

4-5-2018 2:30 PM

Pelee: A Real-Time Object Detection System on Mobile Devices

Jun Wang, *The University of Western Ontario*

Supervisor: Ling, Charles X., *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in
Computer Science

© Jun Wang 2018

Follow this and additional works at: <https://ir.lib.uwo.ca/etd>



Part of the [Artificial Intelligence and Robotics Commons](#)

Recommended Citation

Wang, Jun, "Pelee: A Real-Time Object Detection System on Mobile Devices" (2018). *Electronic Thesis and Dissertation Repository*. 5278.

<https://ir.lib.uwo.ca/etd/5278>

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlsadmin@uwo.ca.

Abstract

There has been a rising interest in running high-quality Convolutional Neural Network (CNN) models under strict constraints on memory and computational budget. A number of efficient architectures have been proposed in recent years, for example, MobileNet, ShuffleNet, and NASNet-A. However, all these architectures are heavily dependent on depthwise separable convolution which lacks efficient implementation in most deep learning frameworks. Meanwhile, there are few studies that combine efficient models with fast object detection algorithms. This research tries to explore the design of an efficient CNN architecture for both image classification tasks and object detection tasks. We propose an efficient architecture named PeleeNet, which is built with conventional convolution instead. On ImageNet ILSVRC 2012 dataset, our proposed PeleeNet achieves a higher accuracy by 0.6% and 11% lower computational cost than MobileNet, the state-of-the-art efficient architecture. It is also important to point out that PeleeNet is of only 66% of the size of MobileNet and 1/49 size of VGG.

We then propose a real-time object detection system on mobile devices. We combine PeleeNet with Single Shot MultiBox Detector (SSD) method and optimize the architecture for fast speed. Meanwhile, we port SSD to iOS and provide an optimized code implementation. Our proposed detection system, named Pelee, achieves 70.9% mAP on PASCAL VOC2007 dataset at the speed of 17 FPS on iPhone 6s and 23.6 FPS on iPhone 8. Compared to TinyYOLOv2, the most widely used computational efficient object detection system, our proposed Pelee is more accurate (70.9% vs. 57.1%), 2.88 times lower in computational cost and 2.92 times smaller in model size.

Keywords: Real-time Object Detection, Convolutional Neural Network, Efficient Architecture, Mobile Device, Embedded Vision

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Charles X. Ling for the continuous support and guidance of my study and research. I would not be able to complete this thesis without his instructions and help. Thanks are also given to my colleagues for their collaboration and valuable discussions. Especially my colleagues, Xiang Li and Shuang Ao, gave me so many insightful suggestions. Last but not least, my acknowledgement and love go to my family: my parents Youku Wang and Fengqin Cao, my wife Li Sun and my two lovely kids Alice and Daniel. Without their encouragement, continuous love and support, I would not be able to pursue my study in Western University and finish this thesis.

My research is supported by Mitacs grant and scholarship from the School of Graduate Studies in Western University. Special acknowledgement is also given to the open source community. Without the excellent open source software and projects, I would not be able to complete this thesis.

Contents

Abstract	i
Acknowledgements	i
List of Figures	vii
List of Tables	ix
List of Appendices	x
1 Introduction	1
1.1 Motivation	1
1.2 Contributions	2
1.3 Thesis Outline	4
2 Background	5
2.1 Image Classification vs. Object Detection	5
2.2 Training vs. Inference	5
2.3 Convolutional Neural Network	6
2.3.1 Layers in CNN	6
2.3.2 Network Architecture	8
VGGNet	8
GoogLeNet	9
ResNet	9
DenseNet	9
2.4 Datasets	10

2.4.1	ImageNet ILSVRC 2012	10
2.4.2	CIFAR-10	11
2.4.3	Stanford Dogs	11
2.4.4	PASCAL VOC	12
2.5	Related Work	13
2.5.1	Network Acceleration and Compression	13
2.5.2	Object Detection	16
3	PeleeNet: An Efficient Feature Extraction Network	18
3.1	Introduction	18
3.2	Methodology	19
3.2.1	Review of DenseNet	19
3.2.2	Two-Way Dense Layer	20
3.2.3	Stem Block	20
3.2.4	Dynamic number of Channels in Bottleneck Layer	21
3.2.5	Transition Layer Without Compression	21
3.2.6	Composite Function	22
3.2.7	Overview of Architecture	23
3.3	Experiments	23
3.3.1	Dataset	23
3.3.2	Evaluation	24
3.3.3	Training Parameters	25
3.3.4	Impact of Different Elements	25
3.3.5	Effects of Different Feature Enhancement Methods	27
3.3.6	Results on Stanford Dogs	28
3.3.7	Results on ILSVRC 2012	29
3.4	Summary	29
4	Pelee: A Real-Time Object Detection System	33
4.1	Introduction	33
4.2	Methodology	34

4.2.1	Review of SSD	34
	Multi-scale Feature Maps for Detection	34
	Default Boxes and Aspect Ratios	34
	Training Objective	35
	Hard Negative Mining	36
4.2.2	Feature Map Selection	36
4.2.3	Residual Prediction Block	37
4.2.4	Small Convolutional Kernel for Prediction	38
4.3	Experiments	38
4.3.1	Dataset	38
4.3.2	Evaluation	38
	Bounding Box Evaluation	38
	Mean Average Precision (mAP)	39
4.3.3	Data Augmentation	39
4.3.4	Training Parameters	40
4.3.5	Results on PASCAL VOC 2007	40
4.4	Summary	41
5	Benchmark on Real Devices	47
5.1	Introduction	47
5.2	Benchmark on Mobile Phone	47
5.2.1	Benchmark of Efficient Classification Model on ILSVRC 2012	48
5.2.2	Benchmark of Efficient One-stage Detector on VOC 2007	48
5.3	Benchmark on CPU and GPU	49
5.4	Summary	50
6	Conclusion and Future Work	53
6.1	Conclusion	53
6.2	Future Work	54
	Bibliography	55

A	Architecture of DenseNet-41	59
B	Merging Batch Normalization Layer with Convolution Layer	61
C	More Detection Examples: TinyYOLOv2 vs. Pelee	62
	Curriculum Vitae	66

List of Figures

1.1	Scenarios of on-device artificial intelligence	2
2.1	Image classification vs. object detection	6
2.2	Training vs. inference	7
2.3	Convolution layer	8
2.4	Pooling layer	9
2.5	Algorithm of Batch Normalization	10
2.6	Inception model with dimension reduction	11
2.7	High level diagram of ResNet architecture	12
2.8	High level diagram of DenseNet architecture	13
2.9	Deep compression - three stage compression pipeline	14
2.10	Convolution with XNOR-bitcount	14
2.11	Standard convolution filters vs. depthwise convolution filters	15
2.12	High level diagram of Faster R-CNN architecture	16
2.13	High level diagram of one-stage detector architecture	17
3.1	A deep DenseNet with three dense blocks	19
3.2	A dense block with 5 layers and growth rate 4	20
3.3	Two-way dense layer	21
3.4	Structure of stem block	22
3.5	Cosine learning rate annealing vs. step learning rate decay	29
3.6	Comparison of different models on Stanford Dogs	31
4.1	Architecture of SSD	35
4.2	Default boxes and aspect ratios in SDD	42

4.3	Default boxes in SSD vs. anchor boxes in Faster R-CNN	42
4.4	Residual Prediction Block	43
4.5	Data augmentation in SSD	44
4.6	Average Precision on VOC 2007	45
4.7	Detection examples: TinyYOLOv2 vs. Pelee	46
5.1	Workflow of CoreML	52

List of Tables

3.1	Computational cost of bottleneck layer	22
3.2	Overview of PeleeNet architecture	24
3.3	Experimental configuration for PeleeNet	26
3.4	Impact of different elements in DenseNet-41	27
3.5	Effects of different feature enhancement methods	28
3.6	Results on Stanford Dogs	30
3.7	Effects of various design choices and components on performance	31
3.8	Results on ILSVRC 2012	32
4.1	Parameters of feature maps	37
4.2	Training parameters of Pelee	40
4.3	Effects of various design choices on performance	41
4.4	Results on PASCAL VOC 2007	41
5.1	Benchmark of efficient classification models on ILSVRC 2012	49
5.2	Benchmark of efficient one-stage detector on VOC 2007	49
5.3	Benchmark of one-stage detector on VOC 2007	51
A.1	Architecture of DenseNet-41	60

List of Appendices

Appendix A Architecture of DenseNet-41	59
Appendix B Merging BatchNorm Layer with Conv Layer	61
Appendix C More Detection Examples: TinyYOLOv2 vs. Pelee	62

Chapter 1

Introduction

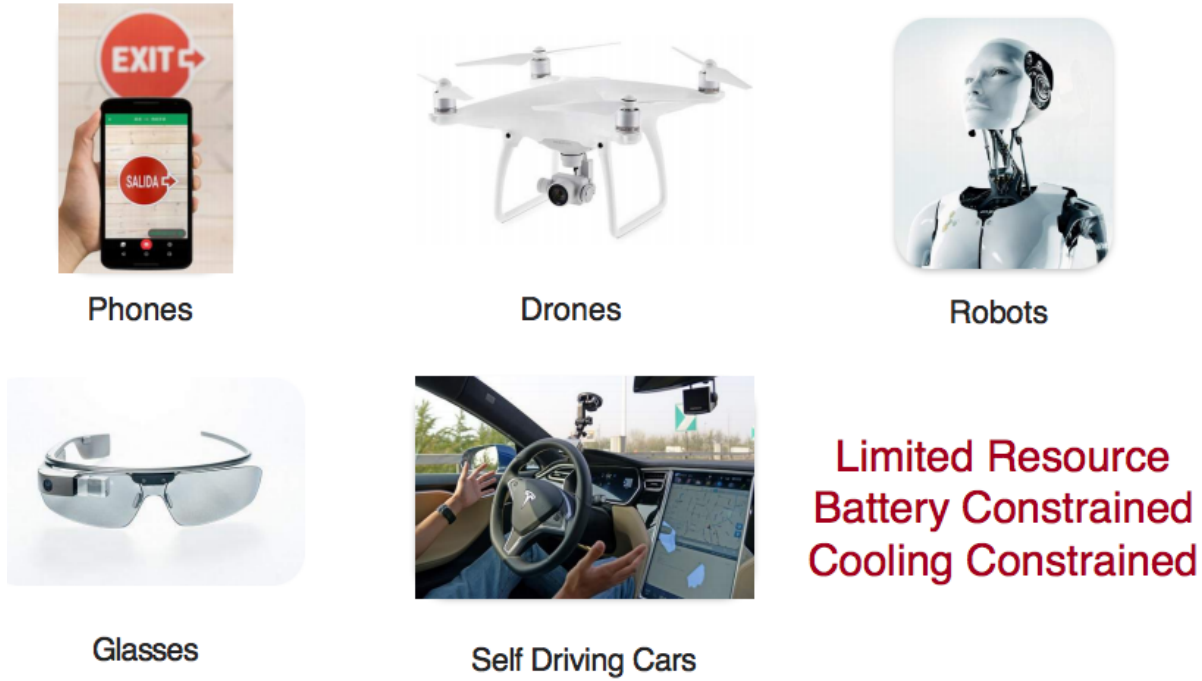
This chapter briefly introduces the overall content of this thesis. The first section describes the source of motivation for this research. The second section is a short description of the major contributions of the research. The thesis outline is introduced in the end.

1.1 Motivation

Convolutional Neural Networks have evolved to be the state-of-the-art technique in computer vision. However, these algorithms are computationally intensive, which makes it difficult to deploy on embedded devices with limited hardware resources. Meanwhile, on-device Artificial Intelligence (AI) is highly demanded in many real-world applications, e.g. robotics, self-driving car, Advanced Driver Assistance System (ADAS) and augmented reality (AR), etc.

The increasing needs of running high-quality CNN models under strict constraints on memory and computational budget encourage the study on model acceleration and compression. Many innovative methods [13][40][41][16][29][8] have been proposed in recent years, which promotes on-device AI greatly. However, most of these papers mainly focus on classification tasks. There are few papers that combine model acceleration methods with fast object detection algorithms [15].

This study tries to explore the design of the efficient CNN architecture for both image classification tasks and object detection tasks. The reason for choosing object detection is that, from the practical perspective, object detection is a basic application scenario in the real world, considering that multiple objects being in a picture is common. From the academic perspective, object detection task means more technical challenges than classification task, which not only



From Song presents Deep Learning Tutorial and Recent Trends at FPGA17, Monterey

Figure 1.1: Scenarios of on-device artificial intelligence

needs to judge the category of objects, but also gives the specific location of each object. It covers both the classification problem and the regression problem. The research results are able to provide strong evidences for other machine learning tasks.

1.2 Contributions

In summary, our main contributions are listed as follows:

We propose a variant of DenseNet [14] architecture called PeleeNet for mobile devices. PeleeNet follows the innovate connectivity pattern and some of key design principals of DenseNet. It is also designed to meet strict constraints on memory and computational budget. Experimental results on Stanford Dogs [19] dataset show that our proposed PeleeNet is higher in accuracy than the one built with the original DenseNet architecture by 5.05% and higher than MobileNet [13] by 6.53%. PeleeNet achieves a compelling result on ImageNet ILSVRC 2012 [2] as well. The top-1 accuracy of PeleeNet is 71.3% which is higher than that of MobileNet by 0.6%. It is also important to point out that PeleeNet is only 66% of the size of MobileNet. Some of the

key features of PeleeNet are:

- **Two-Way Dense Layer** We use a 2-way dense layer to get different scales of receptive fields. One way of the layer uses a small kernel size (3x3), which is good enough to capture small-size objects. The other way of the layer uses a larger 5x5 kernel to learn visual patterns for large objects. To reduce computational cost, we follow the common practice of replacing 5x5 convolution with two stacked 3x3 convolution. (Fig. 3.3)
- **Stem Block** We design an cost efficient stem block before the first dense layer. The structure of stem block is shown on Fig. 3.4. This stem block can effectively improve the feature expression ability without adding computational cost too much - better than other more expensive methods, e.g., increasing channels of the first convolution layer or increasing growth rate.
- **Dynamic Number of Channels in Bottleneck Layer** Another highlight is that the number of channels in the bottleneck layer varies according to the input shape to make sure the number of output channels does not exceed the number of its input channels. Compared to the original DenseNet structure, our experiments show that this method can save up to 28.5% of the computational cost with a small impact on accuracy.
- **Transition Layer without Compression** Our experiments show that the compression factor proposed by DenseNet hurts the feature expression. We always keep the number of output channels the same as the number of input channels in transition layers.
- **Composite Function** We use the conventional wisdom of post-activation (Convolution - Batch Normalization [17] - Relu) as our composite function instead of pre-activation used in DenseNet. For post-activation, all batch normalization layers can be merged with convolution layer at the inference stage, which can accelerate the speed greatly.

We optimize the network architecture of Single Shot MultiBox Detector (SSD) [26] for speed acceleration and then combine it with PeleeNet. Our proposed system, named Pelee, achieves 70.9% mAP on PASCAL VOC [3] 2007 object detection dataset. It also outperforms Tiny-YOLOv2 [31], the most widely used computational efficient object detection system, in terms of a higher accuracy by 13.8%, 2.88 times lower in computational cost and 2.92 times smaller in model size. The major enhancements proposed to balance speed and accuracy are:

- **Feature Map Selection** We build object detection network in a way different from the original SSD with a carefully selected set of 5 scale feature maps (19 x 19, 10 x 10, 5 x 5, 3 x 3, and 1 x 1). To reduce computational cost, we do not use 38 x 38 feature map.
- **Residual Prediction Block** We follow the design ideas proposed by [24] that encourage features to be passed along the feature extraction network. For each feature map used for detection, we build a residual [9] block before conducting prediction.

- **Small Convolutional Kernel for Prediction** Residual prediction block makes it possible for us to apply 1x1 convolutional kernels to predict category scores and box offsets. Our experiments show that the accuracy of the model using 1x1 kernels is almost the same as that of the model using 3x3 kernels. However, 1x1 kernels reduce the computational cost by 21.5%.

We provide an efficient implementation of SSD algorithm on iOS. We have successfully ported SSD to iOS and provided an optimized code implementation. Our proposed system runs at the speed of 17.1 FPS on iPhone 6s and 23.6 FPS on iPhone 8. The speed on iPhone 6s, a phone released in 2015, is 2.6 times faster than that of the official SSD implementation on a server with a powerful Intel i7-6700K@4.00GHz CPU.

We provide a benchmark test for different efficient classification models and different one-stage object detection methods on NVIDIA GPU, Intel CPU and iPhone.

1.3 Thesis Outline

Chapter 2 describes the background information related to this thesis, including a number of key terminologies, datasets we used and related work of network acceleration and object detection. Chapter 3 introduces PeleeNet, our proposed feature extraction network. Chapter 4 describes our proposed object detection system, named Pelee. In this chapter, we discuss the architecture of SSD framework and our improvement on detection speed. In chapter 5, a benchmark of efficient classification models and one-stage object detectors is given. Chapter 6 draws a conclusion to the whole project and discusses the possible future works.

