has been proved a simple and efficient way to build a deeper neural network. ResNeXt [17] and Xception [27] use group convolution layer to replace the traditional convolution. It reduces the parameters and increases the accuracy simultaneously. DenseNet [28] densely concatenate several layers, and further reduces parameters while keeping competitive accuracy. SENet [18] introduces an architectural unit that boosts performance at slight computation cost. Another different research is Dilated Residual Network [29]which extracts features with less strides. DRN achieves notable results on segmentation, while has little discussion on object detection.

2.1.2 Speed Perspective

As building deeper and larger CNNs is a primary trend for solving major vision tasks, the most accurate CNNs usually have hundreds of layers and thousands of channels [15, 17, 26], thus requiring computation at billions of FLOPs. However, the increasing needs of running high quality deep neural networks on embedded devices encourage the study on efficient model designs.

Tuning deep neural architectures to strike an optimal balance between accuracy and performance has been an area of active research for the last several years. Both manual architecture search and improvements in training algorithms, carried out by numerous teams have resulted in dramatic improvements over early designs. Recently there has been lots of progress in algorithmic architecture exploration included hyperparameter optimization [30] as well as various methods of network pruning [31] and connectivity learning. MobileNet [20] and MobileNet v2 [6] utilizes the depthwise separable convolutions and gains adorable results among light-weight models. ShuffleNet [1, 3] introduces the channel shuffle operation to change the connectivity structure of the internal convolutional blocks, and generalized groups convolution and depthwise separable convolution in a novel form. NASNet [32], proposed by reinforcement learning and model search to explore efficient model designs, achieves comparable performance.

2.2 Detection Head

2.2.1 Accuracy Perspective

The first framework to utilize deep neural network features into detection system is R-CNN [21], which uses hand-engineered methods to generate region proposals, such as Selective Search [10], Edge Boxes [11] and MCG [33]. Then Fast R-CNN [22] is proposed to jointly train object classification and bounding box regression, which improves the performance by multi-task training. Following Fast R-CNN, Faster R-CNN [4] introduces Region Proposal Network (RPN) to generate proposals by using network features. Benefited from richer proposals, it marginally increases the accuracy of object detection and becomes a milestone of R-CNN serials detectors.

Most of the following works strengthen Faster R-CNN by bringing more computation into network. Deformable Convolutional Networks [24] are proposed to model geometric transformations by learning additional offsets without supervision. Feature Pyramid Networks (FPN) [34] exploits inherent multi-scale and pyramidal hierarchy of deep convolutional networks to construct feature pyramids. Based on FPN, Mask R-CNN [23] further extends a mask predictor by adding a extra branch in parallel with the bounding box recognition. RetinaNet [35] is another FPN based single stage detector, which involves Focal-Loss to address class imbalance issue caused by extreme foreground-background ratio.

2.2.2 Speed Perspective

The hot research topic in object detection from speed perspective is proposal free detector. YOLO and YOLO v2 [36, 37] simplifies object detection as a regression problem, which directly predicts the bounding boxes and associated class probabilities without proposal generation. SSD [38] further improves the performance by producing predictions of different scales from different layers. Unlike box-center based detectors, DeNet [39] first predicts all boxes corners, and then quickly searching the corner distribution for non-trivial bounding boxes.

Object detection literatures have been also striving to improve the speed of two-stage detectors. Back to original R-CNN, which forwards each proposal separately through the whole network, SPPNet [40] is the first detector to share the computations among candidate boxes. Both Fast/Faster R-CNN [4, 22] accelerates the network by uniforming the detection pipeline. R-FCN [5] shares computations between RoI subnetworks, which speeds up inference as a large number of proposals are utilized. Currently, Light-Head R-CNN [2] is state-of-the-art two stage object detector in terms of speed, which achieves 31.5 mmAP at 95 FPS on COCO by using a thin feature map and a cheap R-CNN subnet.

2.3 Multitask Learning

As an approach to inductive transfer that improves generalization by using the domain information contained in the training signals of related tasks as an inductive bias, multitask learning has been used successfully across all applications of machine learning, from natural language processing [41] and speech recognition [42] to computer vision [22] and drug discovery [43]. Multitask learning comes in many guises: joint learning, learning to learn, and learning with auxiliary tasks are only some names that have been used to refer to it.

In the context of deep learning, multitask learning is typically done with either hard or soft parameter sharing of hidden layers. Hard parameter sharing is the most commonly used approach to MTL in neural networks and goes back to [44]. It is generally applied by sharing the hidden layers between all tasks, while keeping several task-specific output layers. Hard parameter sharing greatly reduces the risk of overfitting. In fact, [45] shows that the risk of overfitting the shared parameters is an order N – where N is the number of tasks – smaller than overfitting the task-specific parameters, i.e. the output layers. In soft parameter sharing on the other hand, each task has its own model with its own parameters. The distance between the parameters of the model is then regularized in order to encourage the parameters to be similar. [46] for instance use the $l2$ norm for regularization, while [47] use the trace norm.

While most of recent deep learning approaches have used hard and soft parameter sharing – either explicitly or implicitly – as part of their model, a few papers have looked at developing better mechanisms for multitask learning in deep neural networks. In multitask learning for computer vision, approaches often share the convolutional layers, while learning task-specific fully-connected layers. [48] improve upon these models by proposing Deep Relationship Networks. In addition to the structure of shared and task-specific layers, they place matrix priors on the fully connected layers, which allow the model to learn the relationship between tasks, similar to some of traditional Bayesian models. This approach, however, still relies on a pre-defined structure for sharing, which may be adequate for well-studied computer vision problems, but prove errorprone for novel tasks. Starting at the other extreme, [49] propose a bottom-up approach that starts with a thin network and dynamically widens it greedily during training using a criterion that promotes grouping of similar tasks. However, the greedy method might not be able to discover a model that is globally optimal, while assigning each branch to exactly one task does not allow the model to learn more complex interactions between tasks. [50] start out with two separate model architectures just as in soft parameter sharing. They then use what they refer to as cross-stitch units to allow the model to determine in what way the task-specific networks leverage the knowledge of the other task by learning a linear combination of the output of the previous layers.

In conclusion, current approaches for multitask learning are all applied on single machine by designing a uniform architecture while adding more branches and units. It is quite inflexible in terms of task amount and computation capacity. Thus, we try to maximize the efficiency and capacity of multitask learning by proposing a distributed architecture with a parameter center – **distributed multitask learning**.

Chapter 3 Approach

Our light-weight CNN architecture design for faster and better object detector includes light-weight backbone network and light-weight detection head.

First, we choose ShuffleNet v2 [1] proposed by our team as our light-weight backbone network due to its novel performance on both accuracy and speed in ImageNet classification. By analyzing the key components of two state-of-the-art light-weight CNN models – ShuffleNet [3] and MobileNet v2 [6], [1] perform a series of controlled evaluations to benchmark the related operations, from which our team concludes three heuristic principles for fast inference in our platforms. Next, guided by the heuristics, [1] propose a new inference-efficient architecture – ShuffleNet v2, which is demonstrated much faster than the previous counterparts on both GPU and ARM platforms and achieves superior accuracy over the previous state-of-the-arts, for example, on the task of ImageNet classification it outperforms ShuffleNet v1 by 2% and MobileNet v2 1.5% at the complexity of 140MFLOPs.

For light-weight detection head, as Faster R-CNN [4] and R-FCN [5] are slow due to their heavy heads, we follow the design of detection framework in Light-Head R-CNN [2] to build an efficient yet accurate two-stage detector. Specifically, we apply a large-kernel separable convolution to produce thin feature maps with small channel number. This design greatly reduces the computation of following RoI-wise subnetwork and makes the detection system memory-friendly. A cheap single fully-connected layer is attached to the pooling layer, which well exploits the feature representation for classification and regression.

Last, we propose a novel multitask training framework, **distributed multitask training**, to perform training on this light-weight CNN architecture. We establish a distributed training framework, including PS (parameter center) and TS (task site), in which a PS holds and updates the parameters in a neural network, while TSes commit changes and download task-specific parameters. We demonstrate that our distributed multitask learning method can enhance the performance of our light-weight detector by

0.3∼1 mmAP on COCO detection under different multitask settings.

3.1 Backbone Network: ShuffleNet v2

3.1.1 Heuristic Priciples for Fast Inference

The backbone networks for object detection are usually pretrained on ImageNet [25]classification, because better classification result on large image database usually indicates better capability of feature extraction. Earlier works such as [7, 13–15, 17] usually focus on absolute accuracy. However, with the development of mobile devices such as smartphones, people start to focus on accuracy-efficiency, i.e. seeking for more accurate models under given theoretical complexity budget, and light-weight convolutional neural network gradually becomes one of the central topics in modern practical visual processing systems.

One of the estimation metrics of CNN model complexity is the number of float-point operations, or FLOPs[1]. It is undoubtedly one of the crucial factors to the actual inference time of neural networks, which has been widely used as an evaluating metric in architecture tuning [3,6,15,17,18,20,27,28,51–56] and neural network simplication [31, 57–67]. The more FLOPs a model contains, the more complex it tends to be. However, a lot of previous works such as MobileNet v2 [6] and NASNet [32] imply that different models with comparable theoretical complexity may have very different inference speeds in reality. For example, in [62, 65] the authors point that structured pruning schemes usually result in more actual speedup than unstructured ones, and MobileNet v2 [6] is much faster than NASNet-A [32] even though these two models are under comparable FLOPs. Here an important question is raised: given a FLOPs budget, which neural network architecture may lead to faster inference speed?

[1] give a seemingly perfect answer to the above question. As actual running time is related to a lot of factors such as device infrastructure, memory/cache hierarchy, computational model and implementation optimization (blocking, SIMD, data layout, etc), it is nearly impossible to conclude a simple rule for inference efciency.

[1]In the paper, *FLOPs* follows the same definition as [3], i.e. the number of multiply-adds.

With the rapid development of computational devices, it is harder and harder to design an actual efficient structure without taking target device properties into account. For example, some earlier model acceleration methods [58–60] report considerable actual speedup via tensor decomposition; however, recent work [31] reproduces the former [59] and points out such factorization is even slower on GPU although the model has 75% fewer FLOPs. Our team inspects the phenomenon and finds out the cause - latest NVIDIA GPUs and the corresponding CUDNN [68] library have optimized only for large 3×3 convolutions instead of the factorized ones. Thus, our team chooses to empirically investigate heuristic design principles for efficient network architectures on modern platforms, with GPU and ARM as the target computing devices:

Deep learning platform. MegBrain, the product-level closed-source framework of Megvii. Compared with most of the open-source solutions, this framework is more efficient and considered to be more suitable for practical situations. Full optimization options (e.g. tensor fusion, which is used to reduce the overhead of small operations) are switched on.

Newtork models. ShuffleNet v1 [3] and MobileNet v2 [6]. The size of input image is 224×224.

GPU device. A single NVIDIA GeForce GTX 1080Ti. The convolution library is CUDNN 7.0 [68]. The benchmarking function of CUDNN to select the fastest algorithms for dierent convolutions respectively is also activated.

ARM device. Qualcomm Snapdragon 810 (a highly-optimized Neon-based implementation). All results are evaluated with single thread and averaged on 100 times experiments.

Table 3.1 shows the result of the empirical study. It is easy to see that convolution is the most time-consuming operation in the neural networks. Surprisingly, element-wise operations (such as AddTensor, ReLU, etc) also occupy a considerable fraction of inference time in both structures and platforms, even though their theoretical complexities

Setting	Conv	Data	Element-wise	Shuffle	Other
ShuffleNet v1 on GPU	50%	20%	15%	5%	10%
ShuffleNet v1 on ARM	87%	1%	4%	0%	8%
MobileNet v2 on GPU	54%	22%	23%	-	1%
MobileNet v2 on ARM	89%	0.3%	10.5%	-	0.2%

Table 3.1 Inference time occupations for different operators in ShuffeNet v1 (1$\times$, $g = 3$) [3] and MobileNet v2 (1$\times$) [6].

are negligible. The cost of other operators is relatively small.

Analysis and discussion. According to the above empirical studies, our team concludes three heuristic principles for inference-efficient CNN model design as follows:

- **P1:** Convolution in an inference-efficient CNN model should keep the number of output channels the same as input channels.

  As modern light-weight networks usually adopt depthwise separable convolution [3, 6, 20, 27] whose pointwise part occupies most of the complexity, here we mainly consider pointwise convolution (i.e. 1×1 convolution). Two parameters control the shape of pointwise convolution kernels: the number of input channels c_1 and output channels c_2. Intuitively, given the total FLOPs budget $C = hwc_1c_2$ where h and w define the spatial size of the feature map, it is beneficial to produce efficient inference if $c_1 = c_2$. Suppose a simple case when the entire input/output feature maps and kernel weights could be fully cached, then the number of memory-access operations (MACs) during the inference is:

$$\text{MACs} = hw(c_1 + c_2) + c_1c_2 \geq 2\sqrt{hwC} + \frac{C}{hw}. \qquad (3\text{-}1)$$

  It is easy to see that MACs are minimized if and only if $c_1 = c_2$, while smaller MACs usually implies more efficient execution. In practical situations, the cache is usually not large enough to hold the entire feature maps, and complex cache blocking strategies are adopted in the runtime library to reduce memory-access requirements [69], which may cause different effects from the above theoretical analysis. However, experiments in [1] indicate that it would result in significantly faster inference if the values of c_1 and c_2 are close than they are much different.

- **P2:** Dense convolutions are better than group convolutions to build an inference-efficient model.

  In the field of light-weight CNN design, group convolutions are commonly used [3, 17, 53, 55] to reduce the complexity of traditional dense convolutions. However, after some studies on the implementation, our team finds several drawbacks of group convolutions which may lead to inefficient inferencing: first, given the constraint total FLOPs, larger number of groups results in more feature map channels and increasing the overhead brought by the additional MACs; second, to our knowledge, group convolutions are less flexible to cache/shared memory blocking, which makes it not easy to optimize.

  To study how group convolutions affect the actual efficiency, [1] stack 10 point-wise group convolutional layers as the benchmark model. Our team tests a variety of group numbers/input resolutions/FLOPs on both GPU and ARM and repeatedly stacks the structure unit for 10 times to benchmark the speed. The results prove our guess – convolutions with large group numbers are still much slower under the same FLOPs, for example, $group = 8$ cost around 150% more time on GPU and up to 30% more on ARM than the counterpart dense convolutions.

- **P3:** The number of fragments in an inference-efficient model should be as small as possible.

  Recent years have witnessed the development of some of the state-of-the-art CNN models (e.g *Inception* series [14, 16, 26, 70] and auto-generated architectures [32, 71, 72]) adopt "multi-path" architecture, which involves a lot of small operators (called "fragmented operators" here) in each building block instead of a few large ones. For example, in NASNet-A [32] the number of fragments (i.e. the number of individual convolutions or pooling operations in one building block) is 13, in contrast, in other structures like ResNet [15] the number is 2 or 3. Though such multi-path design may bring some benefits on accuracy, we intuitively think that too many fragments make it inefficient for inference – the side effects are just similar to the case of group convolutions. Furthermore, on GPU device, fragments also harm paralleling and introduce extra overheads such as kernel launching and synchronization. Empirical study in [1] proves our guess.

Let's go over recent advances in light-weight neural network architectures [3, 6, 20, 27, 32, 71, 72], most of which are designed following the accuracy-efficient cues (i.e. seeking for the most accurate model at given constraint FLOPs); however from inference-efficiency perspective, those models could be suboptimal as their structures conflict with some principles. For example, ShuffleNet v1 [3] violates P2 (group convolutions) and P1 (bottleneck-like building block); MobileNet v2 [6] violates P1 (inverted bottleneck); some auto-generated structures such as NASNet [32] mainly violate P3 (too many fragments and random connections). In the next subsection, we introduce our backbone choice, ShuffleNet v2 [1], an inference-efficient CNN architecture designed under the guidance of these heuristics.

3.1.2 ShuffleNet v2: an Inference-efficient Architecture

The building block of ShuffleNet v2 is inspired by the previous state-of-the-art work ShuffleNet v1 [3]. First of all, let's have a brief review on it. [3] believe that one of the challenges to light-weight models is the very limited feature map channels under constraint computation complexity. To address the issue, two techniques are adopted in [3] – pointwise group convolutions and bottleneck-like structures, which allows to generate more channels without significantly increasing the FLOPs. Then the shuffle operation is introduced to enable the cross talk between different feature groups. Fig 3.1(a) illustrates the building blocks of ShuffleNet v1.