Chapter 2

Background

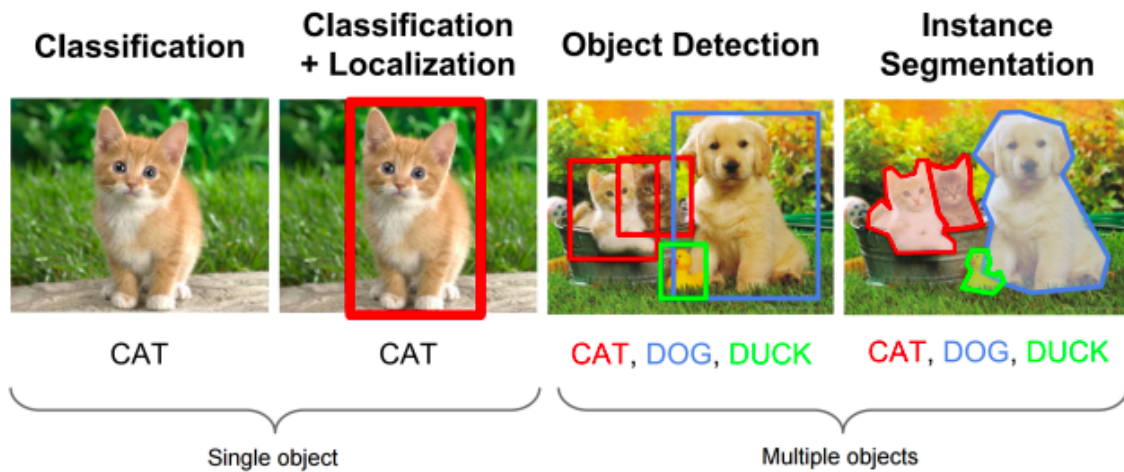
This chapter describes the background information related to this study. Section 2.1 and section 2.2 clarifies some key terminologies. Section 2.3 introduces the major layers of Convolutional Neural Network (CNN) and some popular network architectures. Section 2.4 briefly introduces datasets used in this study. The last section introduces some previous work for network acceleration and object detection.

2.1 Image Classification vs. Object Detection

Image recognition/classification refers to predicting the label of an image among predefined labels. It assumes that there is single object of interest in the image and it covers a significant portion of image. Detection is about not only finding the class of object but also localizing the extent of an object in the image. The object can be lying anywhere in the image and can be of any size. (Fig. 2.1)

2.2 Training vs. Inference

Deep Learning is a most popular approach for learning representations. It has been widely used for all kinds of computer vision tasks and achieved state-of-the-art performance. On a high level, working with deep learning is a two-phase process: training and inference. Training is the phase in which a neural network tries to learn from the data. Inference is the phase in which a trained network is deployed to the product environment and is used to infer/predict the



From http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

Figure 2.1: Image classification vs. object detection

new data.

There are different performance goals for training and inference. Usually people can tolerate a long training time to get a powerful model, but it is hoped that inference time should be as short as possible. Training can be sped up by using multiple GPUs in parallel with large batch size. However, inference typically batches a smaller number of inputs than training to meet the low latency requirement of AI-powered services. (Fig. 2.2)

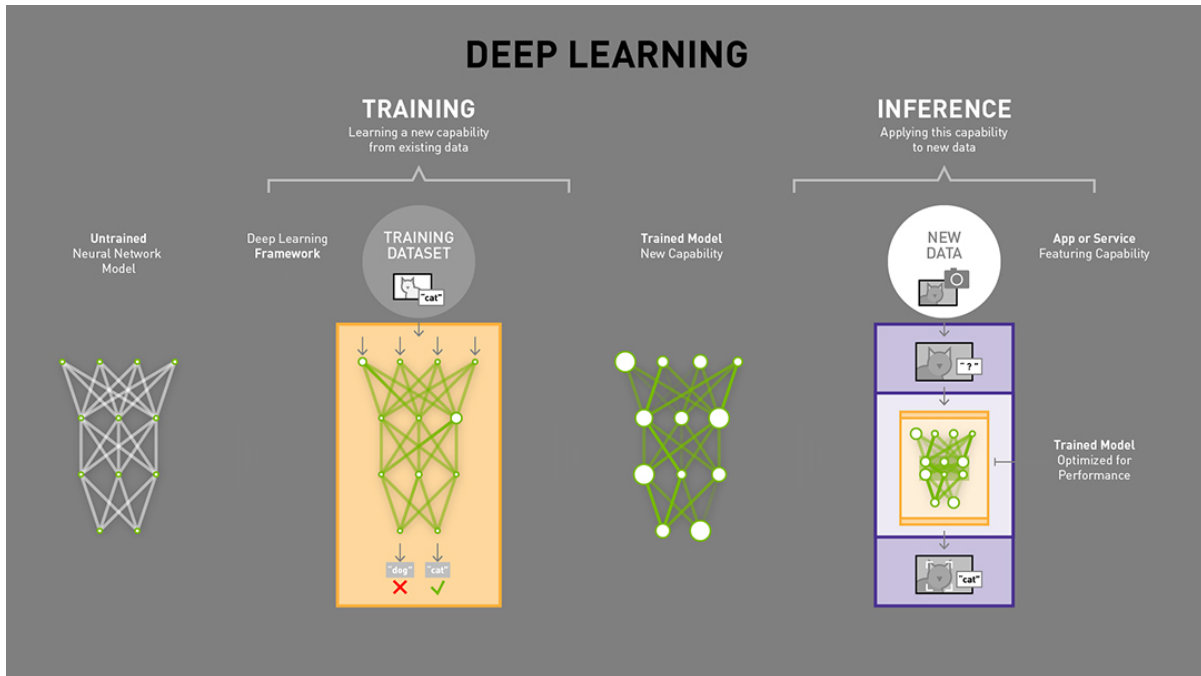
2.3 Convolutional Neural Network

2.3.1 Layers in CNN

Convolutional Neural Networks (CNN) is one of the most popular deep learning methods. A CNN consists of different types of layers. Each layer has a simple work-flow: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters. This section introduces some major layers used in our study.

Convolutional Layer

Convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full



From <https://blogs.nvidia.com/blog/2016/08/22/difference-deep-learning-training-inference-ai/>

Figure 2.2: Training vs. inference

depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. (Fig. 2.3)

Relu Layer

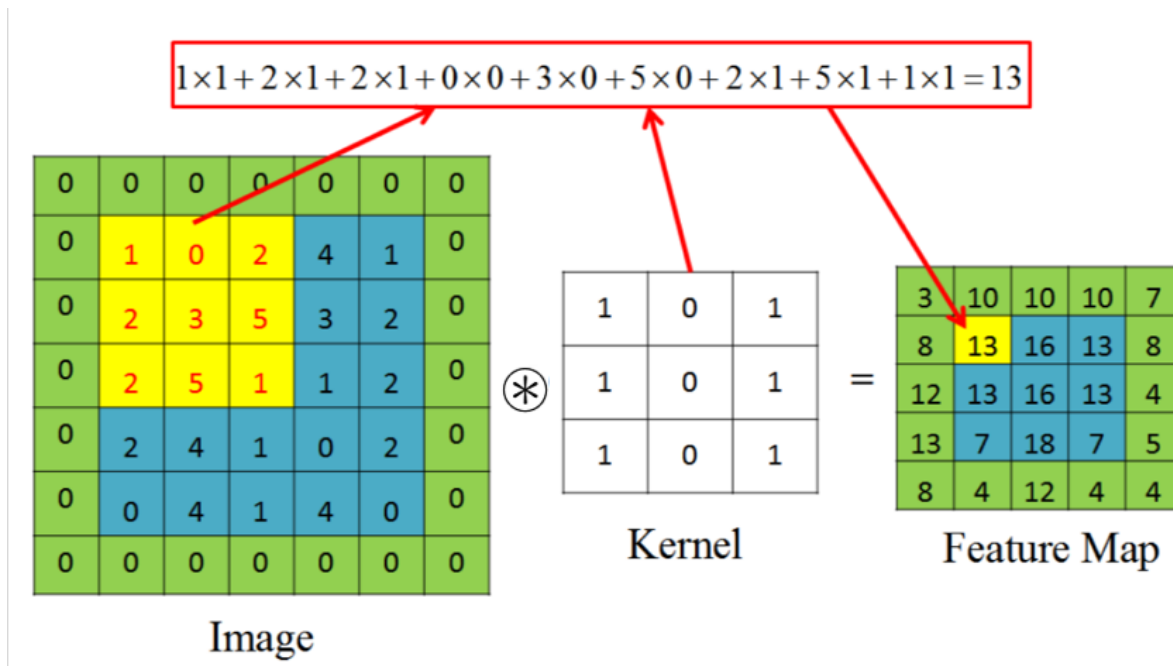
ReLU is the abbreviation of Rectified Linear Units. This layer applies the non-saturating activation function $f(x) = \max(0, x)$. It perfects the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Pooling Layer

Another important layer of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling and average pooling are the two most common. Max pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. (Fig. 2.4)

Batch Normalization Layer

Batch Normalization, proposed by Sergey Ioffe and Christian Szegedy[17], is a method to reduce internal covariate shift in neural networks. Batch Normalization allows us to use much higher learning rates and to focus less on weights initialization. (Fig. 2.5)



Convolution operation with 3 x 3 kernel, stride 1 and padding 1. \otimes denotes the convolutional operator

Figure 2.3: Convolution layer

2.3.2 Network Architecture

The first CNN that had a great success on image classification is the LeNet proposed by Y.LeCun in 1989 [23]. Its successor AlexNet [22] won ILSVRC challenge in 2012 and significantly outperformed the runner-up (top 5 error of 16% compared to runner-up with 26% error). After that many excellent network architectures have been proposed. This section lists some famous network architectures that are related to this research or have inspired our design.

VGGNet

VGGNet, proposed by Karen Simonyan and Andrew Zisserman [35], is the runner-up in ILSVRC 2014. It is the first architecture that uses much smaller 3x3 filters in each convolutional layer and also combines them as a sequence of convolutions to emulate greater receptive fields.

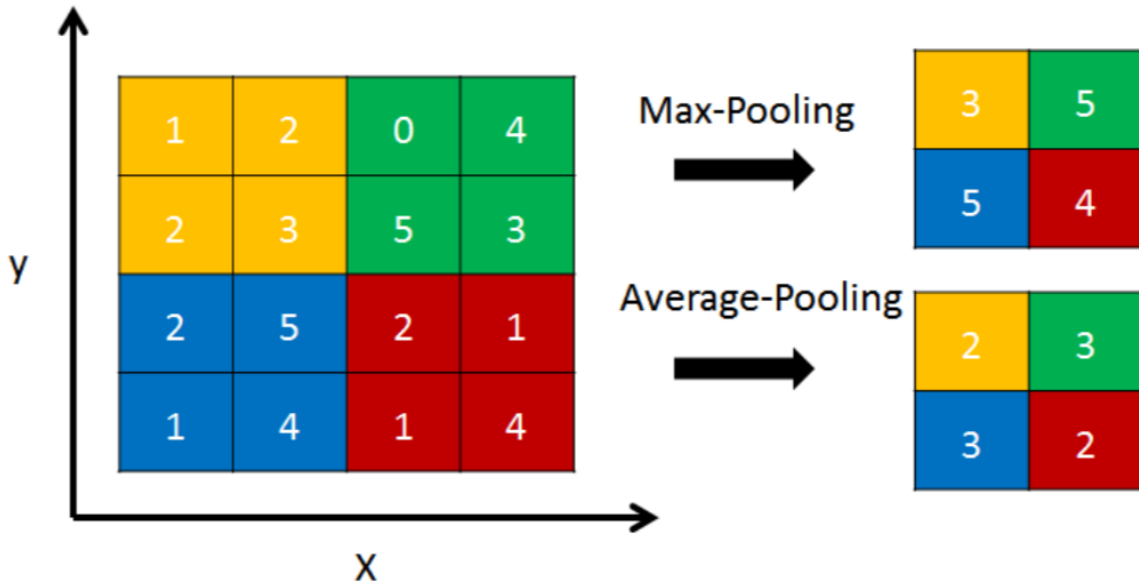


Figure 2.4: Pooling layer

GoogLeNet

GoogLeNet, proposed by Szegedy et al. [37] from Google, is the ILSVRC 2014 winner. Its main contribution is the development of an Inception Module that dramatically reduces the number of parameters in the network. (Fig. 2.6)

ResNet

Residual Network (ResNet), proposed by Kaiming He et al. [9], is the winner of ILSVRC 2015 and COCO 2015. It features special skip connections to make training a very deep network easier. ResNet is currently by far the state of the art Convolutional Neural Network model and is widely used in all kinds of computer vision tasks. (Fig. 2.7)

DenseNet

Densely Connected Convolutional Network (DenseNet), proposed by Gao Huang and Zhuang liu et al.[14], is a network architecture where each layer is directly connected to every other layer in a feed-forward fashion (within each dense block). DenseNet encourages feature reuse and substantially reduces the number of parameters (Fig. 2.8). DenseNet is the foundation of our study. We will give a detailed description of DenseNet in Section 3.2.1

Input : Values of x over a mini-batch $\{x_{1..m}\}$

Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}
 \mu &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && //mini - batch\ mean \\
 \sigma^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2 && //mini - batch\ variance \\
 \hat{x}_i &\leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} && //normalize \\
 y_i &\leftarrow \gamma \hat{x}_i + \beta && //scale\ and\ shift
 \end{aligned}$$

Figure 2.5: Algorithm of Batch Normalization

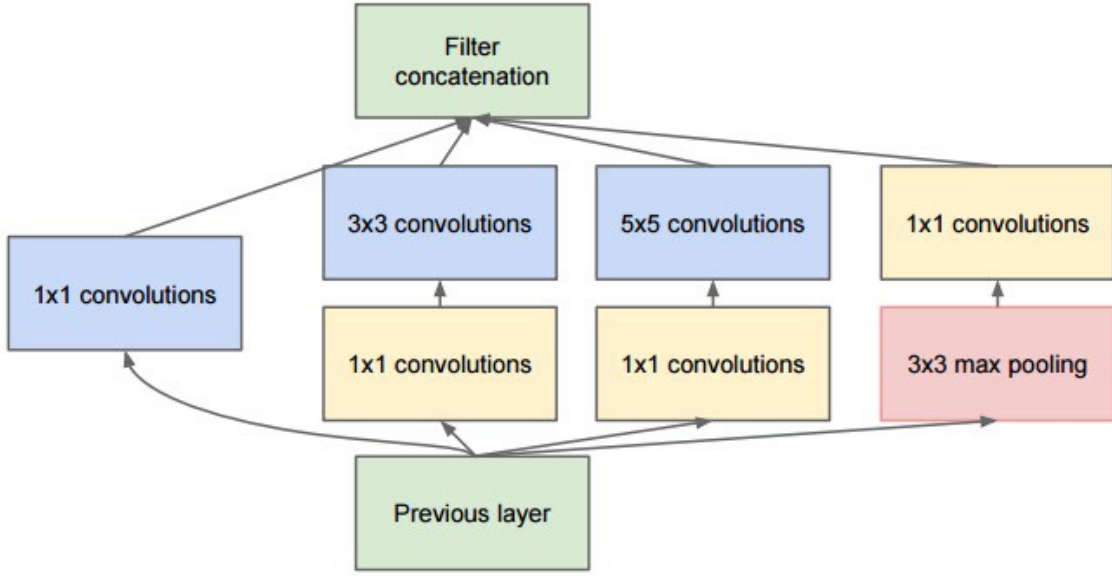
2.4 Datasets

Here we introduce datasets that are used in different tasks. ImageNet [2] ILSVRC 2012 and a customized Stanford Dogs [19] are used for image classification tasks. PASCAL VOC [3] 2007 and 2012 are used for object detection tasks.

2.4.1 ImageNet ILSVRC 2012

ImageNet [2] is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet is called a "synonym set" or "synset". ImageNet uses "WordNet ID" (wnid) to uniquely identify a synset. For example, the wnid of synset "dog, domestic dog, Canis familiaris" is "n02084071".

ImageNet ILSVRC 2012 is a classification dataset widely used to measure the performance of a network architecture. The training dataset contains 1000 categories and 1.2 million images. The validation dataset contains 50,000 images with 50 images per class.