According to what is discussed in Sec. 3.1.1, both techniques (pointwise group convolutions and bottlenecks) violate the fast-inference principles. How to efficiently generate "thick" feature maps without passing them through convolution layer? A straightforward idea is to introduce channel concatenation operation. However, a new problem raises: since channel concatenation needs at least two-way inputs, according to **P1**, all

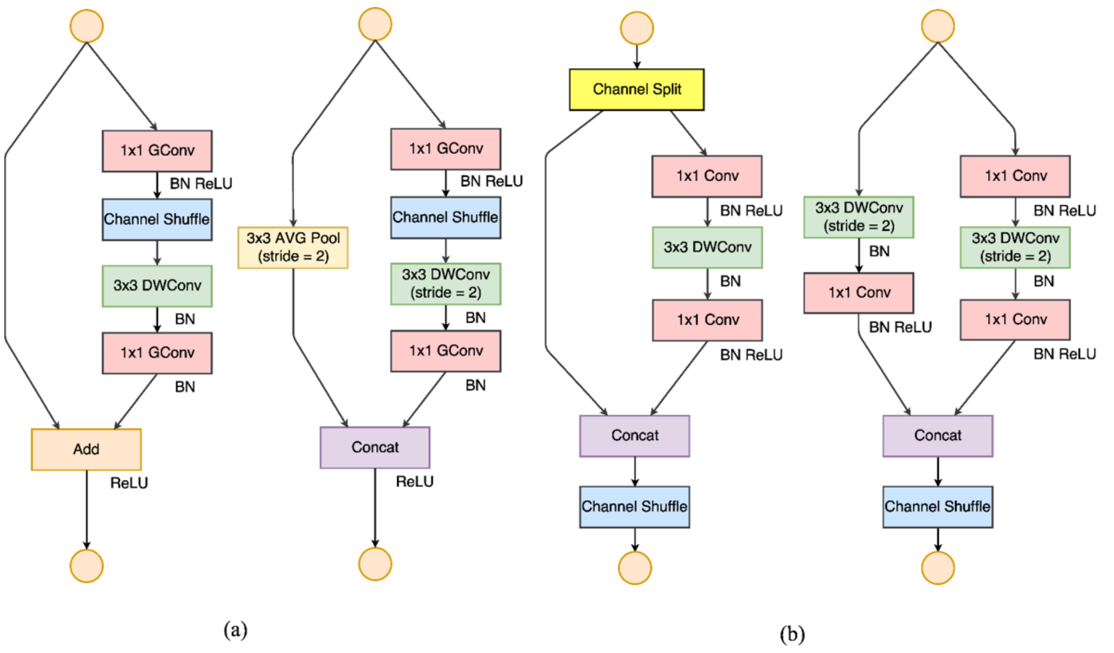

Figure 3.1 Building blocks. (a) ShuffleNet v1 [3]; (b) ShuffleNet v2 [1]. DWConv – depthwise convolution. GConv – group convolution.

convolutions must keep the number of input channels; thus the number of channels after concatenation will be at least twice the number of input channels. In other words, feature map thickness grows exponentially with the increasing of depth, which is unacceptable.

In order to get out of the dilemma, our team introduces a new operator – *channel split* – as an inverse to channel concatenation. For each building block, suppose the input feature map has c channels, after channel split, there are two branches with c' and $c - c'$ channels respectively. Following **P3**, we leave one branch identity and put several convolutions only in another branch. All convolutions keep the same number of input and output channels to satisfy **P1**. Then a channel concatenation operator is introduced to generate the output feature map, whose number of channels right equals to the number of input channels. As a result, we get “thicker” feature maps (compared with convolution outputs) without breaking any of the principles nor changing the shape of input feature map.

We propose a new building block, say *ShuffleNet v2*, as illustrated in Fig 3.1(b, left). Apart from the points introduced above, there are several other differences from ShuffleNet v1 (Fig 3.1(a, left)): first, following **P2**, our team cancels group convolutions, so *channel shuffle* operation is not needed in the right branch; instead, our team moves it after the concatenation to mix up information from the two branches. Second, residual shortcut is also removed; most of other element-wise operations like ReLU and *depth-wise convolutions* exist only in one branch to merge as many element-wise operations as possible. For the building block with $stride = 2$, our team simply removes the channel split operator – thus the output channels double by nature – and adds a pair of depthwise separable convolutions to perform downsampling (see Fig 3.1(b, right)).

Our team repeatedly stacks ShuffleNet v2 building blocks to construct the whole network, basically following the top-level architecture of [3] except for one difference: here we insert an additional 1×1 convolution layer right before global averaged pooling to mix up features, which is not needed in ShuffleNet v1. Similar to [3], the number of channels are scaled in each blocks to generate networks of different complexities, marked as 0.5×, 1×, etc. Details are presented in Table 3.2. It is worth a note that in ShuffleNet v2, successive 3 channel operators – channel concatenation, channel shuffle and channel split – could be fused as a complex operation, which is beneficial to fast inferencing. In the experiment, c' is set to $0.5 \times c$ for simplicity.

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1 MaxPool	112×112 56×56	3×3 3×3	2 2	1	24	24	24	24
Stage2	28×28 28×28		2 1	1 3	48	116	176	244
Stage3	14×14 14×14		2 1	1 7	96	232	352	488
Stage4	7×7 7×7		2 1	1 3	192	464	704	976
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# Weights					1.4M	2.3M	3.5M	7.4M

Table 3.2 ShuffleNet v2 [1] architectures at different complexities.

3.2 Detection Head: Light-Head R-CNN

3.2.1 Heavy Head in Current Two-stage Detectors

As we discussed in the start of this chapter, conventional two-stage object detectors like Faster R-CNN [22] and R-FCN [5] usually involve a heavy head, which has negative effects on the computational speed. “Head” in this thesis refers to the structure attached to our backbone base network.

More specifically, Faster R-CNN employs two large fully connected layers or all the convolution layers in ResNet 5-th stage [4, 73] for per RoI’s recognition and regression. It is time-consuming in terms of per-region prediction and even gets worse when a large number of proposals are utilized. In addition, the number of feature channels after ROI pooling is large, which makes the first fully connected layer consume a large memory and potentially influence the computational speed.

Different from Fast/Faster R-CNN which apply a per-region subnetwork many times, Region-based fully convolutional networks (R-FCN) [5] tries to share computation across all RoIs. However, R-FCN needs to produce a very large additional score maps with $\#classes \times p \times p$ (p is the followed pooling size) channels, which is also memory and time consuming. The heavy head design of Faster R-CNN or R-FCN makes the two-stage approach less competitive than single-stage detector.

Having these issues in mind, we decide to follow the design of Light-Head R-CNN [2] to utilize light-head design in our light-weight model. The improvments are based on two components: R-CNN subnet and RoI warping.

3.2.2 Light-Head Design for Light-weight Detector

R-CNN subnet

As it shows in Figure 3.2 (A), Faster R-CNN adopts a powerful R-CNN and this subnet utilizes two large fully connected layers or whole ResNet stage 5 [4, 73] as a second stage classifier, which is beneficial to the detection performance. Therefore Faster R-CNN and its extensions perform leading accuracy in the most challenging benchmarks

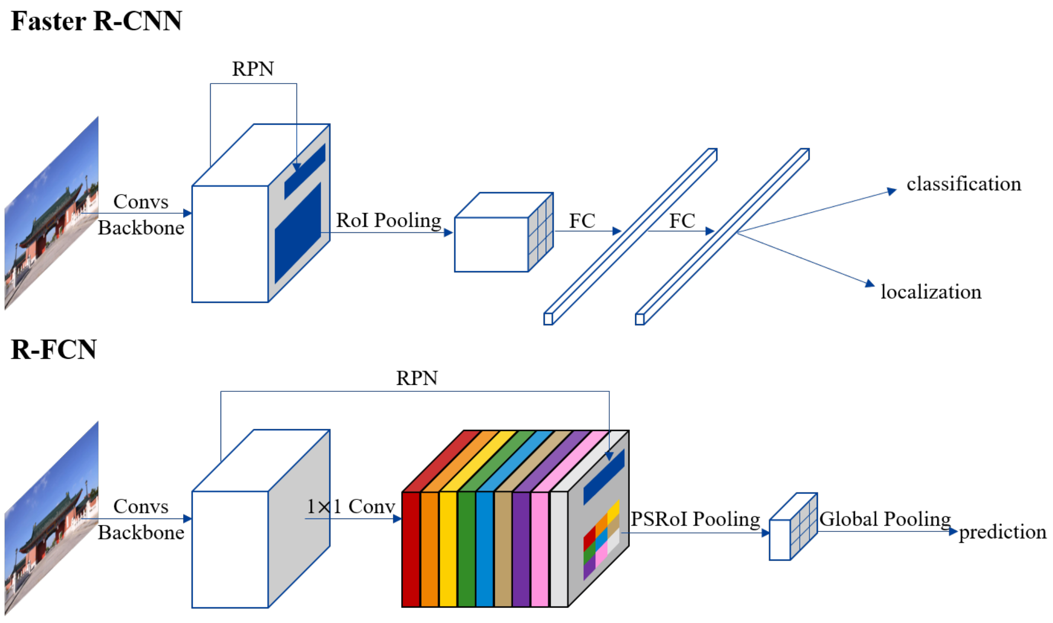

Figure 3.2 Overview of Faster R-CNN [4] and R-FCN [5].

like COCO. However, the computation could be intensive especially when the number of object proposals is large.

To speed up RoI-wise subnet, R-FCN first produces a set of score maps for each region, whose channel number will be $\#classes \times p \times p$ (p is the followed pooling size), and then pool along each RoI and average vote the final prediction, as it shows in Figure 3.2 (B). Using a computation-free R-CNN subnet, R-FCN gets comparable results by involving more computation on RoI shared score maps generation.

As mentioned above, both Faster R-CNN and R-FCN have heavy heads and the heavy heads are at different positions. From **accuracy** perspective, although Faster R-CNN is good at RoI classification, it usually involves global average pooling in order to reduce the computation of first fully connected layer, which is harmful for spatial localization. For R-FCN, it directly pools the prediction results after the position-sensitive pooling and the performance is usually not as strong as Faster R-CNN without RoI-wise computation layers.

From **speed** perspective, Faster R-CNN passes every RoI independently through a computation-costly R-CNN subnet, which slows down the network speed especially

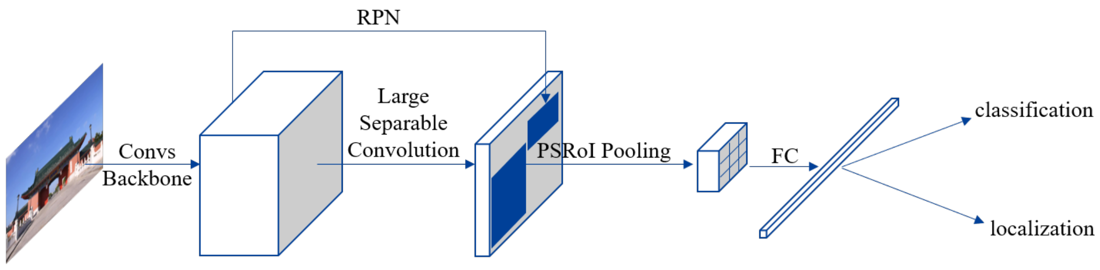

Figure 3.3 Overview of Light-Head R-CNN [2]. It builds "thin" feature maps before RoI warping, by large separable convolution and adopts a single fully-connected layer with 2048 channels in our R-CNN subnet.

when the number of proposals is large. R-FCN uses cost-free R-CNN subnet as a second stage detector. But as R-FCN needs to produce a very large score map for RoI pooling, the whole network is still time consuming.

Having these issues in mind, in the new Light-Head R-CNN [2], our team proposes to utilize a simple, cheap fully-connected layer for our R-CNN subnet, which makes a good trade-off between the performance and computational speed. Figure 3.3 provides the overview of Light-Head R-CNN. As the computation and memory cost for the fully-connected layer also depends on the number channel maps after RoI operation, we next introduce how our team designs the ROI warping.

Thin feature maps for RoI warping

Before feeding proposals into R-CNN subnet, RoI warping is involved to make the shape of feature maps fixed.

In the Light-Head R-CNN, our team proposes to generate the feature maps with small channel number (thin feature maps), followed by conventional RoI warping. In our experiments, we find that RoI warping on **thin feature maps** will not only improves the accuracy but also saves memory and computation during training and inference. Considering PSRoI pooling on thin feature maps, we can bring more computation to strengthen R-CNN and decrease the channels. In addition, if we apply RoI pooling on our thin feature maps, we can reduce R-CNN overhead and abandon Global Average Pooling to improve performance simultaneously. Moreover, without losing time efficiency, large convolution can be involved for thin feature map production.

3.2.3 Architecture Details

Following the above discussions, we present our implementation details for general object detection. The pipeline of our approach is in Figure 3.3.

Basic feature extractor. We utilize ShuffleNet v2 [1] proposed by our team as the backbone network due to its fast inference speed and great performance on ImageNet classification. For comparison, we also follow the setting in [2] and test the performance of this light-weight detector on Inception [70].

Thin feature maps. Following the setting in [2], we apply large separable convolution layers [26, 74] on C_5, the structure is shown in Figure 3.4. In our approach, we let $k = 15$, $C_{mid} = 256$ for setting L, and $k = 15$, $C_{mid} = 64$ for setting S. We also reduce the C_{out} to $10 \times p \times p$ which is extremely small compared with the value $\#classes \times p \times p$ used in R-FCN. Due to benefits from the larger valid receptive field caused by large kernel, the feature maps we pooled on contain more information and are more powerful.

Region Proposal Network. RPN is a sliding-window class-agnostic object detector that uses features from C_4. RPN pre-defines a set of anchors, which are controlled by several specific scales and aspect ratios. In our models, we set three aspect ratios $\{$1:2, 1:1, 2:1$\}$ and five scales $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ to cover objects of different shapes. Since there are many proposals heavily overlapping with each other, non-maximum suppression (NMS) is used to reduce the number of proposals before feeding them into RoI prediction subnetwork. We set the IoU (intersection-over-union) threshold of 0.7 for NMS. Then we assign anchors training labels based on their IoU ratios with ground-truth bounding boxes. If the anchor has IoU over 0.7 with any ground-truth box, it will be set a positive label. Anchors which have highest IoU for ground-truth box will also be assigned a positive label. Meanwhile, if extra anchors have IoU less than 0.3 with all ground-truth box, their labels will be negative. More details can be

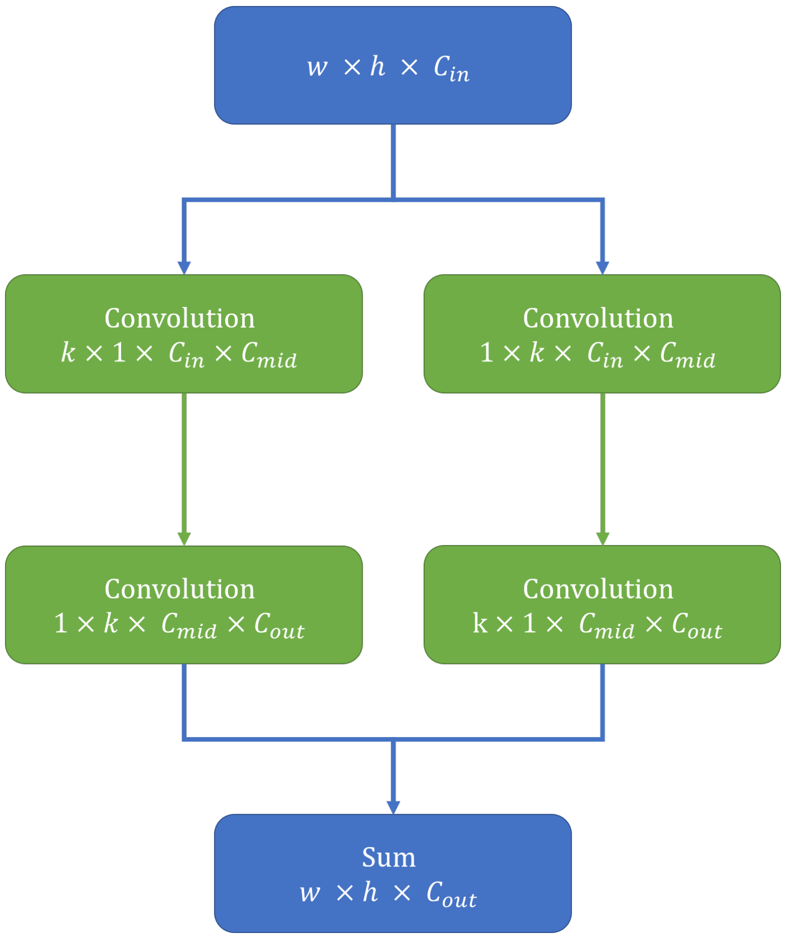
$w \times h \times C_{in}$
Convolution $k \times 1 \times C_{in} \times C_{mid}$
Convolution $1 \times k \times C_{in} \times C_{mid}$
Convolution $1 \times k \times C_{mid} \times C_{out}$
Convolution $k \times 1 \times C_{mid} \times C_{out}$
Sum $w \times h \times C_{out}$