From Going deeper with convolutions [37]

Figure 2.6: Inception model with dimension reduction

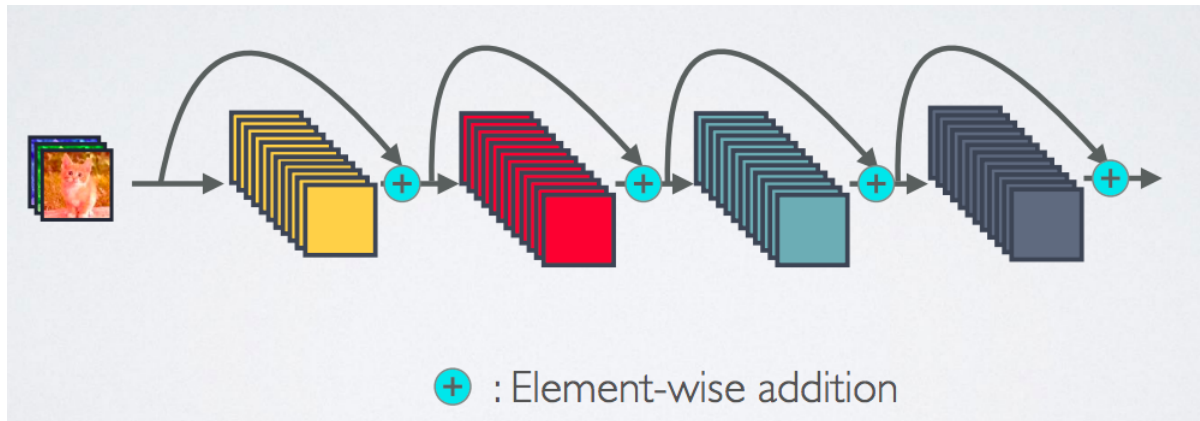
2.4.2 CIFAR-10

CIFAR-10 [21] is a popular toy image classification dataset. This dataset consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. Although CIFAR-10 is widely used for ablation study, we do not use this dataset in this thesis. The main reason is that we have limited computing resource and hope that the hyper parameters got from ablation study can be used for ImageNet ILSVRC task. However, the hyperparameters used on CIFAR-10 are not able to be used on ILSVRC because CIFAR-10 is a low resolution dataset while ILSVRC is a high resolution dataset. Moreover, the network architectures on these two datasets are also different. Therefore, we build a customized Stanford Dogs dataset described as follows.

2.4.3 Stanford Dogs

Stanford Dogs [19] dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image classification.

Fine-grained classification, as a sub-field of image recognition, aims to distinguish subordinate



From <http://www.cs.cornell.edu/gaohuang/papers/DenseNet-CVPR-Slides.pdf>

Figure 2.7: High level diagram of ResNet architecture

categories within entry level categories. We believe the dataset used for this kind of task is complicated enough to evaluate the performance of the network architecture. However, there are only 14,580 training images, with about 120 images per class, in the original Stanford Dogs dataset, which is not large enough to train the model from scratch.

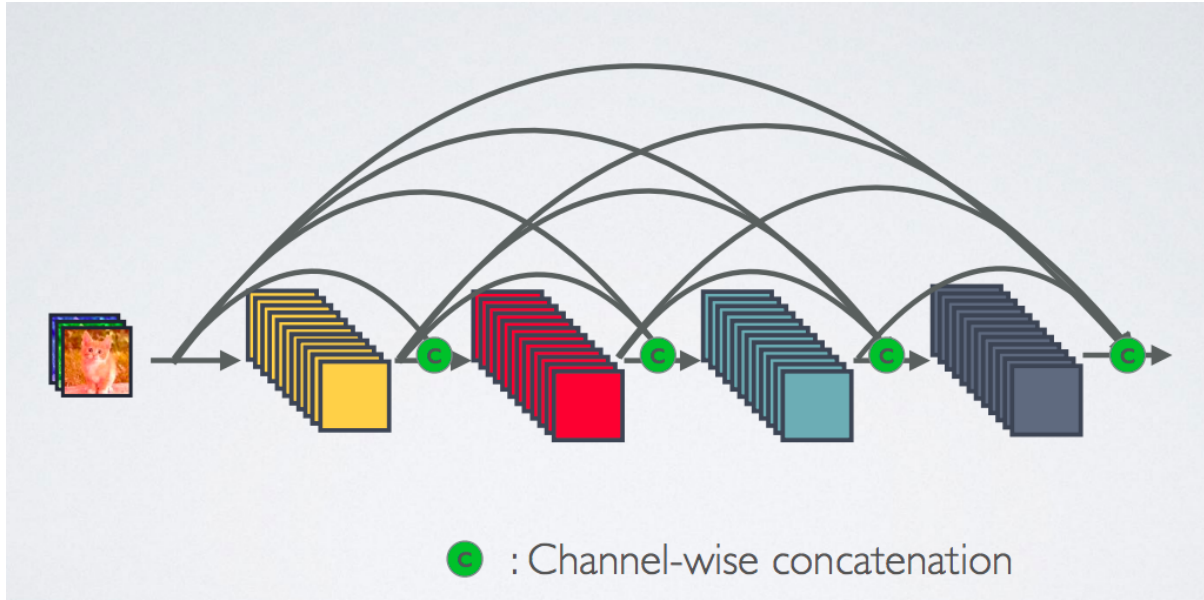
Instead of using the original Stanford Dogs, we build a subset of ILSVRC 2012 according to the ImageNet wnid used in Stanford Dogs. Both training data and validation data are exactly copied from the ILSVRC 2012 dataset. In the following chapters, the term of Stanford Dogs means this subset of ILSVRC 2012 instead of the original one. Contents of this dataset:

- Number of categories: 120
- Number of training images: 150,466
- Number of validation images: 6,000

2.4.4 PASCAL VOC

PASCAL VOC [3] Challenge is a challenge in visual object recognition funded by PASCAL network of excellence. The datasets from the challenges are widely used for the benchmark of detection and segmentation tasks. The twenty object classes that have been selected are:

- **Person:** person
- **Animal:** bird, cat, cow, dog, horse, sheep
- **Vehicle:** aeroplane, bicycle, boat, bus, car, motorbike, train
- **Indoor:** bottle, chair, dining table, potted plant, sofa, tv/monitor



From <http://www.cs.cornell.edu/gaohuang/papers/DenseNet-CVPR-Slides.pdf>

Figure 2.8: High level diagram of DenseNet architecture

2.5 Related Work

2.5.1 Network Acceleration and Compression

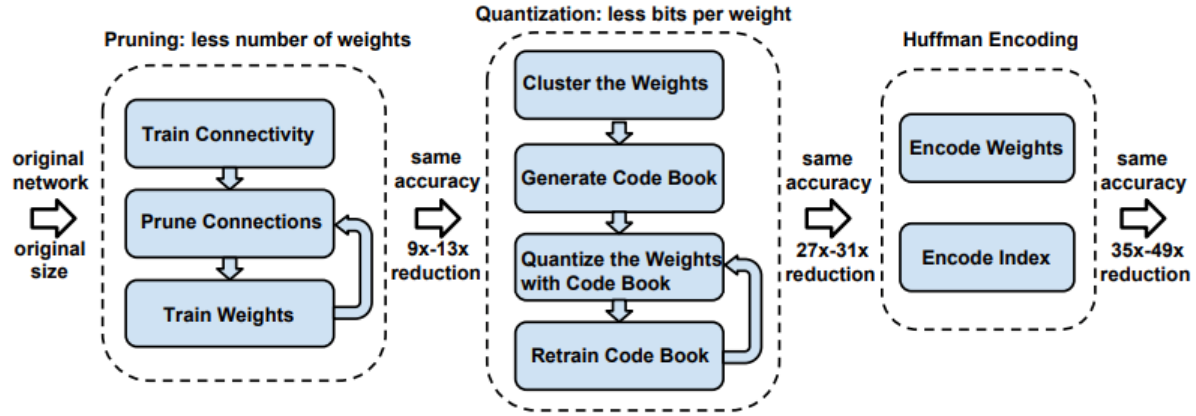
The high computational cost and the large model size are two main factors restricting CNN being used on embedded systems. Researchers have worked great efforts to solve this problem in different ways.

Weights Pruning and Quantization

Song Han et al. propose deep compression, a three-stage pipeline: pruning, trained quantization and Huffman coding [8], to reduce the storage requirement of neural networks by 35x to 49x without affecting their accuracy. (Fig. 2.9)

Network Binarization

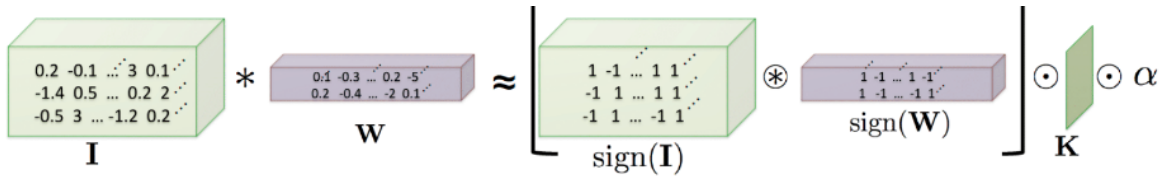
Several methods attempt to binarize the weights and the activations in neural networks to speed up inference time. The most famous one is XNOR-Net, proposed by Mohammad Rastegari et al. [29]. In XNOR-Net, both the filters and the input to convolutional layers are approximated with binary values. Therefore, the convolution operation can be estimated by XNOR operation. Theoretically, this can result in 58x faster speed and 32x smaller model size compared to the full-precision counterpart. However, in XNOR-Net, the accuracy drops greatly. For example,



From Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding [8]

Figure 2.9: Deep compression - three stage compression pipeline

for ResNet-18, XNOR-Net is 18.1% lower in top-1 accuracy than the full precision network on ImageNet ILSVRC 2012 (51.2% vs 69.3%). (Fig. 2.10)



From Xnor-net:Imagenet classification using binary convolutional neural networks [29]

Figure 2.10: Convolution with XNOR-bitcount

Knowledge Distilling

Another approach is knowledge distillation [11][33] which trains a student network from the softened output of an ensemble of deeper and wider networks, called teacher network. The main idea is to allow the student network to capture not only the information provided by the true labels, but also the feature learned by the teacher network.

Efficient Network Architectures

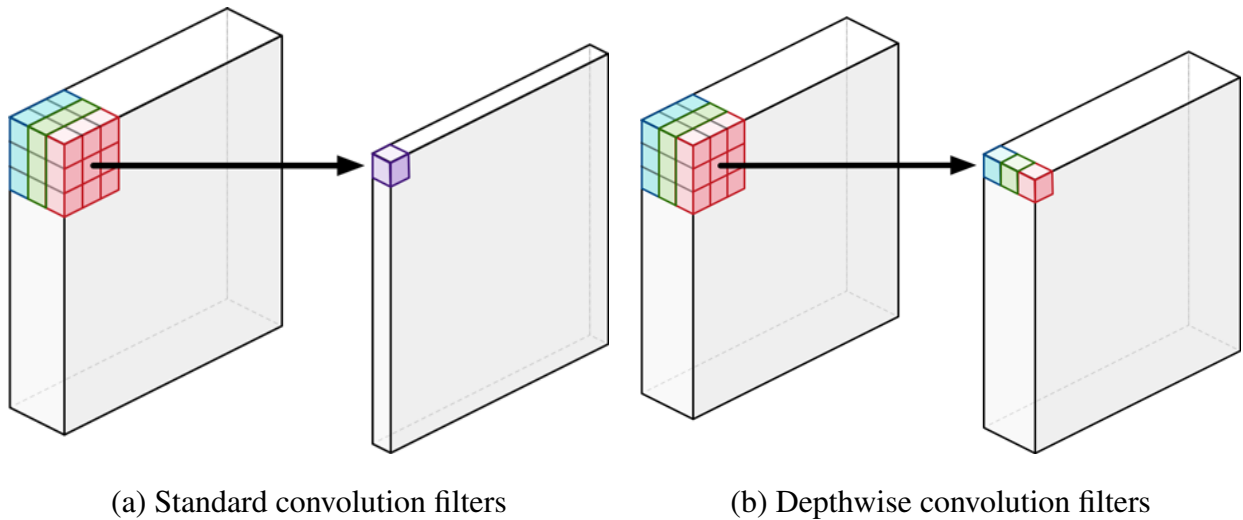
Instead of depending on a large model, efficient architecture design tries to design and train a network from scratch to meet the strictly constraints budget.

GoogLeNet [37] is the earliest architecture focusing on computational efficiency and low parameter count. It spends only 10% of computational cost of VGG and maintains a similar

accuracy to VGG. Many of its outstanding design ideas have greatly affected the current network design. For example, S. Hong et al. [20] widely uses inception block in PVANet, which is a high accuracy real time object detection system.

MobileNet [13] is a pioneering work on efficient model architecture and for the first time shows the VGG level accuracy for a fully convolutional network with about 600 Million MACs (the number of Multiply-Accumulates which measures the number of fused Multiplication and Addition operations) computational budget. MobileNet is based on a streamlined architecture that uses depthwise separable convolutions to build light weight CNNs. ShuffleNet [40] is another efficient model which utilizes pointwise group convolution and channel shuffle together with depthwise separable convolution to balance computational cost and accuracy. NASNet [41] is a complicated network obtained by reinforcement learning and model search. It achieves state of the art results both among large scale models and efficient models. NASNet architecture uses depthwise separable convolutions with different sizes ranging from 3×3 to 5×5 , 7×7 . However, the size of 7×7 is rarely used in human designed efficient models.

All these three famous efficient models heavily depend on depthwise separable convolution. Depthwise separable convolution is initially used in Inception models [37][38] to reduce the computation in the first few layers. Depthwise separable convolution is made up of two layers: depthwise convolution and pointwise convolution. Depthwise convolution applies a single filter per each input channel. Pointwise convolution is just a standard 1×1 convolution which is used to create a linear combination of the output of the depthwise layer. (Fig. 2.11)



From <http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/>

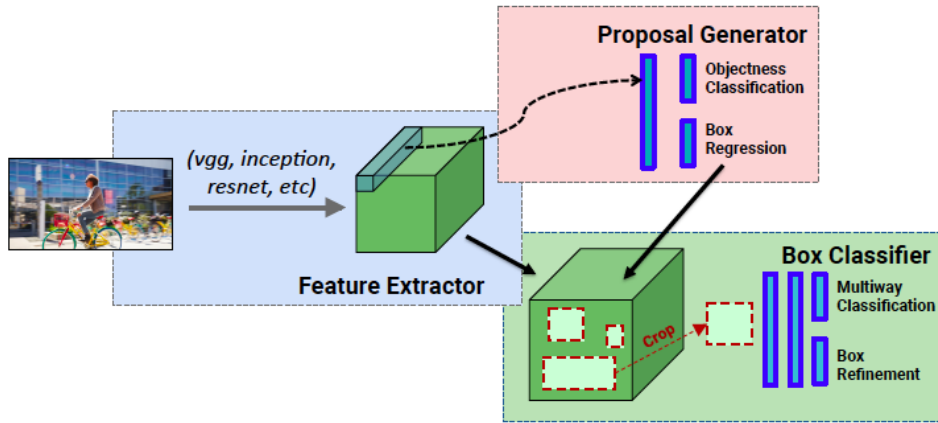
Figure 2.11: Standard convolution filters vs. depthwise convolution filters

2.5.2 Object Detection

Object detection is one of the main areas of researches in computer vision. Recent progress in object detection is heavily driven by the successful application of deep Convolutional Neural Networks (CNN). State-of-the-art CNN based object detection methods can be divided into two groups: region proposal-based detector and one-stage detector.

Proposal-based Detector

Proposal-based detector, also called two-stage detector, includes R-CNN [6], Fast RCNN [5], Faster R-CNN [32] and R-FCN [1]. There are two stages in proposal-based detector. Some potential object regions are generated at the first stage, and then classification and location processing are made on these proposed regions at the second stage. R-CNN uses selective search [39] to generate region proposal and then uses a CNN network to extract features from these proposed regions - each region is processed by the CNN network separately. Faster R-CNN improve the efficiency by sharing computation and using neural networks to generate the region proposals. (Fig. 2.12)

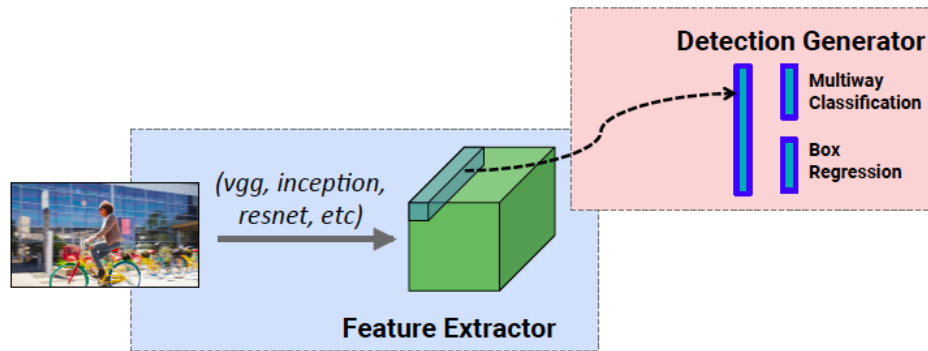


From Speed/accuracy trade-offs for modern convolutional object detectors [15]

Figure 2.12: High level diagram of Faster R-CNN architecture

One-stage Detector

One-stage detector, for example YOLO [30] and SSD [26], uses a single feed-forward convolutional network to directly predict object classes and locations. YOLO frames object detection as a regression problem that spatially separates bounding boxes and associates class probabilities. In this way, both object classes and locations can be directly predicted by a convolutional network. SSD improves YOLO in several aspects, including using multi scales of features for prediction and using default boxes and aspect ratios for adjusting varying object shapes. (Fig. 2.13)



From Speed/accuracy trade-offs for modern convolutional object detectors [15]

Figure 2.13: High level diagram of one-stage detector architecture

Numerous excellent studies focus on pushing one-stage detection algorithms forward.

- YOLOv2 [31] proposes various improvements on YOLO detection method - both on speed and accuracy. YOLOv2 uses a lightweight backbone network called Darknet19 and a fully-convolutional architecture to accelerate the speed. By combining batch normalization, dimension clusters, anchor box and multi-scale training, YOLOv2 has improved the accuracy by up to 15% in mAP on VOC2007.
- DSSD [4], and Residual Feature and Unified Prediction Network (RUN) [24] improve the accuracy of SSD algorithm by optimizing the network structure of the detection parts. DSSD combines the deconvolutional layers with the existing multiple layers to reflect the large-scale context. RUN enriches the representation power of feature maps using multi-way residual block and unified prediction module. Deeply Supervised Object Detector (DSOD) [34] shows a high accuracy object detector can be trained without relying on ImageNet pre-training model when it combines a variant DenseNet architecture with SSD framework. However, these optimizations result in a much slower speed in comparison to the original SSD since more operations are added. Huang et al. [15] investigates various combination of network structures and detection frameworks, and indicates that a detector combined SSD and MobileNet can achieve real time speeds and can be deployed on a mobile device.
- Additionally, Lin et al. [25] proposes a novel loss function called Focal Loss to resolve the extreme foreground-background class imbalance problem. The RetinaNet proposed in this paper, which combines Focal Loss with ResNet and FPN structure, achieves the state-of-the-art performance.

Chapter 3

PeleeNet: An Efficient Feature Extraction Network

3.1 Introduction

Tremendous progress has been made in recent years on object detection due to the use of convolutional neural network (CNN). A CNN based object detector depends on a powerful feature extraction network. The feature extraction network directly affects memory, speed and performance of the detector. In SSD framework, over 90% of computation cost is consumed by the feature extraction network. In this chapter, we explore an architecture design for the efficient feature extraction network.

A number of efficient network architectures have been proposed in recent years, for example, MobileNet [13], ShuffleNet [40], and NASNet-A [41]. These models have been shown to achieve VGG level accuracy on ImageNet ILSVRC classification task with only 1/30 of theoretical computational cost and model size. However, these models are heavily dependent on the depthwise separable convolution which lacks efficient implementation in most deep learning frameworks and hardware. Therefore, the actual speed is far less than the theoretical speed.

We propose a variant of DenseNet [14] architecture called PeleeNet which is built with conventional convolution operation instead. Our PeleeNet is designed to meet strict constraints on memory and computational budget. We propose the methods of two-way dense layer, stem block, transition layer without compression to improve the accuracy and also the methods of dynamic number of channels in bottleneck layer and post-activation to accelerate the speed. A series of controlled experiments on Stanford Dogs dataset and ImageNet ILSVRC 2012 dataset show the effectiveness of our design principles. Compared to the state-of-the-art architecture

MobileNet, PeleeNet achieves a higher accuracy by 6.53% and 11% lower in computational cost on Stanford Dogs dataset (80.03% top-1) and a higher accuracy by 0.6% on ILSVRC 2012 dataset (71.3% top 1). Based on the DenseNet architecture, which is well known for encouraging feature reuse and substantially reducing the number of parameters, our proposed PeleeNet is of only 66% of the model size of MobileNet.

3.2 Methodology

3.2.1 Review of DenseNet

DenseNet [14] consists of multiple dense blocks, each of which consists of multiple dense layers. Each dense layer produces k features, where k is referred to as the growth rate of the network. The greatest contribution of DenseNet is that the input of each layer is a concatenation of all feature maps generated by all preceding layers within the same dense block. (Fig. 3.1 Fig. 3.2) Some other key points in DenseNet are:

Composite function. The composite function used in DenseNet consists of three consecutive operations: batch normalization (BN) [17], followed by a rectified linear unit (ReLU) [7] and a 3 x 3 convolution (Conv).

Bottleneck layers. Following the practice in [9][38], DenseNet uses a 11 convolution as bottleneck layer before each 33 convolution to reduce the number of input feature-maps, and thus to improve computational efficiency. Each bottleneck layer produces $4k$ feature-maps (k is the growth rate).

Compression factor. Compression factor θ (range from 0 to 1 and default is 0.5) is introduced at transition layer to further improve model compactness. If a dense block contains m feature maps, the following transition layer will generate $\lceil \theta * m \rceil$ output feature maps. When $\theta = 1$, the number of feature-maps across transition layers remains unchanged.

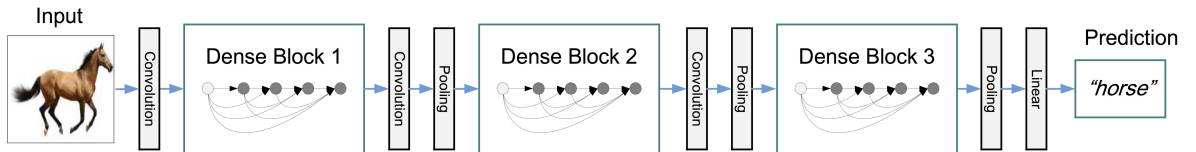


Figure 3.1: A deep DenseNet with three dense blocks

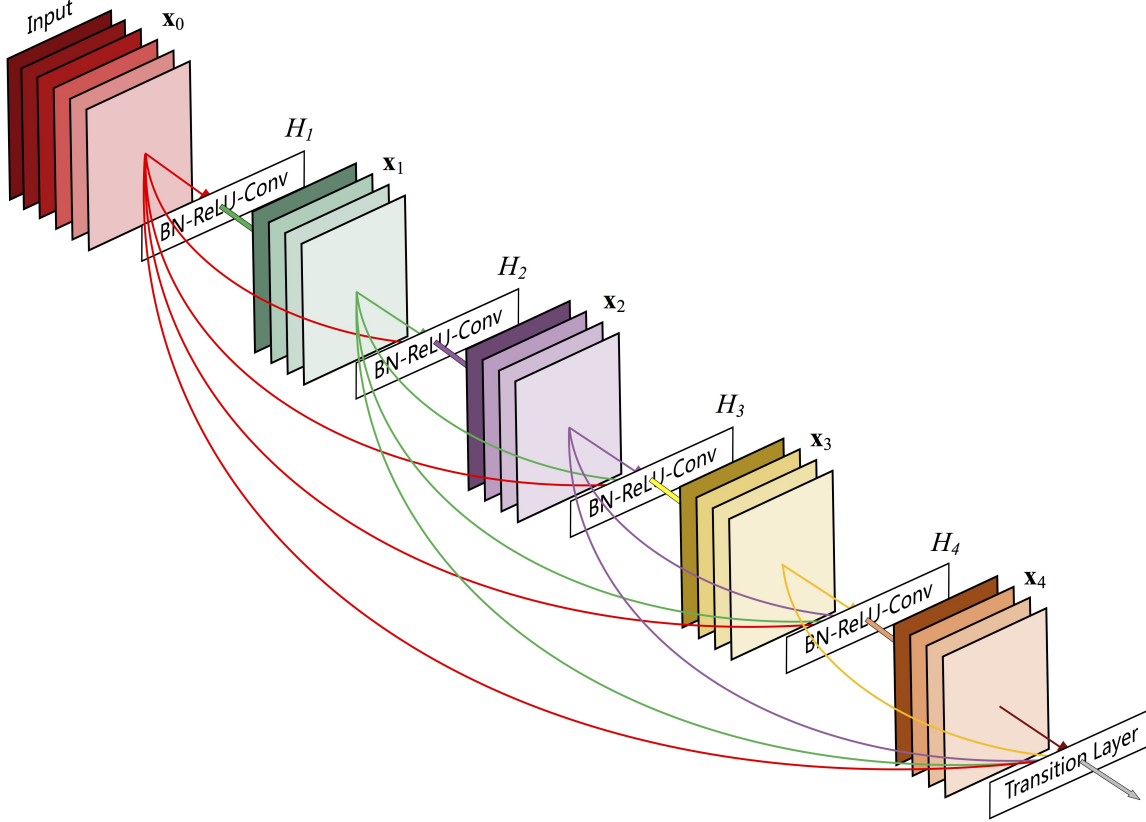


Figure 3.2: A dense block with 5 layers and growth rate 4

3.2.2 Two-Way Dense Layer

We use a 2-way dense layer to get different scales of receptive fields. One way of the layer uses a small kernel size (3×3), which is good enough to capture small-size objects. The other way of the layer uses a larger 5×5 kernel to learn visual patterns for large objects. To reduce the computational cost, we follow the common practice of replacing 5×5 convolution with two stacked 3×3 convolution [38]. The experimental results show that the 2-way dense layer achieves at least 2% higher accuracy on Stanford Dogs. (Fig. 3.3)

3.2.3 Stem Block

Motivated by DSOD [34] and Inception-v4 [36], we define a stem block that has the simplified structure of dense block. Stem block can effectively improve the feature expression ability without adding computational cost too much - better than other more expensive methods, e.g., increasing channels of the first convolution layer or increasing growth rate. (Fig. 3.4)

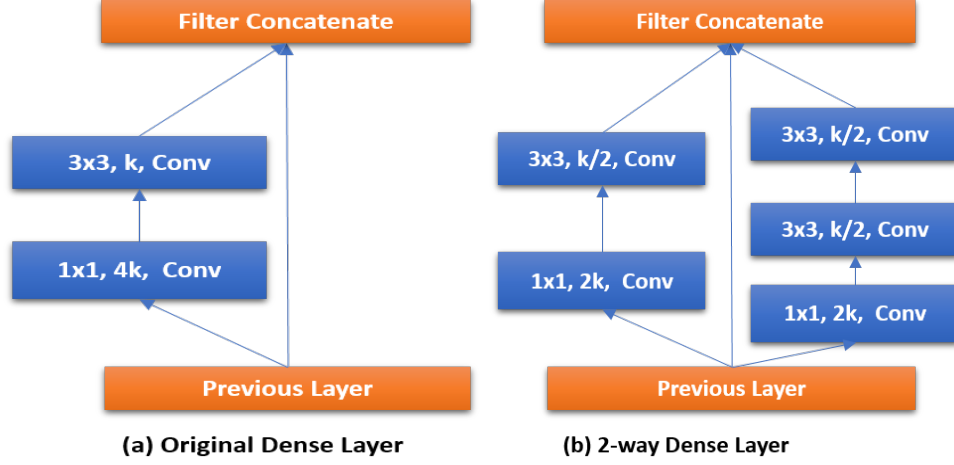


Figure 3.3: Two-way dense layer

3.2.4 Dynamic number of Channels in Bottleneck Layer

Another highlight is that the number of channels in bottleneck layer varies according to the input shape to make sure the number of channels in bottleneck layer does not exceed the number of its input channels. This method can effectively reduce the computational cost with little impact on the accuracy.

In DenseNet, the number of channels in the bottleneck layer is suggested to be 4 times the growth rate. Our experiments show that a larger bottleneck channels indicates a higher accuracy. However, a larger bottleneck channels also means an increasing computational cost. We observe that for the first layers, the number of bottleneck channels is much larger than the number of its input channels, which means that for these layers, bottleneck layer increases the computational cost instead of reducing the cost. As we can see from Table 3.1, for the first 4 layers, bottleneck layers increase about 3 times computational cost in total than the one without bottleneck layers. To maintain the consistency of the architecture, we still add the bottleneck layer to all dense layers, but the number is dynamically adjusted according to the input shape, to ensure that the number of channels does not exceed the input channels. Compared to a fixed 4-time-growth-rate scheme, the model with 4 time growth rate and dynamic scheme can reduce the computational cost by 28.5% with a small amount of loss of accuracy.

3.2.5 Transition Layer Without Compression

To improve model compactness, DenseNet proposes a compression factor θ in transition layer and set it to 0.5 to reduce half of the feature map in the transition layer. We always set this factor to be 1 in our models. Experimental result shows that $\theta = 1$ yields 1.1% higher Stanford

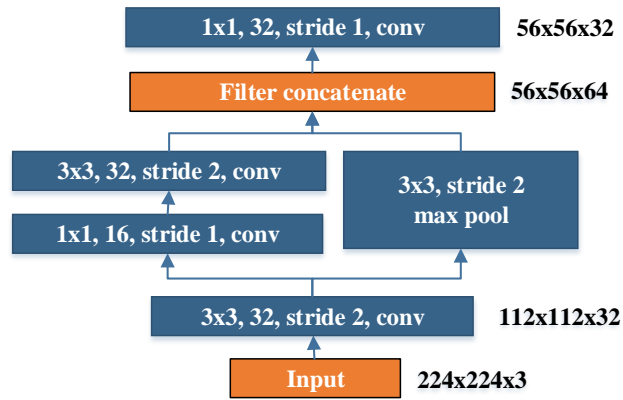


Figure 3.4: Structure of stem block

Dense Layer	Input Channels	Computational Cost (Million MACs)		
		Fixed 4 x	Without Bottleneck Layer	Dynamic + 4x
Layer 1	24	125	22	31
Layer 2	56	138	51	60
Layer 3	88	38	24	30
Layer 4	120	41	20	38
Total		342	117	159

Table 3.1: Computational cost of bottleneck layer

Dogs accuracy than $\theta = 0.5$.

3.2.6 Composite Function

We use the conventional wisdom of post-activation (Convolution - Batch Normalization - Relu) as our composite function instead of pre-activation (Batch Normalization - Relu - Convolution) used in DenseNet architecture. The main reason is not for accuracy but for the actual speed. For post-activation, all batch normalization layers can be merged with convolution layer at the inference stage, which can accelerate the speed greatly.

DenseNet uses a pre-activation composite function, which was originally proposed by Kaiming

He et al. [10] and has been proved to be able to effectively improve the performance of a very deep network. This method has made a significant contribution to DenseNet. Since DenseNet uses the concatenation operation to combine features, this method can achieve a unique scale and bias to previous features. For ResNet, there is no difference in the accuracy between the pre-activation and the post-activation when the depth of the model is less than 50. However, for DenseNet, pre-activation has a positive effect on both deep model and shallow model. Without pre-activation, the CIFAR-100 error of a 100-layer DenseNet grows from 22.27 to 24.30 [28]. Our experiments show that there is 0.92% lower accuracy on Stanford Dogs for a 41-layer DenseNet model without pre-activation.

Although this approach increases the accuracy, at least half of the batch normalization layers cannot be merged into the convolution layer to reduce the inference time. In consideration of this factor, in our architecture design, we use the traditional post-activation. To compensate for the negative impact caused by this change, we use a shallow and wide network structure. Meanwhile, a 2-way dense layer is used to improve the accuracy. We also add a 1×1 convolution layer after the last dense block to get the stronger representational abilities.

3.2.7 Overview of Architecture

Our proposed architecture is shown as follows in Table 3.2. The entire network consists of a stem block and four stages of feature extractor. Except the last stage, the last layer in each stage is average pooling layer with stride 2. A four-stage structure is a commonly used structure in the large model design. ShuffleNet et al. use three stage structure and shrink the feature map size at the beginning of each stage. Although this can effectively reduce computational cost, we argue that early stage features are very important for vision tasks, and that premature reducing the feature map size can impair representational abilities. Therefore, we still maintain a four-stage structure. The number of layers in the first two stages are specifically controlled to an acceptable range.

3.3 Experiments

3.3.1 Dataset

Except Section 3.3.7, all other experiments are based on Stanford Dogs dataset. As previously described, this Stanford Dogs dataset is a subset of ILSVRC 2012 rather than the original one. In this subset, there are 120 categories, about 150K training data and 6K validation data with 50 images per category.