Figure 3.4 Large separable convolution performs a $k \times 1$ and $1 \times k$ convolution sequentially. The computational complexity can be further controlled through C_{mid}, C_{out}.

referred to [4].

R-CNN subnet. According to [2], we only employ a single fully-connected layer with 2048 channels (without dropout) in R-CNN subnet, followed by two sibling fully connected layer to perform RoI classification and regression. Only 4 channels are applied for each bounding box location because we share the regression between different classes. Benefited from the powerful feature maps for RoI warping, a simple Light-Head R-CNN can also achieve remarkable results, while maintaining the efficiency.

3.3 Training Framework: Distributed Multitask Learning

Besides combining ShuffleNet v2 [1] and Light-Head R-CNN [2] to produce our light-weight object detector, we also propose a novel training framework in this thesis – distributed multitask learning.

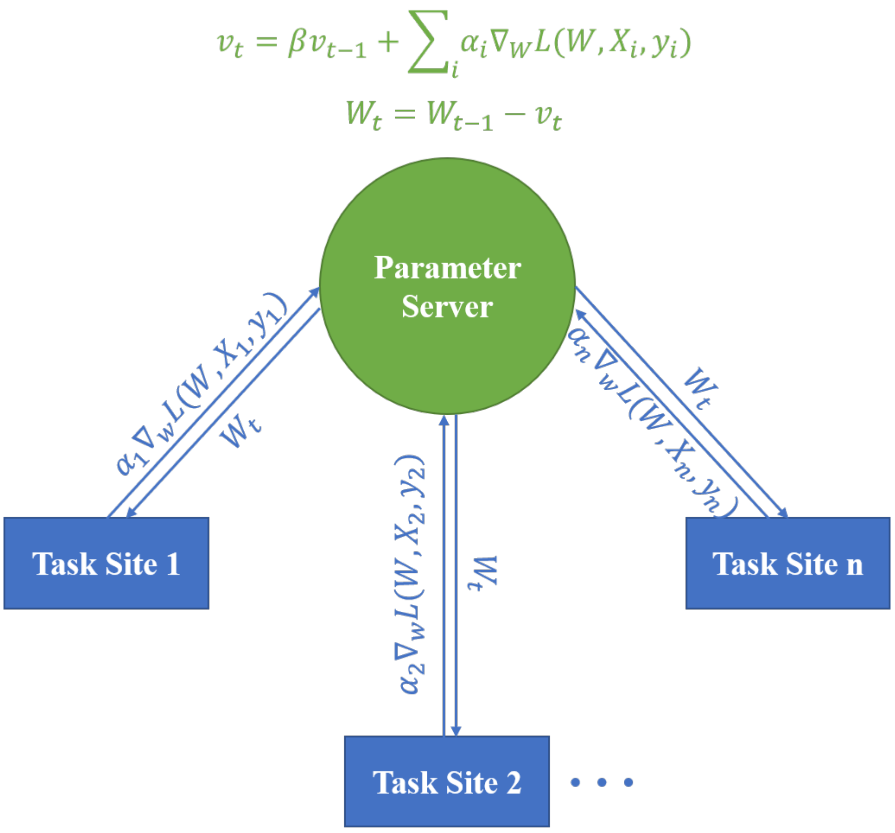
$v_t = \beta v_{t-1} + \sum_i \alpha_i \nabla_W L(W, X_i, y_i)$
$W_t = W_{t-1} - v_t$
Parameter Server
$\alpha_1 \nabla_W L(W, X_1, y_1)$
W_t
$\alpha_2 \nabla_W L(W, X_2, y_2)$
W_t
$\alpha_n \nabla_W L(W, X_n, y_n)$
W_t
Task Site 1
Task Site 2
Task Site n
• • •

Figure 3.5 The framework of distributed multitask learning. Remote momentum optimizers in different task sites commit weight change to the server and fetch parameters back in next iteration.

Conventional multitask learning usually relies on sharing some low-level layers among all tasks, while assigning several task-specific output layers to different tasks and combining their loss functions for joint training, which can greatly reduce the risk of overfitting.

However, this old form of multitask learning can't find ways to resist the following drawbacks:

- There can only be one data input source, namely, all tasks share the same input data. But chances are that some data have labels for only a proportion of the tasks, which means these data can't be utilized for joint training.
- The whole network tends to get very large if more tasks join the training, which may result in low training speed under limited resource budget.
- The learning rate must be the same for all parameters in the whole network, while the suitable learning rate may varies from task to task.
- Due to the unification of different tasks into a single one, the training must be done on one machine.

Having these issues in mind, we propose a much more flexible and efficient training method – distributed multitask learning (Figure 3.5).

The training framework can be seen as a distributing generalization of *Momentum Optimizer* on single machine, because it is commonly used in vision tasks. But the whole framework can be adapted easily to other optimzing rules. Momentum [75] is a method that helps accelerate SGD in the relevant direction and dampens oscillations by adding a fraction β as the update vector of the past time step to the current update vector:

$$\begin{aligned} v_t &= \beta v_{t-1} + \alpha \nabla_W L(W, X, y) \\ W_t &= W_{t-1} - v_t \end{aligned} \tag{3-2}$$

Our distributed multitask learning framwork consists of a parameter server (PS) and several task site(TS). The training procedure goes as follows:

1. PS starts and waits for the connection of TS.
2. Training in TS starts and Remote Optimizer in TS connects to PS through the IP address of PS. TS makes registration and submits the parameter names and their initialization values.

3. After the number of registered tasks reaches some level, PS starts to send training start signals to all of the registered task sites.

4. Distributed learning starts in each site and after one iteration of backpropagation, Remote Optimizer commits the value of $\alpha_i \nabla_W L(W, X_i, y_i)$ to PS.

5. PS updates the parameter value by the formula:

$$\begin{aligned} v_t &= \beta v_{t-1} + \sum_i \alpha_i \nabla_W L(W, X_i, y_i) \\ W_t &= W_{t-1} - v_t \end{aligned} \tag{3-3}$$

6. Remote Optimizer pulls the parameter to local TS and starts next loop.

As we can see, our proposed multitask learning framework solves the drawbacks of conventional multitask learning perfectly:

- This framework can allow different tasks use different data sources. As it is not "one" task that unifies all, but truly "multiple" task learning as it focuses on the parameter name rather than network graph.

- With more tasks joining, the size of networks used for previous tasks stay the same, which has little impact on the training speed.

- Learning rate can vary from tasks to tasks, which adds flexibility to the whole training process.

- Training needs to be done on different machines to accelerate the efficiency.

Here we propose two promising usage of this framework to enhance the performance of current models on computer vision tasks.

Joint training of image classification and object detection. The backbone network of detetction is usually borrowed from the pretrained models on ImageNet [76] then finetuned on the detection dataset such as COCO [77]. A notable phenomenon is that the classification performance of backbone model decreases with the training on object detection. “Networks are easy to forget!” Chances are that the training converges to a plain local optimum in single detection training. With the assistance of image classification training, the training process may find a better convergence point.