Stage		Layer	Output Shape	
			PeleeNet (k=32)	PeleeNet (k=16)
Stage 0	Stem Block		56 x 56 x 32	56 x 56 x 24
Stage 1	Dense Block	DenseLayer x 3	28 x 28 x 128	28 x 28 x 72
	Transition Layer	1 x 1 conv, stride 1		
		2 x 2 average pool, stride 2		
Stage 2	Dense Block	DenseLayer x 4	14 x 14 x 256	14 x 14 x 136
	Transition Layer	1 x 1 conv, stride 1		
		2 x 2 average pool, stride 2		
Stage 3	Dense Block	DenseLayer x 8	7 x 7 x 512	7 x 7 x 264
	Transition Layer	1 x 1 conv, stride 1		
		2 x 2 average pool, stride 2		
Stage 4	Dense Block	DenseLayer x 6	7 x 7 x 704	7 x 7 x 360
	Transition Layer	1 x 1 conv, stride 1		
Classification Layer		7 x 7 global average pool	1 x 1 x 704	1 x 1 x 360
		1000D fully-connecte,softmax		

Table 3.2: Overview of PeleeNet architecture

Since both training data and validation data are from counterparts in ILSVRC 2012 without any change, we can evaluate a model pre-trained on ILSVRC 2012 on this customized Stanford Dogs dataset. By this way, we can get some baseline information to help evaluate our model design. We have evaluated some pre-trained models, e.g. MobileNet, DenseNet121, VGG16 and ResNet50. The accuracy of the pre-trained MobileNet is 73.5%, which is slightly higher than the MobileNet model we trained from scratch (72.9%) on this dataset.

3.3.2 Evaluation

Both Stanford Dogs and ILSVRC 2012 validation set are balanced dataset. The task performance on these datasets is evaluated using top-1 and top-5 accuracy. Top-1 accuracy means the predicted label must exactly match the true label. Top-5 accuracy means that any of the 5

highest probability answers must match the true label.

$$Accuracy = n^{-1} \sum_{i=1}^n 1(\hat{y}_i - y_i)$$

Cross Entropy Loss is a widely used loss function in classification problem. The true probability p_i is the true label, and the given distribution q_i is the predicted value of the current model. We can use cross entropy to get a measure for similarity between p and q :

$$H(p, q) = - \sum_x p(x) \log q(x)$$

Softmax Classifier is a widely used multi-class classifier in CNN. For softmax classifier, the cross-entropy loss is then given by:

$$L_i = - \log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right)$$

or equivalently $L_i = -f_{y_i} + \log \sum_j e^{f_j}$ where $f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$

3.3.3 Training Parameters

All models are trained by PyTorch, an open source machine learning library supported by Facebook. We follow most of the training settings and hyper-parameters used in ResNet on ILSVRC 2012 - except for the cosine learning rate decay policy used in section 3.3.6 and section 3.3.7.

3.3.4 Impact of Different Elements

We first evaluate the impact of the different elements (e.g., compress factor, bottleneck width, and pre-activation) of the dense block on accuracy and computational cost in order to support our decision making. We build a DenseNet-like network called DenseNet-41 as our baseline model. There are two differences between this model and the original DenseNet. The first one is the parameters of the first conv layer. There are 24 channels on the first conv layer instead of 64, the kernel size is changed from 7 x 7 to 3 x 3 as well. The second one is that the number of layers in each dense block is adjusted to meet the computational budget. The details of the architecture of DenseNet-41 can be seen at Appendix A.

Based on this architecture, we build 5 models as shown from Table 3.4. The table shows that:

Batch Size	256
Optimizer	SGD
Learning Rate	0.1
Learning Rate Decay Policy	Drop 0.1 every 30 epochs
Momentum	0.9
Weight Decay Rate	0.0001
Data Augmentation	Random sized crop
Total Epochs	120

Table 3.3: Experimental configuration for PeleeNet

DenseNet architecture is a perfect fit for embedded vision. The basic DenseNet-41 model already presents a good performance. Its accuracy is 1.52% higher than the one of MobileNet (75.02 vs. 73.5) at similar computational cost.

The compression factor hurts the feature expression. When the compression ratio is changed from 0.5 to 1, the accuracy increases by 1.1% (from 75.02% to 76.12%) with 20% computational cost increased as well.

Adjusting bottleneck channels is an effective way to balance computational cost and accuracy. Compared to the baseline model, when the number of bottleneck channel is changed to 3 times the growth rate and the compression factor is not applied, the accuracy increases by 0.45% (from 75.02% to 75.47%) with 5.3% less computation cost.

A larger bottleneck channels indicates a higher accuracy, but a larger bottleneck channels also means a increasing computational cost. For the channels with 3 times growth rate, the computational cost is 21% less than the one with 4 times growth rate, but the accuracy is also reduced by 0.65%.

Pre-activation composite function makes a significant contribution to accuracy increase. The model with pre-activation composite function is 0.92% higher in accuracy than the one with post-activation.

The dynamic bottleneck channels scheme can well balance the accuracy and computational cost. Compared to the scheme of fixed 4 times growth rate, dynamic scheme can save up to 28.5% of the computational cost. The computational cost is even far lower than the model with fixed 3 times growth rate with 0.28% higher accuracy.

Model	Dynamic bottleneck channels	Million MACs	Million Parameters	Top-1 Accuracy
MobileNet	N/A	569	4.24	73.5
DenseNet-41 (Bottleneck width = 4x Compression = 0.5)	✗	543	1.07	75.02
DenseNet-41 (Bottleneck width = 4x Compression = 1.0)	✗	652	1.57	76.12
DenseNet-41 (Bottleneck width = 4x Compression = 1.0) Post-activation	✗	652	1.57	75.2
DenseNet-41 (Bottleneck width = 3x Compression = 1.0)	✗	515	1.23	75.47
DenseNet-41 (Bottleneck width = 4x Compression = 1.0)	✓	466	1.49	75.75

Table 3.4: Impact of different elements in DenseNet-41

3.3.5 Effects of Different Feature Enhancement Methods

The second experiment compares different feature enhancement methods including the two-way dense layer, the stem block, increasing the number of dense layers and increasing the number of channels of the first convolution layer. All models are built with dynamic bottleneck channels scheme and without compression factor.

From Table 3.5, we can see that:

Stem block can effectively improve the accuracy. The result of stem block method is 0.87% higher in accuracy than the one without stem block. Stem block method outperforms the other two feature enhancement methods: increasing the channel of the first convolution layer and adding more dense layers. The result of stem block method is 0.82% higher in accuracy than the one of increasing the channel of the first convolution layer (76.57 vs. 75.75) and is 0.75% higher than the one of the method of adding more dense layers (76.57 vs. 75.82) , with much lower computational cost than these two methods.

Two-way dense layer can effectively improve accuracy without adding computational cost. The

model with two-way dense layer shows 2% higher accuracy than the one with one-way dense layer.

Early stage layers significantly increase accuracy. The accuracy increases by 0.55% when another dense layer added at stage 1.

No.	Model	Stem Block	2-way Dense Layer	Million MACs	Million Parameters	Top-1 Accuracy
1	DenseNet-41 (24-32-2466)	✗	✗	466	1.49	75.75
2	DenseNet-41 (36-32-2466)	✗	✗	517	1.55	75.75
3	DenseNet-45 (24-32-2486)	✗	✗	515	1.79	75.82
4	DenseNet-41 (24-32-2486)	✓	✗	502	1.5	76.57
5	DenseNet-45 (32-32-2486)	✓	✓	528	1.98	78.8
6	PeleeNet (32-32-3486)	✓	✓	507	2.18	79.25

DenseNet-41($A - k - B$) describes the network structure. A denotes the number of channels in the first convolution layer. k is the growth rate in dense blocks. B denotes the number of dense layer in each dense block.

Table 3.5: Effects of different feature enhancement methods

3.3.6 Results on Stanford Dogs

This section describes the result on Stanford Dogs and the result compared to other pre-trained models. We use a different data augmentation method in this section. Besides random-sized cropping, we also randomly adjust brightness and contrast of training images. This new data augmentation approach brings a 0.3% performance boost. Different from previous sections, the model is trained with a cosine learning rate annealing schedule (PeleeNet cosine), similar to what is used by [28][27].

Cosine Learning Rate Annealing means that the learning rate decays with a cosine shape (the learning rate of epoch t ($t \leq 120$) set to $0.5 * lr * (\cos(\pi * t/120) + 1)$). (Fig. 3.5)

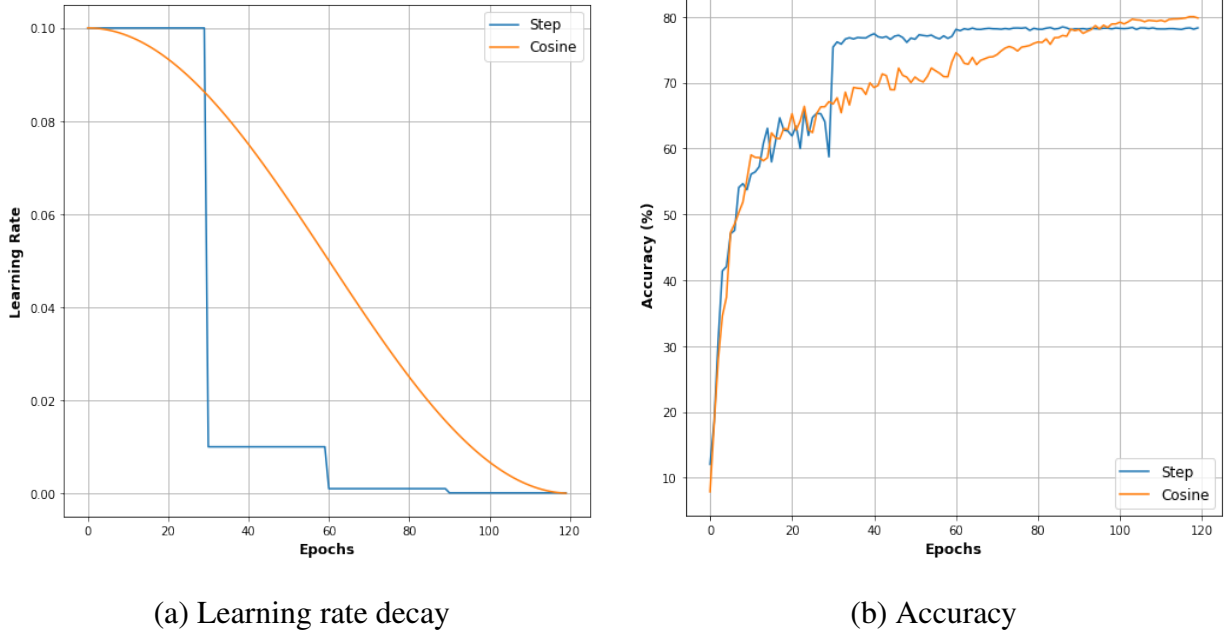


Figure 3.5: Cosine learning rate annealing vs. step learning rate decay

Based on the results of the above experiments, the effects of various design choices on the performance can be summarized as Table 3.7. After combining all these design choices, PeleeNet achieves 80.03% accuracy on Stanford Dogs, which is 5.01% higher in accuracy than the baseline DenseNet-41 at less computational cost.

3.3.7 Results on ILSVRC 2012

As we can see from Table 3.8, PeleeNet achieves a compelling result on ILSVRC 2012. PeleeNet is 0.6% more accurate than MobileNet and 0.4% more accurate than ShuffleNet.

3.4 Summary

DenseNet architecture is well suited for computing constrained scenarios. Because of its innovative connectivity pattern, the features can be effectively reused to build an efficient model without depthwise separable convolution.

The enhancements we propose can effectively improve model performance. The accuracy can be improved by up to 5.01% when they are combined.

Model	Million MACs	Million Parameters	Top-1 Accuracy
VGG16	15,346	14.71	75.45
ResNet50	3,832	23.48	79.48
DenseNet-121	2,833	6.87	78.65
1.0 MobileNet	569	3.32	73.5
PeleeNet (k=32)	507	2.18	79.55
PeleeNet (k=32)- cosine	507	2.18	80.03
0.5 MobileNet	150	1.34	66.8
PeleeNet (k=16)	150	0.55	75.2

Table 3.6: Results on Stanford Dogs

PeleeNet is good at the fine-grained image recognition. It achieves the state of the art result on Stanford Dogs dataset. The top 1 accuracy is 80.03%, which is 6.53% higher than that of MobileNet. This accuracy is even higher than that of DenseNet-121 and ResNet50. Moreover, the computational cost of PeleeNet is only 18.6% of the cost of DenseNet-121 and only 13.7% of the one in ResNet50.

PeleeNet achieves a compelling result on ImageNet ILSVRC 2012 dataset as well (Top 1 accuracy 71.3%). It outperforms MobileNet and ShuffleNet in terms of accuracy, model size, and computational cost.

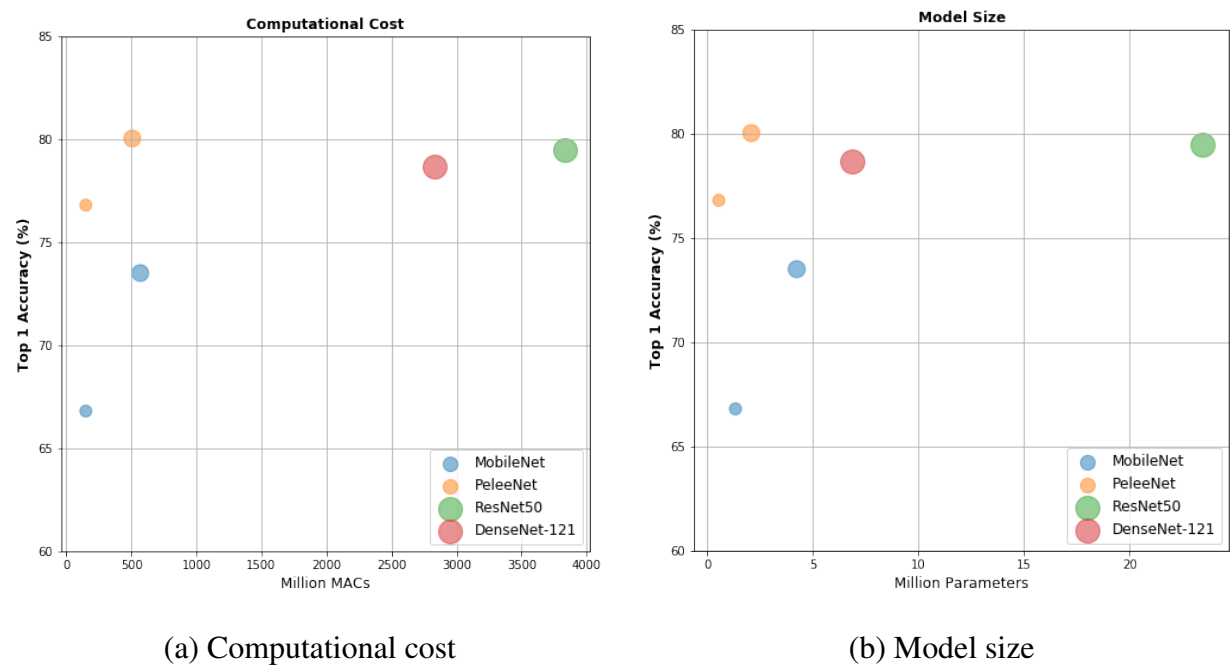


Figure 3.6: Comparison of different models on Stanford Dogs

	From DenseNet-41 to PeleeNet									
Transition layer without compression	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Post-activation		✓				✓	✓	✓	✓	
Dynamic bottleneck channels			✓	✓	✓	✓	✓	✓	✓	
Stem Block				✓	✓	✓	✓	✓	✓	
Two-way dense layer					✓	✓	✓	✓	✓	
Go deeper						✓	✓	✓	✓	
Extra data augmentation								✓	✓	
Cosine learning rate annealing										✓
Top 1 accuracy	75.02	76.1	75.2	75.8	76.8	78.8	79.25	79.55	80.03	

Table 3.7: Effects of various design choices and components on performance

Model	Million MACs	Million Parameters	Top-1 Accuracy	Top-5 Accuracy
VGG16	15,346	14.71	71.5	89.8
ResNet50	3,832	23.48	75.9	93.0
DenseNet-121	2,833	6.87	75.0	92.3
1.0 MobileNet	569	4.24	70.7	89.5
ShuffleNet 2x (g = 3)	524	5.2	70.9	-
NASNet-A	564	5.3	74.0	91.6
PeleeNet (k=32)	508	2.8	71.3	90.3

Table 3.8: Results on ILSVRC 2012

Chapter 4

Pelee: A Real-Time Object Detection System

4.1 Introduction

Considering the compelling speed advantage of the one-stage detection algorithms, we build our object detection system based on SSD [26] algorithm. Compared with YOLO [30] [31] series, the multi-scale feature maps used in SSD is a more efficient way to perform prediction. SSD with a 300 x 300 input size achieves better results than the 416 x 416 YOLOv2 [31] counterpart.

This chapter introduces our object detection system and the optimization for SSD. The main purpose of our optimization is to improve the speed with acceptable accuracy. Due to the time limitation, we mainly focus on the network architecture design. Our network structure can be combined with other optimization methods to further improve the accuracy in the future.

A number of enhancements have been proposed to manage a balance between speed and accuracy. Except for our efficient backbone network proposed in last chapter, we also build feature extraction network in a way different from the original SSD with a carefully selected set of 5 scale feature maps. In the meantime, we follow the design ideas of RUN [24] that encourages features to be passed along the feature extraction network. For each feature map used for detection, we build a residual block before conducting prediction. We also use small convolutional kernels to predict object categories and bounding box locations to reduce computational cost. In addition, we use quite different training hyperparameters.

Although these contributions may seem small independently, we note that the final system

achieves 70.9% mAP on VOC 2007, which is 13.4% higher in accuracy than TinyYOLOv2 (57.1% mAP). Compared with TinyYOLOv2, the system is also 2.88 times lower in computational cost and 2.92 times smaller in model size.

4.2 Methodology

4.2.1 Review of SSD

SSD [26] is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes. A non-maximum suppression step is used after this network to produce the final detection. Some key features of SSD are listed as follows:

Multi-scale Feature Maps for Detection

As it shows on Fig. 4.1, SSD adds convolutional feature layers, which decrease in size progressively, to the end of the backbone network. These extra layers together with some layers of the backbone network can detect object at multiple scales, which is different from YOLO [30] that only operates on a single scale feature map. There are 6 different scale of feature maps used for predicting: 38×38 , 19×19 , 10×10 , 5×5 , 3×3 , 1×1 .

Default Boxes and Aspect Ratios

SSD associates a set of default bounding boxes with each feature map cell. SSD predicts the offsets of each default box shape, as well as the per-class scores that indicate the presence of a class instance in each of those boxes. Specifically, for each box out of k at a given location, SSD computes c class scores and the 4 offsets relative to the original default box shape. This results in a total of $(c + 4) \times k \times m \times n$ outputs for a $m \times n$ feature map.

As it shows in Fig. 4.2, default boxes in SSD are similar to the anchor boxes used in Faster R-CNN [5]. However, SSD applies them to several feature maps of different resolutions.

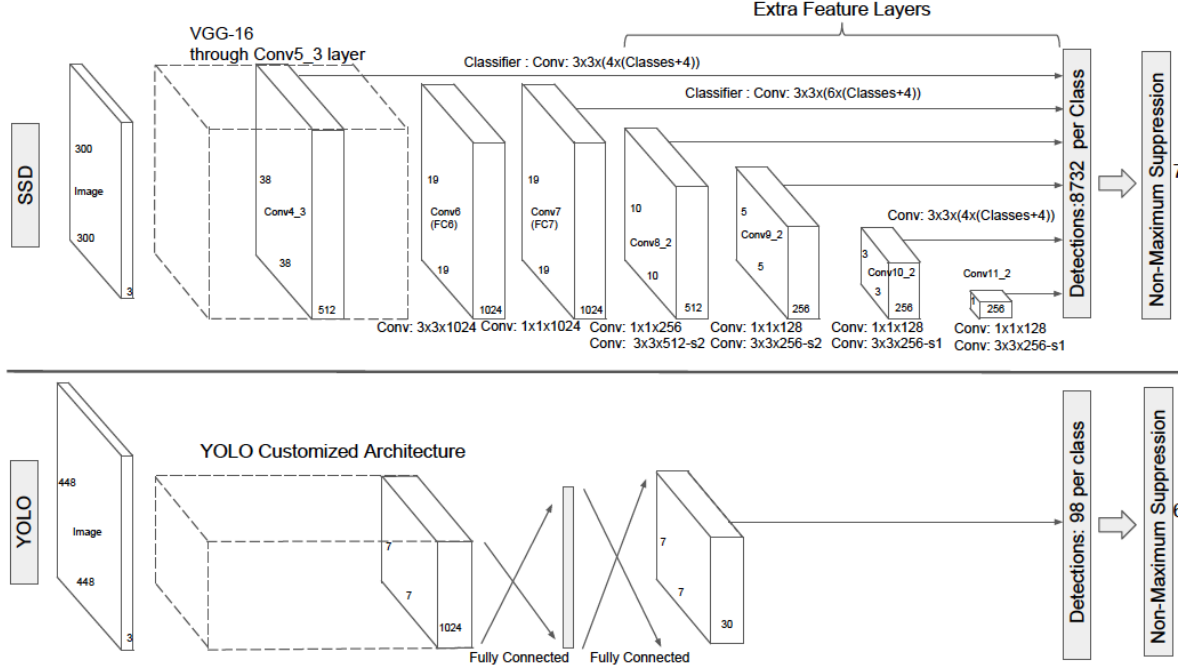


Figure 4.1: Architecture of SSD

Training Objective

The overall objective loss function in SSD is a weighted sum of the localization loss (loc) and the confidence loss (conf):

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

where N is the number of matched default boxes. If $N = 0$, the loss is set to 0.

The localization loss is a Smooth L1 loss between the predicted box (l) and the ground truth box (g) parameters.

$$L_{loc}(x, l, g) = \sum_{i \in Pos} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k smooth_{L1}(l_i^m - \hat{g}_j^m)$$

where

$$\begin{aligned} \hat{g}_j^{cx} &= (g_j^{cx} - d_i^{cx}) / d_i^w & \hat{g}_j^{cy} &= (g_j^{cy} - d_i^{cy}) / d_i^h \\ \hat{g}_j^w &= \log\left(\frac{g_j^w}{d_i^w}\right) & \hat{g}_j^h &= \log\left(\frac{g_j^h}{d_i^h}\right) \end{aligned}$$

The confidence loss is the softmax loss over multiple classes confidences (c).

$$L_{conf}(x, c) = - \sum_{i \in Pos}^N x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg}^N \log(\hat{c}_i^0)$$

where $\hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$

Hard Negative Mining

In SSD, after the matching step, most of the default boxes are negative samples, especially when the number of possible default boxes is large. This introduces a significant imbalance between the positive and negative training examples. The hard-negative mining is used to keep the ratio between the negatives and positives at most 3:1. Hard negative mining is the processing that all default boxes are sorted by the confidence loss and only the highest N boxes are selected as the negatives. This leads to faster optimization and a more stable training.

4.2.2 Feature Map Selection

There are 5 scales of feature maps used in our system for prediction: 19 x 19, 10 x 10, 5 x 5, 3 x 3, and 1 x 1. We do not use 38 x 38 feature map layer to ensure a balance able to be reached between speed and accuracy.

The 38 x 38 feature map has been shown to have a great positive impact on network accuracy in large scale models. For example, the accuracy of the SSD model with 38 x 38 feature map is 1.8% higher than that without 38 x 38 feature map on VOC2007. Our experiments also show that the RUN model with 38 x 38 feature map achieves 4.1% higher accuracy than the one without 38 x 38 feature map in VOC2007 (78.3% vs 74.2%).

However, 38 x 38 scale feature map does not have an huge influence on accuracy on an efficient model. The possible reason is that efficient models focus on reducing the layer or number of channels of the first two blocks which assume the biggest proportion of the computational cost. Therefore, the representation power of 38 x 38 in these efficient models is not as strong as that in large scale models. Meanwhile, 38 x 38 scale feature map takes up 53.4% of the computational cost of all prediction layers. Huang et al. [15] also do not use 38 x 38 scale feature map when combining SSD with MobileNet. However, they add another 2 x 2 feature map to keep 6 scales of feature map used for prediction, which is different from our solution. Table 4.1 shows settings of default boxes for different algorithms in details.

		Original SSD	SSD+MobileNet[15]	Pelee
Prediction 1	Feature Map Size	38 x 38	19 x 19	19 x 19
	Default Box Size	30	60	30.4
	Aspect Ratio	[1, 2, 1/2]	[1, 2, 1/2]	[1, 2, 1/2, 3, 1/3]
Prediction 2	Feature Map Size	19 x 19	10 x 10	19 x 19
	Default Box Size	60	105	60.8
	Aspect Ratio	[1, 2, 1/2, 3, 1/3]	[1, 2, 1/2, 3, 1/3]	[1, 2, 1/2, 3, 1/3]
Prediction 3	Feature Map Size	10 x 10	5 x 5	10 x 10
	Default Box Size	111	150	112.48
	Aspect Ratio	[1, 2, 1/2, 3, 1/3]	[1, 2, 1/2, 3, 1/3]	[1, 2, 1/2, 3, 1/3]
Prediction 4	Feature Map Size	5 x 5	3 x 3	5 x 5
	Default Box Size	162	195	164.16
	Aspect Ratio	[1, 2, 1/2, 3, 1/3]	[1, 2, 1/2, 3, 1/3]	[1, 2, 1/2, 3, 1/3]
Prediction 5	Feature Map Size	3 x 3	2 x 2	3 x 3
	Default Box Size	213	240	215.84
	Aspect Ratio	[1, 2, 1/2]	[1, 2, 1/2, 3, 1/3]	[1, 2, 1/2, 3, 1/3]
Prediction 6	Feature Map Size	1 x 1	1 x 1	1 x 1
	Default Box Size	264	285	267.52
	Aspect Ratio	[1, 2, 1/2]	[1, 2, 1/2, 3, 1/3]	[1, 2, 1/2, 3, 1/3]

Table 4.1: Parameters of feature maps

4.2.3 Residual Prediction Block

We follow the design ideas of RUN [24] and insert a two-way residual block (ResBlock) for each level of feature maps to separate and decouple the backbone network and the prediction module.

In the original SSD, the earlier feature map (for example 38 x 38) needs to learn high-level abstraction to perform prediction. At the same time, it also needs to learn low-level local features to transfer to the next feature maps. This not only makes the network hard to train, but also causes the overall performance decreasing. Wei Liu et al. add L2 normalization layer between the 38 x 38 feature map and the prediction module in SSD. Since the above problem is not merely shown on the 38 x 38 feature map layer, RUN proposed a multi-way ResBlock to

decouple backbone network from the prediction module. This approach clearly differentiates the features to be used for prediction from those to be delivered to the next layer. Moreover, it can prevent the gradient of the prediction module from flowing directly into the feature map of the backbone network. To meet our computational budget, we only use two-way ResBlock. (Fig. ??)

4.2.4 Small Convolutional Kernel for Prediction

We apply 1x1 convolutional kernels instead of 3x3 convolutional kernels used in original SSD to predict category scores and box offsets. Since our feature maps are built with two-way residual blocks, which have extracted the nearby features from the raw feature maps, 1x1 kernels are capable enough for conducting prediction. Our experiments show that the accuracy of the model using 1x1 kernels is almost same as the one of the model using 3x3 kernels. However, 1x1 kernels reduce the computational cost by 21.5%.

4.3 Experiments

4.3.1 Dataset

The proposed method is evaluated on PASCAL VOC 2007 and PASCAL VOC 2012 dataset. Following the common practice, we use VOC 2007 train+val data and VOC 2012 train+val data as our training data and VOC 2007 test data as our validation data set. There are about 16.5K training images in total.

4.3.2 Evaluation

Bounding Box Evaluation

For detection, a common way to determine if one object proposal is right is **Intersection over Union** (IoU). Detections are assigned to ground truth objects and are judged to be true/false positives by measuring bounding box overlap. To be considered a correct detection, the overlap ratio IoU between the predicted bounding box B_p and ground truth bounding box B_{gt} must exceed 0.5 (50%) based on the formula

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

where $B_p \cap B_{gt}$ denotes the intersection of the predicted and ground truth bounding boxes and $B_p \cup B_{gt}$ their union.

Mean Average Precision (mAP)

The interpolated average precision (Salton and McGill 1986) is used on VOC2007 to evaluate detection [3]. For a given class, the precision/recall curve can be computed. Recall is defined as the proportion of all positive examples ranked above a given rank. Precision is the proportion of all examples above that rank which are from the positive class. The AP summarizes the shape of the precision/recall curve, and is defined as the mean precision at a set of eleven equally spaced recall levels $[0, 0.1, \dots, 1]$:

$$mAP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} P_{interp}(r)$$

The precision at each recall level r is interpolated by taking the maximum precision measured for a method for which the corresponding recall exceeds r

$$P_{interp}(r) = \max_{\tilde{r}: \tilde{r} \geq r} p(\tilde{r})$$

where $p(\tilde{r})$ is the measured precision at recall \tilde{r} .

$$Precision = \frac{\sum TruePositive}{\sum TruePositive + \sum FalsePositive}$$

$$Recall = \frac{\sum TruePositive}{\sum TruePositive + \sum FalseNegative}$$

True Positive: a proposal is made for class c and it actually is an object of class c .

False Positive: a proposal is made for class c , but it is not an object of class c .

False Negative: a proposal is made for non-class c , but it actually is an object of class c .

4.3.3 Data Augmentation

We follow the data augmentation strategies used in SSD. Each training image is randomly sampled by one of the following options (Fig. 4.5):

- Use the entire original input image.
- Sample a patch that the IoU with the target object is of 0.1, 0.3, 0.5, 0.7, or 0.9.
- Randomly sample a patch.

- Randomly expand a sampled image.

The size of each sampled patch is $[0.1, 1]$ of the original image size, and the aspect ratio is between $1/2$ and 2 . The overlapped part of the ground truth box will be kept if the center of it is in the sampled patch. After the sampling step, each sampled patch is re-sized to a fixed size and is horizontally flipped with probability of 0.5 , in addition to applying some photo-metric distortions similar to those described in [12].

4.3.4 Training Parameters

Our codes are based on the source codes of SSD¹ and are trained with Caffe [18]. The training parameters are listed as follows. Our batch size and learning rate decay policy are different from the original SSD's.

Batch Size	128
Optimizer	SGD
Learning Rate	0.02
Learning Rate Decay Policy	Drop 0.1 at 20,000 and 25,000 iterations
Momentum	0.9
Weight Decay Rate	0.0005
Total Iterations	40,000

Table 4.2: Training parameters of Pelee

4.3.5 Results on PASCAL VOC 2007

It can be seen from Table 4.3 and Table 4.4 that:

- The 38×38 feature map does not have an huge influence on accuracy. The model using 38×38 feature map achieves 0.7% higher accuracy than the model without using 38×38 feature map (69.3 vs. 68.6) at 24.6% more computational cost.
- Residual prediction block can effectively improve the accuracy and does not increase computational cost too much. The model with residual prediction block achieves 70.6

¹<https://github.com/weiliu89/caffe/tree/ssd>

% mAP, which is 2.0% higher in accuracy than the model without residual prediction block.

- The accuracy of the model using 1x1 kernels for prediction is almost same as the one of the model using 3x3 kernels. However, 1x1 kernels reduce the computational cost by 21.5% and the model size by 33.9%.

38x38 Feature	ResBlock	Kernel Size For Prediction	Million MACs	Million Parameters	mAP
✓	✗	3x3	1670	5.69	69.3
✗	✗	3x3	1340	5.63	68.6
✗	✓	3x3	1470	7.27	70.8
✗	✓	1x1	1210	5.43	70.9

Table 4.3: Effects of various design choices on performance

Detection Frameworks	Million MACs	Million Parameters	mAP (Larger is better)
Tiny-YOLOv2	3490	15.86	57.1
YOLOv2-288	7940	57.96	69.0
SSD+MobileNet	1150	5.78	68
Pelee	1210	5.43	70.9

Table 4.4: Results on PASCAL VOC 2007

4.4 Summary

In this chapter, we combine our proposed PeleeNet with SSD framework. Our experiments shown that PeleeNet has a good generalization ability. All object detection models yield good results on VOC 2007 dataset. Meanwhile, our proposed enhancements can effectively balance detection speed and accuracy. By applying the methods of carefully selected multi-scale feature maps, residual prediction block and small convolutional filters for prediction, our proposed system achieves 70.9% mAP on VOC 2007 dataset with 1,210M MACs computational cost, which is 13% higher in accuracy and 2.88 times lower in computational cost than TinyYOLOv2.