Multiscale training of object detection. Image input scale is one of key factors in object detection. We can utilize this framework to perform multiscale training, to make the detection framework see more images at different scales.

Chapter 4 Experiment

4.1 Performance Study: ShuffleNet v2 + Light-Head R-CNN

After combining ShuffleNet v2 [1] and Light-Head R-CNN [2], in this section, we evaluate our light-weight detector on COCO [77, 78] dataset, which has 80 object categories. There are 80k *train* set and 40k *validation* set, which will be further split into 35k *val-minusmini* and 5k *mini-validation* set. Following the common setting, we combine the *train* set and the *val-minusmini* to obtain the 115K images for training use the 5K *mini-validation* images for validation.

We use AP (averaged precision over intersection-over-union thresholds), the standard COCO metrics to evaluate our approach. To make comparison with ShuffleNet v2 and fully validate the efficiency of Light-Head R-CNN, following [2] we produce an efficient bottleneck Xception like network with 34.1 top 1 error (at 224×224 in ImageNet) to evaluate our methods. The backbone network architecture is shown in Table 4.1. Following Xception design strategies, we replace all convolution layer in bottleneck structure with channel-wise convolution. However we do not use the pre-activation design which is proposed in identity mappings [52] because of shallow network.

More specifically, we have the following changes for the fast inference speed:

1. We abandon $\grave{a}$trous algorithm in our fast models, because it involves much computation compared with small backbone.

2. We set RPN convolution to 256 channels, which is half of original used in Faster R-CNN and R-FCN.

3. We apply large separable convolution with $kernel\_size = 15, C_{mid} = 64, C_{out} = 490$ $(10 \times 7 \times 7)$. Since the middle channels we used is extremely small, large kernel is still efficient for inference.

Layer	Output size	KSize	Stride	Repeat	Output channels
Image	224×224				
Conv1	112×112	3×3	2	1	24
MaxPool	56×56	3×3	2		
Stage2	28×28		2	1	128
	28×28		1	3	128
Stage3	14×14		2	1	256
	14×14		1	7	256
Stage4	7×7		2	1	512
	7×7		1	3	512
GlobalPool	1×1	7×7			
FC					1000
Complexity					145M

Table 4.1 An efficient Xception like architecture for comparison experiment. The complexity is evaluated with FLOPs.

4. We adopt PSPooling with alignment technique as our RoI warping, since it reduces the pooled feature map channels by $k \times k$ times (k is the pooling size). We also find that the detector will obtain better results if we involve RoI-align.

For training we use *train+val* set in COCO except for 5000 images from *minival* set, while using the *minival* set to benchmark. The benchmark metric is COCO standard *mmAP*, i.e. the averaged mAPs at the box IoU thresholds from 0.5 to 0.95. We compare our light-weight model with recent fast detectors like YOLO, SSD, and DeNet. We also compare our ShuffleNet v2 with other three light-weight models: Xception, ShuffleNet v1 and MobileNet v2.

Results are evaluated on COCO test-dev datasets, only one batch of four images is adopted and we absorb the Batch-normalization [16] for fast inference. We also benchmark the inference time on GPU. For fair comparison the batch size is set to 4 to ensure almost full GPU utilization rate. As shown in Table 4.2, our method (ShuffleNet v2 + Light-Head R-CNN) gets 31.8 mmAP at 109 FPS on MS COCO, significantly outperforming the current fast detectors like YOLO and SSD.

Model	backbone	test size	speed(fps)	mmAP
YOLO V2	Darknet	448/448	40	21.6
SSD	VGG	300/300	58	25.1
SSD	ResNet 101	300/300	16	28.0
SSD	ResNet 101	500/500	8	31.2
DSSD [79]	Resnet101	300/300	8	28.0
DSSD	ResNet 101	500/500	6	33.2
R-FCN	ResNet 101	600/1000	11	29.9
DeNet	ResNet 34	512/512	83	29.4
Light Head R-CNN	Xception*	800/1200	95	31.5
Light Head R-CNN	ShuffleNet v1	800/1200	76	29.9
Light Head R-CNN	MobileNet v2	800/1200	94	30.0
Light Head R-CNN	ShuffleNet v2	800/1200	**109**	**31.8**

Table 4.2 Comparisons of our light and fast detector results on COCO test-dev. By combining ShuffleNet v2 [1] and Ligh-Head R-CNN [2], the light-weight detector achieves superior performance on both accuracy and speed.

4.2 Performance Study: Distributed Multitask Training

In this section, we aim at studying the effects of our proposed multitask training method. We propose two methods of multitask training to enhance the performance of detection framework:

1. Joint training of ImageNet classification and COCO detection.
   (a) Task 1: Train ImageNet classification using ShuffleNet v2 model
   (b) Task 2: Train COCO detection using ShuffleNet v2 + Light-Head R-CNN
2. Multiscale training of COCO detection.
   (a) Task 1: Train COCO detection using ShuffleNet v2 + Light-Head R-CNN (Input size: 800/1333)
   (b) Task 2: Train COCO detection using ShuffleNet v2 + Light-Head R-CNN (Input size: 600/1000)

For COCO detection training, we use *train+val* set in COCO except for 5000 images from *minival* set, while using the *minival* set to benchmark. The benchmark metric is

Method	Training Setting	mmAP
ShuffleNet v2 + Light-Head R-CNN	On Single Machine	31.8
ShuffleNet v2 + Light-Head R-CNN	Jointly Train with Classification	32.1
ShuffleNet v2 + Light-Head R-CNN	Multiscale Training	32.8

Table 4.3 Results of distributed multitask learning on COCO test-dev. Our light-weight detector achieves 0.3$\sim$1 mmAP under our distributed multitask learning framework.

COCO standard *mmAP*, i.e. the averaged mAPs at the box IoU thresholds from 0.5 to 0.95.

For ImageNet classification training, we follow the training setting in [3], except that we set the learning rate of ImageNet classification always equal to that of COCO detection. In addition, in order to avoid training unstability, we use warm-up training for the first 1k iterations.

As shown in Table 4.3, distributed multitask learning can enhance the results at most 1 mmAP on COCO detection, which is a notable improvement compared to conventional multitask training methods.