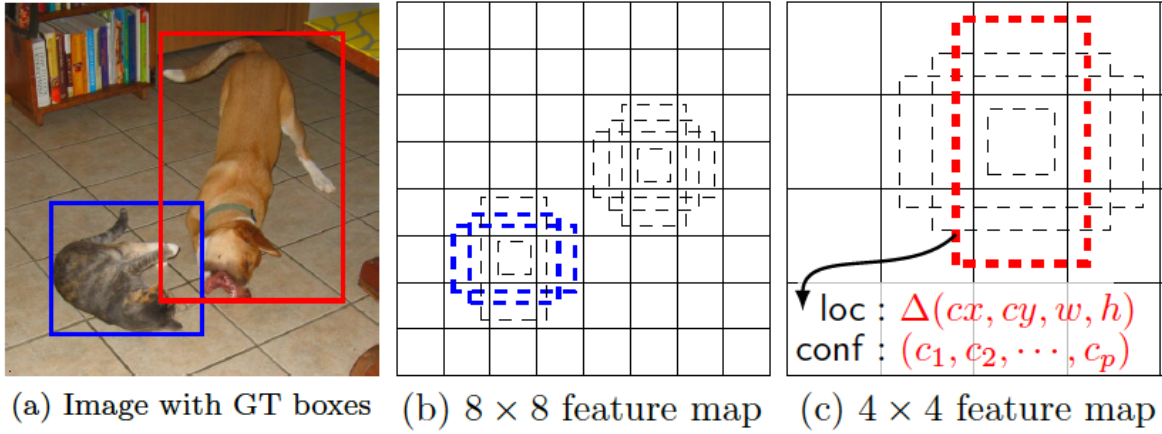


Figure 4.2: Default boxes and aspect ratios in SSD

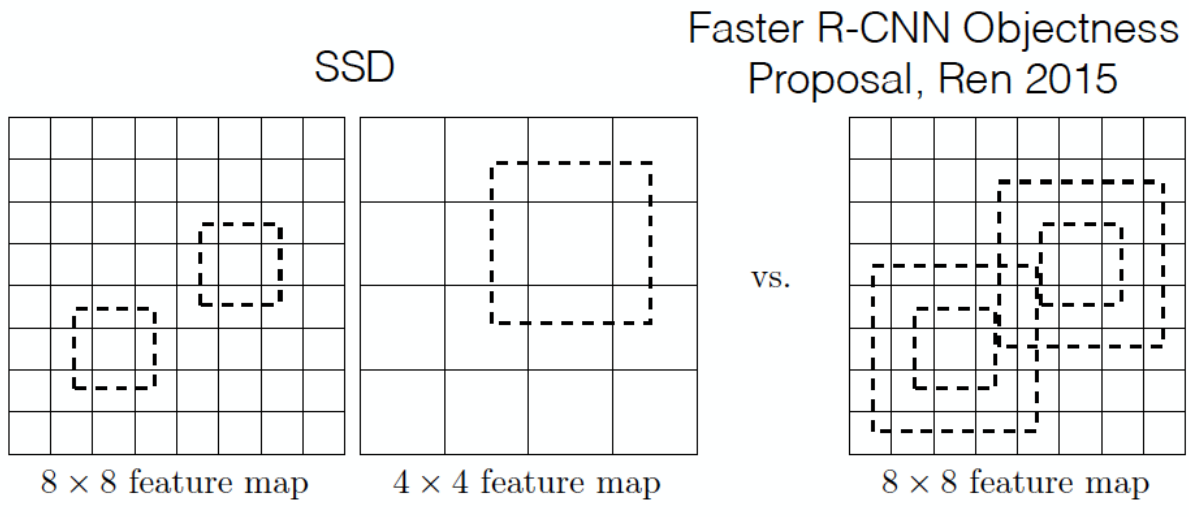


Figure 4.3: Default boxes in SSD vs. anchor boxes in Faster R-CNN

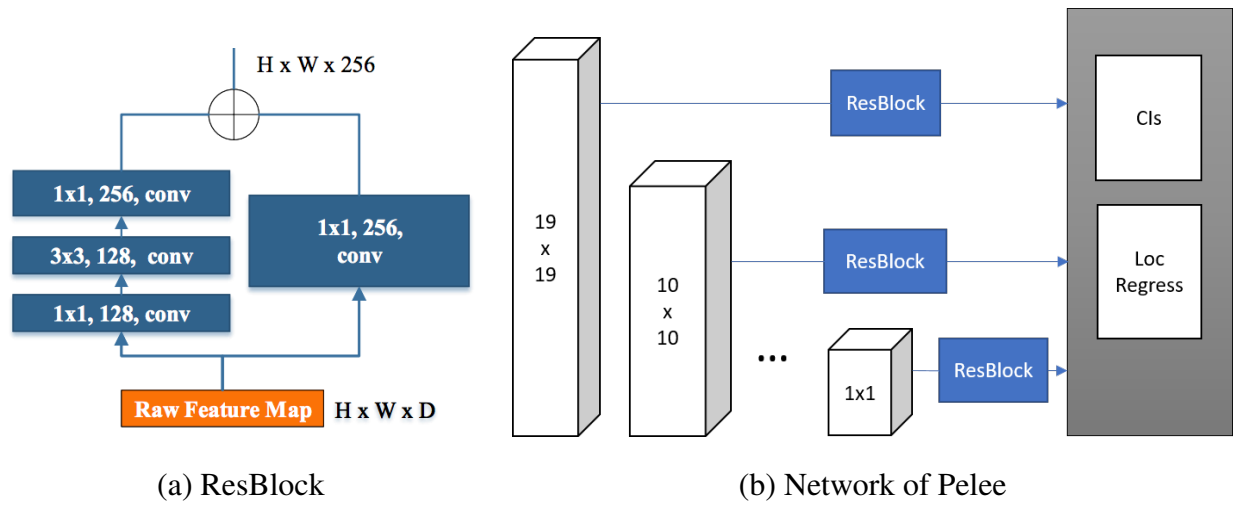


Figure 4.4: Residual Prediction Block

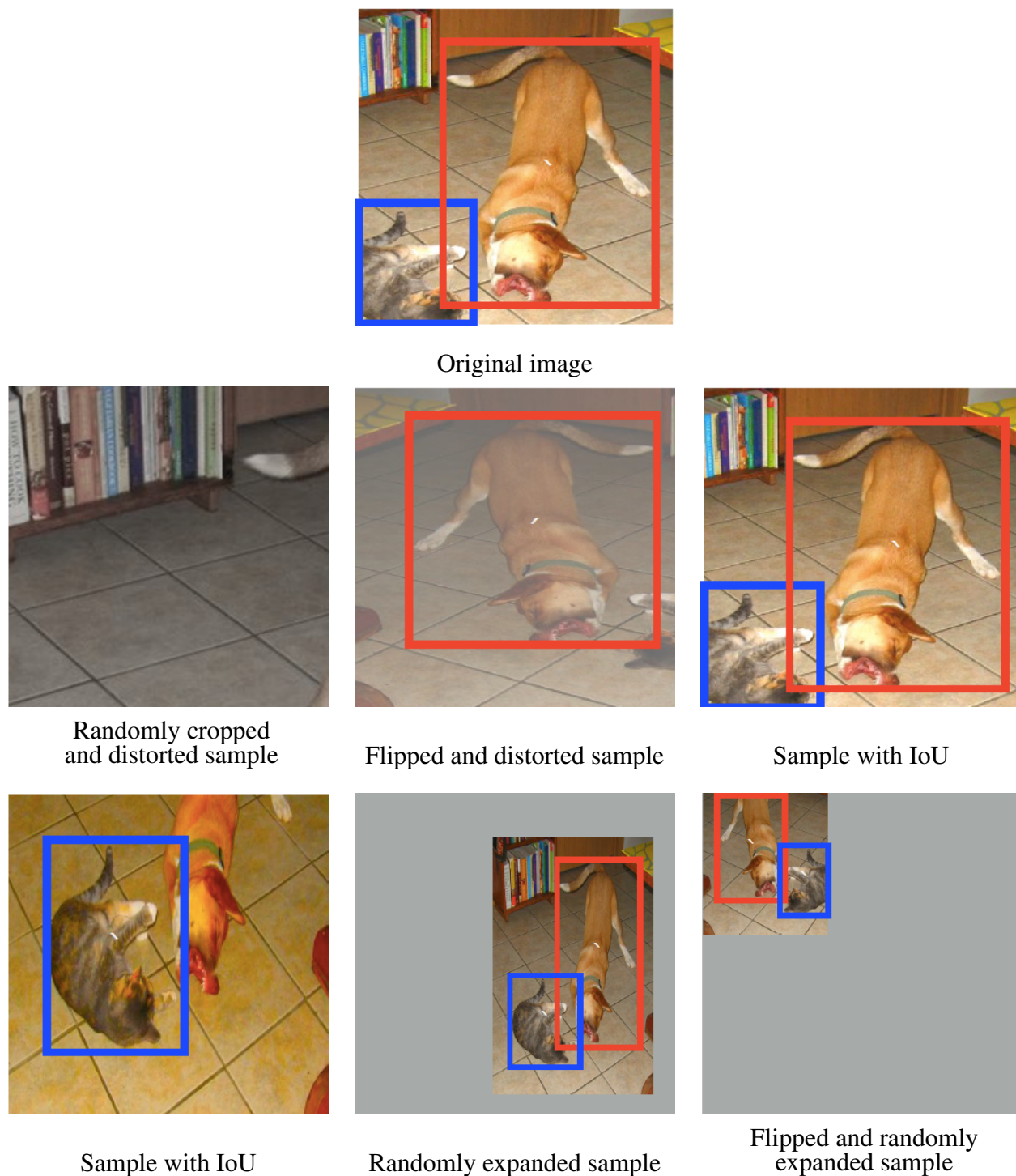


Figure 4.5: Data augmentation in SSD

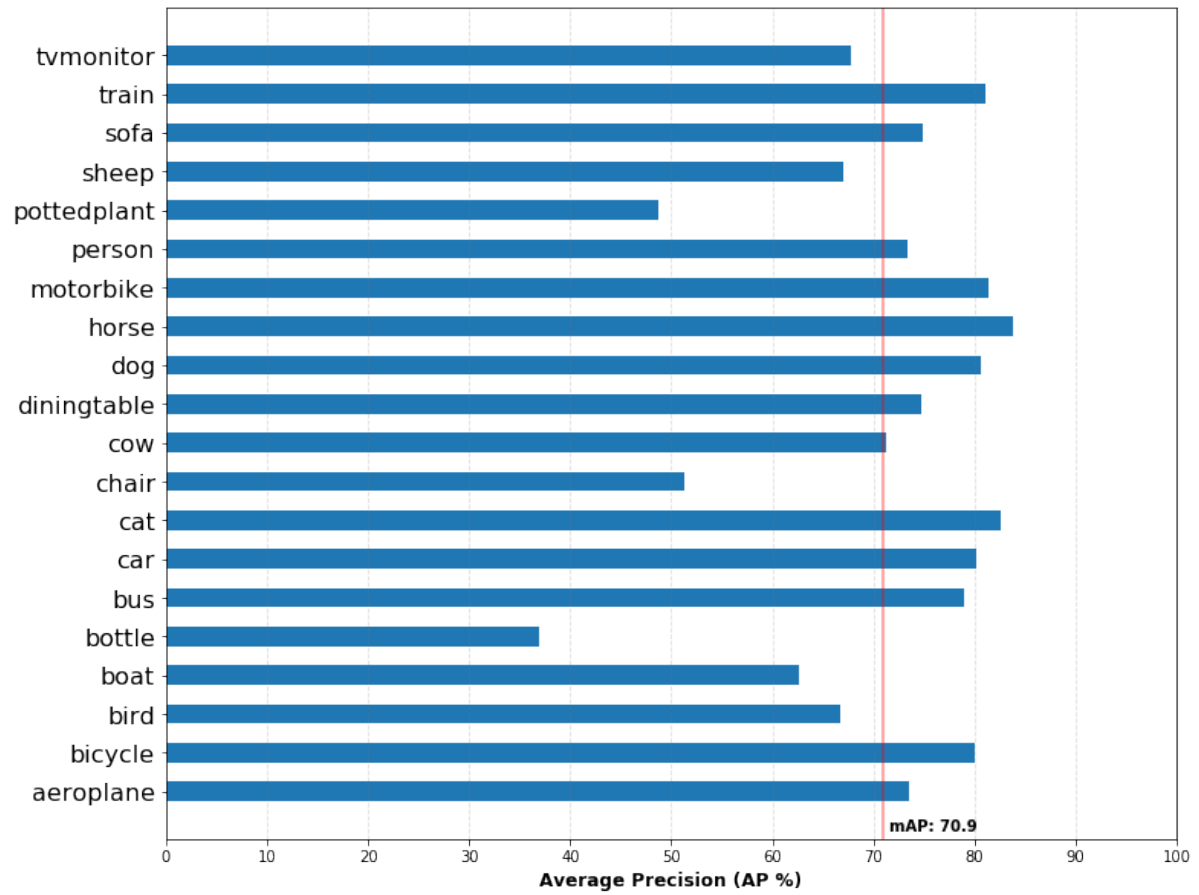
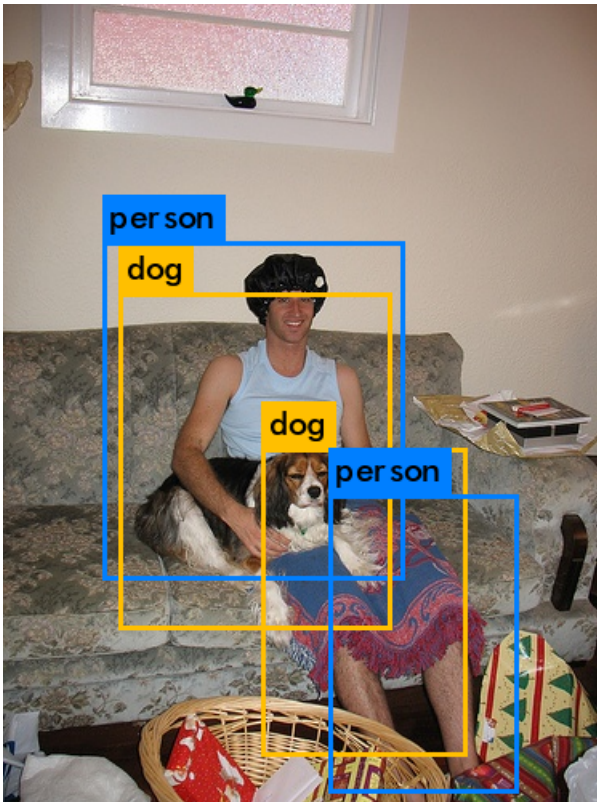
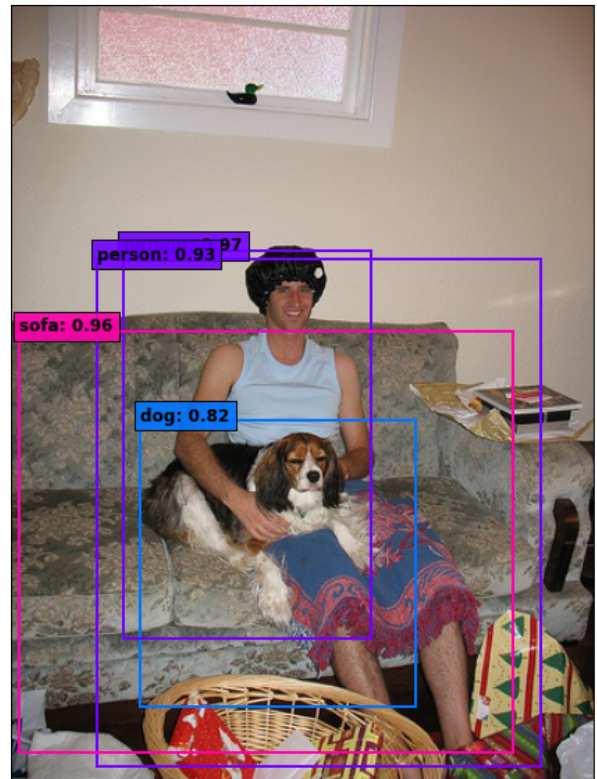


Figure 4.6: Average Precision on VOC 2007



TinyYOLOv2



Pelee

Compared to TinyYOLOv2, our proposed Pelee can correctly detect sofa and also localize all kinds of objects in the image correctly. More detection examples can be seen on Appendix C.

Figure 4.7: Detection examples: TinyYOLOv2 vs. Pelee

Chapter 5

Benchmark on Real Devices

5.1 Introduction

Counting MACs (the number of multiply-accumulates) is widely used to measure the computational cost. However, it cannot replace the speed test on real devices, considering that there are many other factors that may influence the actual time cost, e.g. caching, I/O, hardware optimization etc.,.

This chapter evaluates the performance of efficient models on mobile phones and the performance of several one-stage object detection models on CPU and GPU. We only evaluate the speed. The accuracy information originates from the related peer-reviewed paper and the previous experiments. It is assumed that the model conversion will not result in reduced accuracy.

5.2 Benchmark on Mobile Phone

We evaluate the speed of some efficient classification models, including MobileNet and our proposed PeleeNet, on ImageNet ILSVRC 2012 and the speed of some efficient object detection models, including Tiny-YOLOv2, SSD + MobileNet and ours, on VOC 2007. All the models are directly converted from Caffe models without any model specific optimization being done.

The speed is calculated by the average time of processing 100 pictures. We run 100 picture processing for 10 times separately and average the time. For object detection models, the time reported here includes the image pre-processing time, but it does not include the time of the post

processing part (decoding the bounding-boxes and performing non-maximum suppression). Usually post processing is done on the CPU, which can be executed asynchronously with pre-processing and processing that are executed on mobile GPU. Hence, the actual speed should be very close to our test result.