4.3 Demo: COCO Object Detection

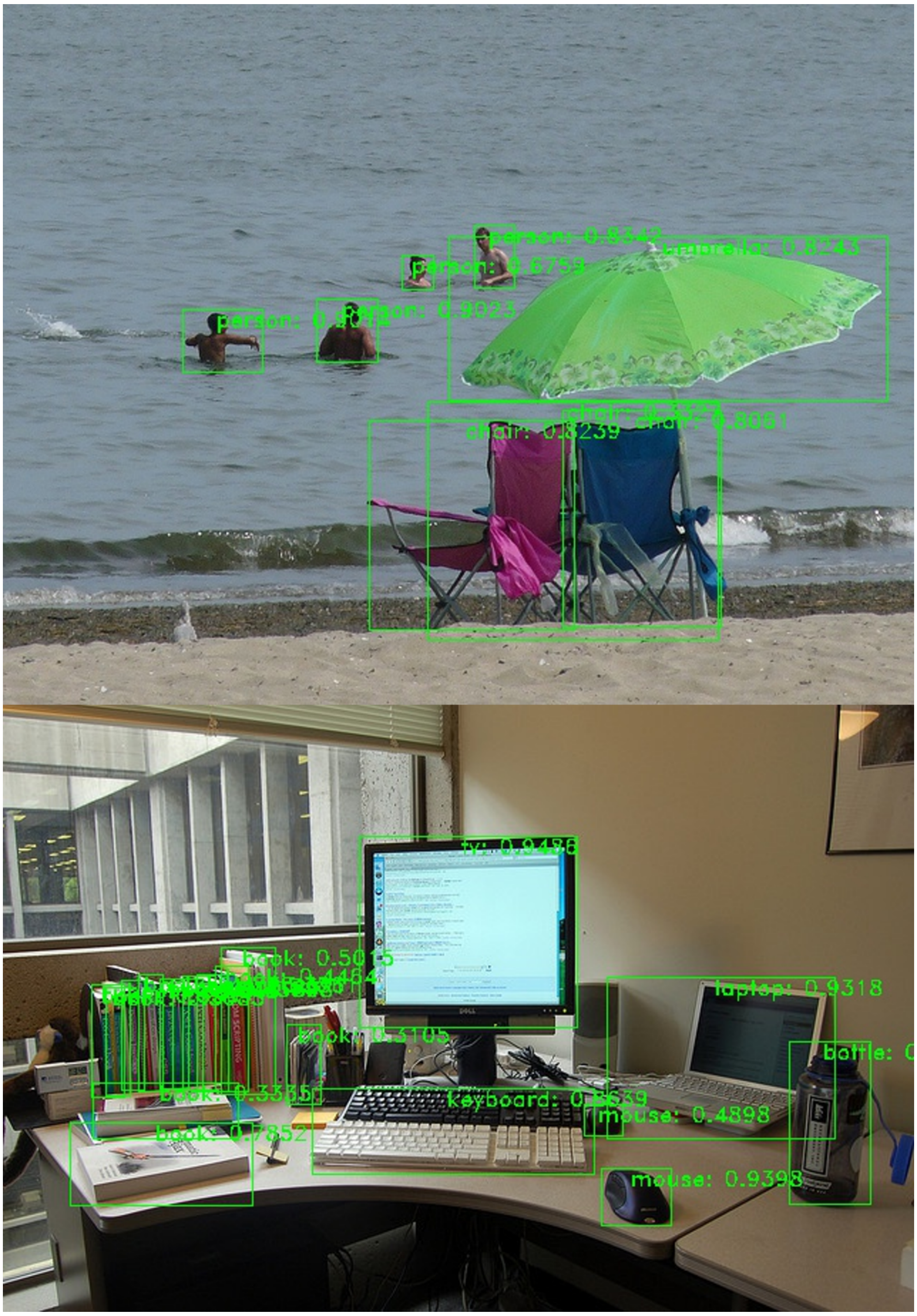

Figure 4.1 Representative results of our light-weight detector (1)

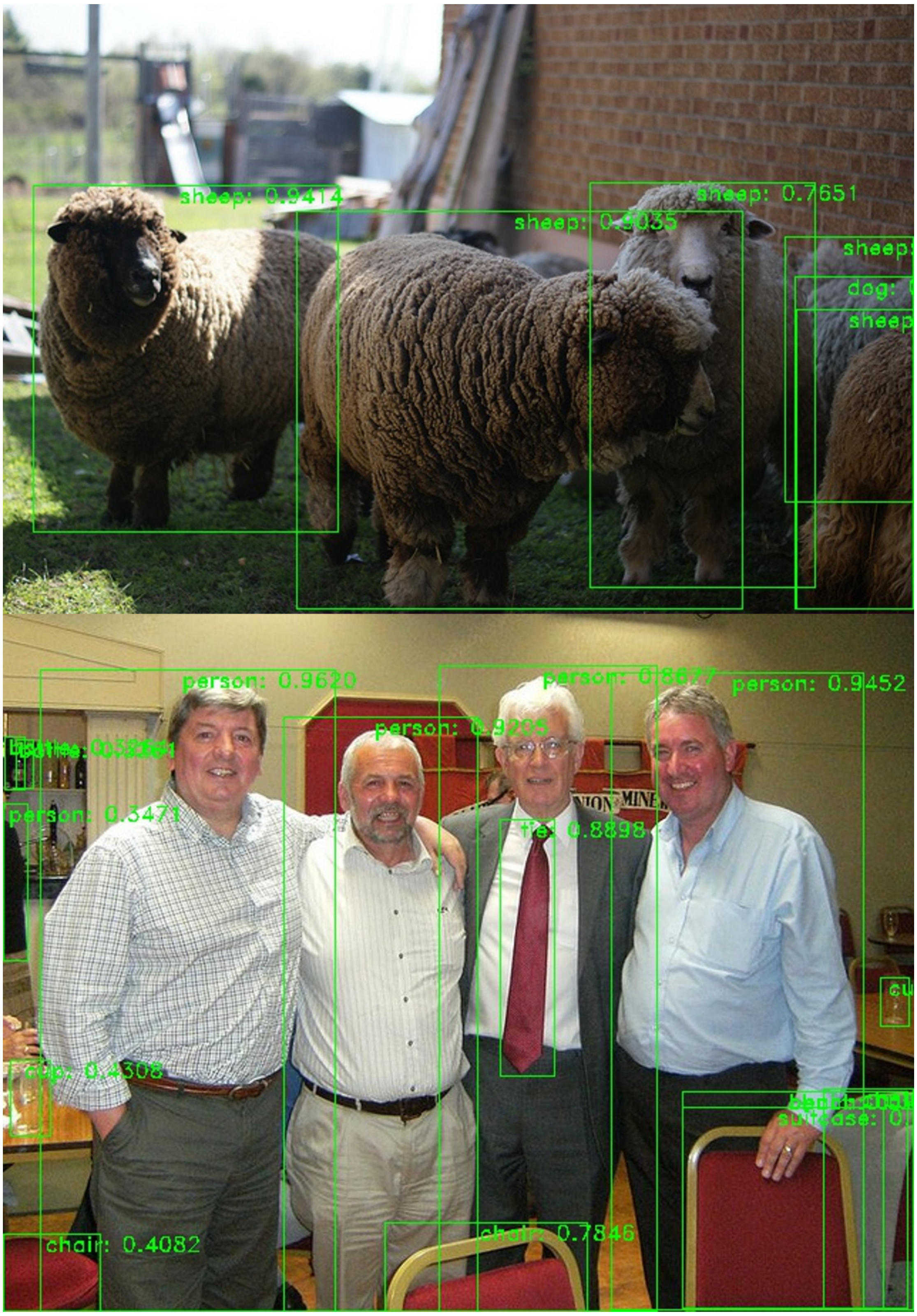

Figure 4.2 Representative results of our light-weight detector (2)

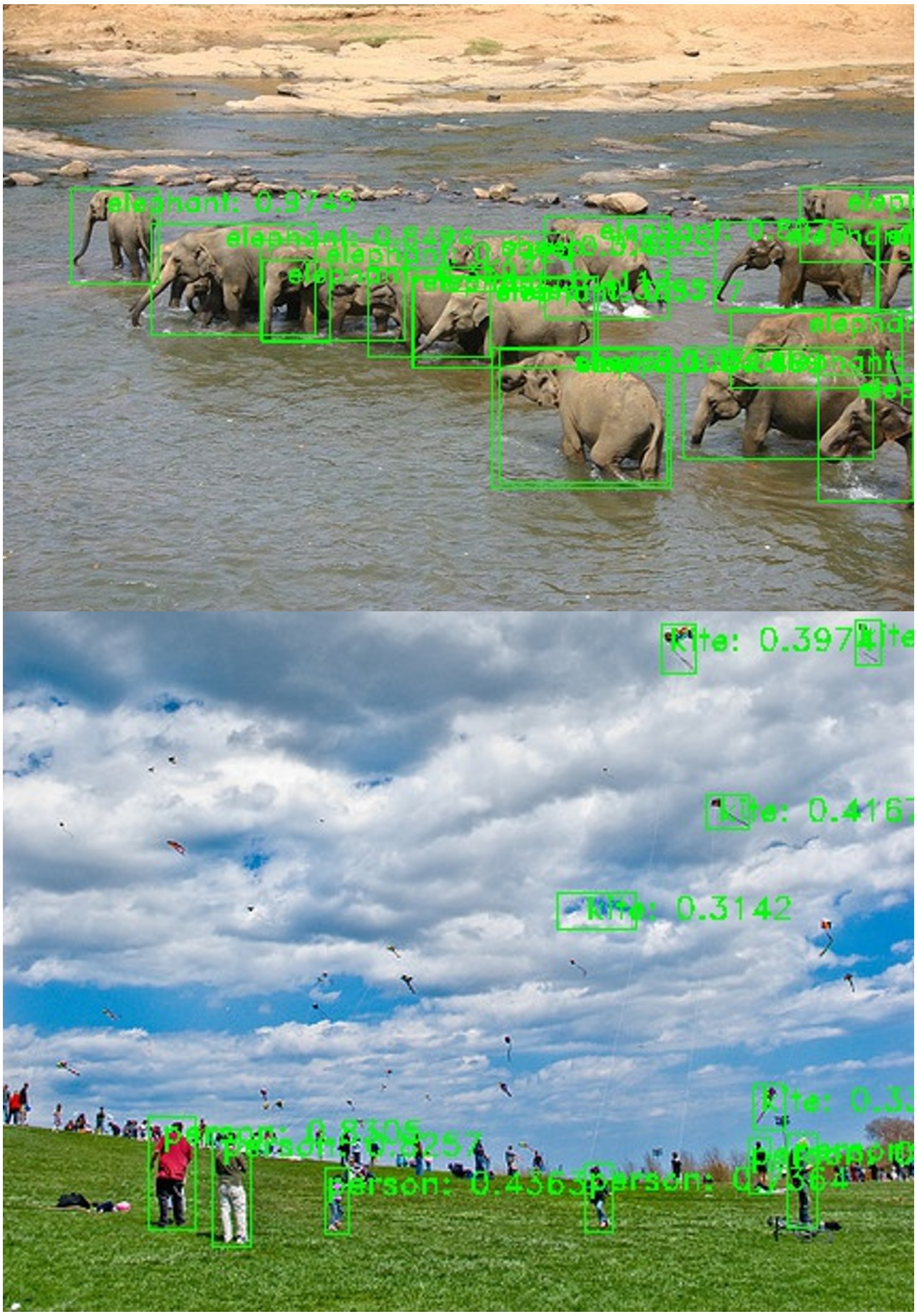

Figure 4.3 Representative results of our light-weight detector (3)

Chapter 5 Conclusion

In this thesis, we first combine ShuffleNet v2 [1](an inference-efficient backbone) and Light-Head R-CNN [2](a fast and light-weight detection framework) to propose a light-weight CNN architecture design for faster and better object detector. We also propose a novel multitask learning framework, distributed multitask learning, to train this detection architecture, including joint training with classification and multiscale training.

Our ultimate ShuffleNet v2 [1] + Light-Head R-CNN [2] architecture, trained under our distributed multitask learning framework, outperforms state-of-art object detectors on COCO while keeping time efficiency. It achieves 32.8 mmAP at 109 FPS on COCO, significantly outperforming the single-stage, fast detectors like YOLO and SSD on both speed and accuracy.

References

[1] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Light-weight cnn architecture design for fast inference," *European Conference on Computer Vision 2018 Underview*, 2018.

[2] 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.

[3] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, pp. 91–110, 2004.

[10] 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.

[11] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *European Conference on Computer Vision.* Springer, 2014, pp. 391–405.

[12] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester, "Discriminatively trained deformable part models, release 5," 2012.

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

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," pp. 1–9, 2015.

[15] 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.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[17] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2017, pp. 5987–5995.

[18] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *arXiv preprint arXiv:1709.01507*, 2017.

[19] 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 less than 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[20] 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.

[21] 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.

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

[23] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” 2017.

[24] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, “Deformable convolutional networks,” *arXiv preprint arXiv:1703.06211*, 2017.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[26] 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.

[27] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” *arXiv preprint arXiv:1610.02357*, 2016.

[28] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 1, no. 2, 2017, p. 3.

[29] F. Yu, V. Koltun, and T. Funkhouser, “Dilated residual networks,” in *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[30] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, P. Prabhat, and R. P. Adams, “Scalable bayesian optimization using deep neural networks,” in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML’15. JMLR.org, 2015, pp. 2171–2180.

[31] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in *International Conference on Computer Vision (ICCV)*, vol. 2, 2017, p. 6.

[32] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” *arXiv preprint arXiv:1707.07012*, 2017.

[33] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik, “Multiscale combinatorial grouping,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 328–335.

[34] 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.

[35] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *arXiv preprint arXiv:1708.02002*, 2017.

[36] 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.

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

[38] 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.

[39] L. Tychsen-Smith and L. Petersson, "Denet: Scalable real-time object detection with directed sparse sampling," *arXiv preprint arXiv:1703.10295*, 2017.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *European Conference on Computer Vision*. Springer, 2014, pp. 346–361.

[41] R. Collobert and J. Weston, "A unified architecture for natural language processing." in *Proceedings of the 25th International Conference on Machine Learning*, vol. 20, no. 1, 2008, pp. 160–167.

[42] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), May 2013*, May 2013.

[43] R. Bharath, K. Steven, R. Patrick, W. Dale, K. David, and P. Vijay, "Massively multitask networks for drug discovery," 2015.

[44] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997.

[45] J. Baxter, "A bayesian/information theoretic model of learning to learn viamultiple task sampling," *Machine Learning*, vol. 28, no. 1, pp. 7–39, 1997.

[46] L. Duong, T. Cohn, S. Bird, and P. Cook, "Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser," in *Proceedings of the*

53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), 2015, pp. 845–850.

[47] Y. Yang and T. Hospedales, “Trace norm regularised deep multi-task learning,” 06 2016.

[48] M. Long, Z. CAO, J. Wang, and P. S. Yu, “Learning multiple tasks with multilinear relationship networks,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1593–1602. [Online]. Available: http://papers.nips.cc/paper/6757-learning-multiple-tasks-with-multilinear-relationship-networks.pdf

[49] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. S. Feris, “Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1131–1140, 2017.

[50] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, “Cross-stitch networks for multi-task learning,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3994–4003, 2016.

[51] K. He and J. Sun, “Convolutional neural networks at constrained time cost,” in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 5353–5360.

[52] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[53] T. Zhang, G.-J. Qi, B. Xiao, and J. Wang, “Interleaved group convolutions for deep neural networks,” in *International Conference on Computer Vision*, 2017.

[54] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[55] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi, “Deep roots: Improving cnn efficiency with hierarchical filter groups,” *arXiv preprint arXiv:1605.06489*, 2016.

[56] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense convolutional networks for efficient prediction," 2017.

[57] 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.

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

[59] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 1943–1955, 2016.

[60] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1984–1992.

[61] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[62] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[63] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2270–2278.

[64] 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.

[65] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2755–2763.

[66] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," *arXiv preprint arXiv:1711.09224*, 2017.

[67] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[68] 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.

[69] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, and P. Dubey, "Distributed deep learning using synchronous stochastic gradient descent," *arXiv preprint arXiv:1602.06709*, 2016.

[70] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." pp. 4278–4284, 2017.

[71] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *arXiv preprint arXiv:1802.01548*, 2018.

[72] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," *arXiv preprint arXiv:1712.00559*, 2017.

[73] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun, "Object detection networks on convolutional feature maps," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 7, pp. 1476–1481, 2017.

[74] 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.

[75] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145 – 151, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608098001166

[76] 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.

[77] 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.

[78] T.-Y. Lin and P. Dollár, “Ms coco api. https://github.com/pdollar/coco,” 2016.

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

Acknowledgements

I would like to thank my advisor Dr. Xiangyu Zhang for his insightful advices and patient guidances throughout this project and the one year. I am truly impressed by his meticulosity and diligence on researches. It is his outstanding expertise and professionalism that lead me to the academic road and help me to overcome many difficulties. I must also thank Dr. Cewu Lu for being my campus advisor and his full support and aid behind my back.

I am grateful for my seniors and peers in the Megvii(Face++) internship experience. I learned so much knowledge and skills essential for my project and thesis from you. I would like to thank Mr. Zeming Li, Mr. Chao Peng, Mr. Ningning Ma, Mr. Tong Yang who pioneered the topic of basemodel design and object detection. I will treasure my experiences in Megvii(Face++) forever.

I would express my sincere thanks to the School of Electronic Information and Electrical Engineering, and Shanghai Jiao Tong University, for providing me with academic opportunities, abundant resources and necessary facilities.

I take this opportunity to express gratitude to the all my instructors in Shanghai Jiao Tong University, especially the faculties in the Department of Computer Science and Engineering. I could never finish my study and this thesis without their well-designed courses, assignments and projects.

Accomplishing the thesis requires more than academic support. I appreciate the companions of my roommates, classmates in my major, and all of my friends in Shanghai Jiao Tong University who care and encourage me over the past four years. Your intelligence and virtues impress and influence me intensely.

I would like to show my utmost gratitude to my parents and family. They always motivate me by giving their consistent love and understanding throughout my life.