Our results are tested on iPhone 6s and iPhone 8. Apple iPhone 6s is a phone released in September of 2015. Its hardware capability is the same as or lower than the mainstream smart phone and embedded vision devices. Apple iPhone 8 is released in September of 2017, which is equipped with the state of the art hardware and the dedicated neural network hardware accelerator. We choose iPhone as our test equipment, mainly because with the CoreML toolchains, a trained machine learning model can be easily converted and integrated into the iOS application. CoreML is a foundational machine learning framework used in Apple products, which features fast performance and easy integration. (Fig. 5.1)

5.2.1 Benchmark of Efficient Classification Model on ILSVRC 2012

It can be seen from Table 5.1 that:

On iPhone 6s, our proposed model runs at a similar speed as MobileNet, although our model indicates 11% lower computational cost. The reason may be that MobileNet is a streamlined shallow and wide model with about 27 convolutional layers, while our model is a multi-branch and narrow model with 113 convolutional layers. Shallow models are easier to be paralleled for more efficient execution. We try another shallow model named PeleeNet-shallow (32-32-3443). It has almost the same architecture as PeleeNet. The only difference is that the last two dense blocks have half number of dense layers with doubled growth rate (78 convolutional layers in total). PeleeNet-shallow runs faster than MobileNet and PeleeNet, although it has a higher computational cost than PeleeNet.

Models on iPhone 8 are about 25% to 30% faster than those on iPhone 6s. MobileNet runs almost the same speed as PeleeNet-shallow on iPhone 8.

5.2.2 Benchmark of Efficient One-stage Detector on VOC 2007

It can be seen from Table 5.2 that:

Pelee runs 17.1 FPS on iPhone6s and 23.6 FPS on iPhone8, which is lightly faster than SSD+MobileNet. On iPhone 6s, both SSD+MobileNet and our proposed model are faster than Tiny-YOLOv2. SSD+MobileNet is 81.3% faster than Tiny-YOLOv2 and our model is 83.9% faster than Tiny-YOLOv2.

Model	Million MACs	Million Parameters	Accuracy		Speed (ms/image)	
			Top-1	Top-5	iPhone6s	iPhone8
1.0 MobileNet	569	4.24	70.7	89.5	47.3	36.0
PeleeNet (k=32)	508	2.8	71.3	90.3	47.5	38.3
PeleeNet-shallow (32-32-3443)	542	2.1	-	-	44.3	35.8

Table 5.1: Benchmark of efficient classification models on ILSVRC 2012

Model	Million MACs	Million Parameters	mAP	Speed (FPS)	
				iPhone6s	iPhone8
Tiny-YOLOv2 [31]	3490	15.86	57.1	9.3	23.8
SSD+MobileNet [15]	1150	5.77	68	16.1	22.8
Pelee	1210	5.43	70.9	17.1	23.6

Table 5.2: Benchmark of efficient one-stage detector on VOC 2007

5.3 Benchmark on CPU and GPU

This section evaluates the performance of some one-stage detectors on NVIDIA GTX 1080 Ti GPU and Intel i7-6700K @ 4.00GHz CPU. All models are converted to Caffe [18] models and evaluated by Caffe time tool. With the exception of DSOD, the batch normalization layers of other models have been merged into the convolution layers. The speed on Titan X GPU are quoted from related papers. The results of the column 1080 Ti and Intel i7 are tested by our experiments.

It can be seen from Table 5.3 that:

The theoretical computational cost cannot provide any reference for the actual speed on GPU. The lower computational cost does not represent faster speed. For example, TinyYOLOv2 runs slightly faster than our proposed Pelee on GPU (119.5 FPS vs 117.5 FPS), although the computational cost of tinyYOLOv2 is 2.88 times higher than the cost of Pelee. Both YOLOv2 and SSD are built in a shallow and wide style. The computational cost of YOLOv2-416 is 1 time lower than the cost of SSD. However, the actual speed of YOLOv2-416 on GTX 1080 Ti is 40.6% slower than SSD's.

The performance of MobileNet heavily depends on an efficient implementation of depthwise separable convolution. Under the same hardware and deep learning framework, there is about 9 times of gap in speed between cuDNN v5.0 and cuDNN v7.0. SSD+MobileNet [15] reaches 212 FPS on cuDNN v7.0 while only 24 FPS on cuDNN v5.0. The performance on cuDNN v5.0 is much slower than other efficient models’.

On CPU, both our proposed model and SSD+MobileNet run at about 6 FPS, which are 1.5 times faster than Tiny-YOLOv2. All high accuracy models ($mAP > 76\%$) on CPU run at a speed slower than 1 FPS.

It is unexpected that the actual speed of the efficient models on iPhone is much faster than that on a powerful desktop CPU. The speed of TinyYOLOv2 on iPhone 6s is 2.8 times faster than that on Intel i7-6700K. Both SSD+MobileNet and our proposed model can run at a 1.6 times faster speed on iPhone 6s than on Intel i7-6700K.

5.4 Summary

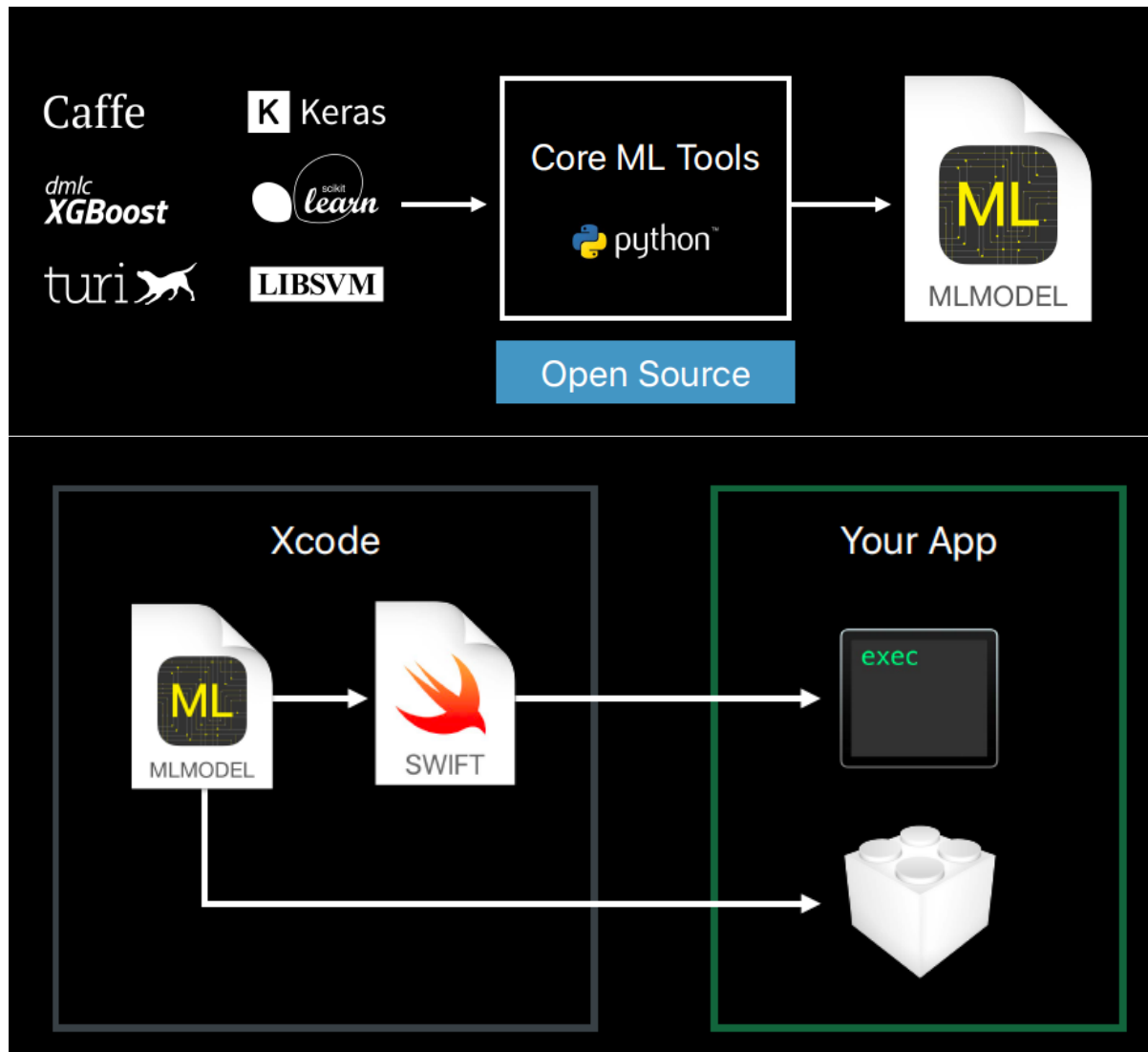
In this chapter, we evaluate the actual speed of different efficient models on mobile phones and the speed of several one-stage object detection models on Intel CPU and NVIDIA GPU. It is shown that our proposed models are able to perform real-time prediction on mobile devices. On iPhone 6s, our proposed classification model can perform 47.5 millisecond per image and our proposed object detection system, named Pelee, can run 17 FPS.

Model	Million MACs	Million Parameters	mAP	Speed (FPS)		
				GPU		CPU
				Titan X	1080 Ti	Intel i7
YOLOv2-544 [31]	28320	57.97	78.6	40	43.5	0.4
RUN300 [24]	31510	27.13	78.6	41.8 ¹	77	0.4
DSSD [4]	-	-	78.6	11.8 ¹	-	-
DSOD300 [34] (64-192-48-1)	15120	14.71	77.7	17.4	31.6	0.6
SSD300 [26]	31370	26.28	77.5	54.5 ¹	86.2	0.4
YOLOv2-416 [31]	16560	57.97	76.8	67	62.5	0.6
DSOD300 [34] (64-64-16-1)	5320	5.83	73.6	-	43.5	1.3
YOLOv2-288 [31]	7940	57.96	69.0	91	92.0	1.0
SSD+MobileNet [15]	1150	5.77	68	-	24 (212 ²)	6.1
Tiny-YOLOv2 [31]	3490	15.86	57.1	207	119.5	2.4
Pelee	1210	5.43	70.9	-	117.5	6.7

Notes:

1. From the paper of "Residual Features and Unified Prediction Network for Single Stage Detection" [24]
2. A huge difference exists in the results between running on cuDNN v5.0 and running on cuDNN v7.0.
SSD+MobileNet runs 212 FPS on cuDNN v7.0.

Table 5.3: Benchmark of one-stage detector on VOC 2007



From <https://developer.apple.com/videos/play/wwdc2017/703/>

Figure 5.1: Workflow of CoreML

Chapter 6

Conclusion and Future Work

6.1 Conclusion

On-device AI enjoys a number of advantages over cloud-based services, which include better privacy protection and enabling the devices to provide reliable execution even without a network connection. With day-to-day development of hardware and efficient CNN architectures, the performance of on-device AI is getting better and better.

Computing power is a major factor that limits CNN used in embedded devices. Our experiments show that by combining efficient architecture design with mobile GPU and hardware-specified optimized runtime libraries, image classification and object detection are able to perform real-time prediction. On iPhone 6s, our proposed image classification model, named PeleeNet, can perform 47.5 millisecond per image and the object detection model, named Pelee, can run 17 FPS with high accuracy.

Model size is another major impediment to the CNN deployed to embedded devices. Based on DenseNet, whose special connectivity pattern can achieve the powerful expression ability with fewer parameters than the pattern of other architectures, our proposed PeleeNet is only 66% the size of MobileNet and only 1/49 the size of VGG but with a higher accuracy than MobileNet and VGG. It is even smaller in model size than the deep compressed [8] VGG model with magnitude faster speed. If combined with the deep compression approach, the model size can be further compressed.

Depthwise separable convolution is not the only way to build an efficient model. Instead of using depthwise separable convolution, our proposed model is built with conventional convolution and can effectively improve the feature express ability when combined with a variant DenseNet architecture. In particular, it excels at fine-grained object recognition. By using the

methods of two-way dense layer, stem block, dynamic channels of bottleneck and removing compression factor, our proposed model, called PeleeNet achieves 80.03% of top-1 accuracy on Stanford Dogs not only much higher than MobileNet, but also higher than ResNet50 and DenseNet-121 with only 18.6% of the computational cost of DenseNet-121 and 13.7% of the computational cost of ResNet50.

Our proposed improvements on SSD effectively balance detection speed and accuracy. By combining a set of selected feature maps, residual prediction block and small size of kernels for prediction, Pelee, our proposed object detection system, achieves 70.9% mAP at 17 FPS speed on an iPhone 6s, outperforming TinyYOLOv2 with higher accuracy, faster speed and 2.92 times smaller model size.

The theoretical computational cost (MACs) can only be used as the design reference but cannot be used for the guidance of the actual speed. Experiments on both server and mobile device show that shallow and wide models demonstrate the actual faster inference speed than narrow and deep models, despite the higher theoretical computational cost. In practice, our design choices should be based on the parallel computing ability of the hardware.

6.2 Future Work

This study provides insightful implications for future model acceleration and compression methods. We can combine PeleeNet with other model acceleration approach and compression methods. For example, there is a possibility to use weight pruning or network binarization in future models to test the experiment results.

This study establishes an excellent example for exploring optimization methods to improve the accuracy of object detection system. Future research can be conducted to explore alternative optimization methods (e.g. knowledge distilling and focal loss) to further improve detection performance.

This study also provides implications for specific optimization on iOS and Android. We can further improve inference speed by working on efficient code implementation and architecture specific code optimization.

Considering the highlighting features of Pelee, that is, fast detection speed, high accuracy and a small model size, we can apply PeleeNet and Pelee to other embedded computer vision tasks, for example, scene text detection and image segmentation.

Bibliography

- [1] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.
- [2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
- [3] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.
- [4] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.
- [5] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.
- [8] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

- [11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [12] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.
- [13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [14] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.
- [15] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016.
- [16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
- [17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.
- [18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.
- [19] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, volume 2, page 1, 2011.
- [20] Kye-Hyeon Kim, Sanghoon Hong, Byungseok Roh, Yeongjae Cheon, and Minje Park. Pvanet: Deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv:1608.08021*, 2016.
- [21] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.
- [22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

- [24] Kyoungmin Lee, Jaeseok Choi, Jisoo Jeong, and Nojun Kwak. Residual features and unified prediction network for single stage detection. *arXiv preprint arXiv:1707.05031*, 2017.
- [25] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.
- [26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [27] Ilya Loshchilov and Frank Hutter. Sgdr: stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.
- [28] Geoff Pleiss, Danlu Chen, Gao Huang, Tongcheng Li, Laurens van der Maaten, and Kilian Q Weinberger. Memory-efficient implementation of densenets. *arXiv preprint arXiv:1707.06990*, 2017.
- [29] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.
- [30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.
- [31] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.
- [32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [33] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.
- [34] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod: Learning deeply supervised object detectors from scratch. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 3, page 7, 2017.
- [35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [36] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017.

- [37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [38] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.
- [39] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.
- [40] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.
- [41] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.

Appendix A

Architecture of DenseNet-41

Layers	Output Size	DenseNet-41 (k=32)
Convolution	112 x 112	3 x 3 conv, stride 2
Pooling	56 x 56	3 x 3 max pool, stride 2
Dense Block (1)	56 x 56	$\begin{pmatrix} 1 \times 1\text{conv} \\ 3 \times 3\text{conv} \end{pmatrix} \times 2$
Transition Layer (1)	56 x 56	1 x 1 conv, stride 1
	28 x 28	2 x 2 average pool, stride 2
Dense Block (2)	28 x 28	$\begin{pmatrix} 1 \times 1\text{conv} \\ 3 \times 3\text{conv} \end{pmatrix} \times 4$
Transition Layer (2)	28 x 28	1 x 1 conv, stride 1
	14 x 14	2 x 2 average pool, stride 2
Dense Block (3)	14 x 14	$\begin{pmatrix} 1 \times 1\text{conv} \\ 3 \times 3\text{conv} \end{pmatrix} \times 6$
Transition Layer (3)	14 x 14	1 x 1 conv, stride 1
	7 x 7	2 x 2 average pool, stride 2
Dense Block (3)	7 x 7	$\begin{pmatrix} 1 \times 1\text{conv} \\ 3 \times 3\text{conv} \end{pmatrix} \times 6$
Classification Layer	1 x 1 x 600	7 x 7 global average pool
120D fully-connecte,softmax		

Table A.1: Architecture of DenseNet-41

Appendix B

Merging Batch Normalization Layer with Convolution Layer

Convolution Layer

conv_weight: W **conv_bias:** B

$$conv_layer \leftarrow W \circledast X + B$$

Batch Normalization Layer

bn_mean: μ **bn_variance:** σ **bn_scale:** γ **bn_shift:** β **epsilon:** ϵ

$$\begin{aligned} \mu &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i & \sigma^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2 \\ \hat{x}_i &\leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} & y_i &\leftarrow \gamma \hat{x}_i + \beta \end{aligned}$$

Merging Batch Normalization Layer with Convolution Layer

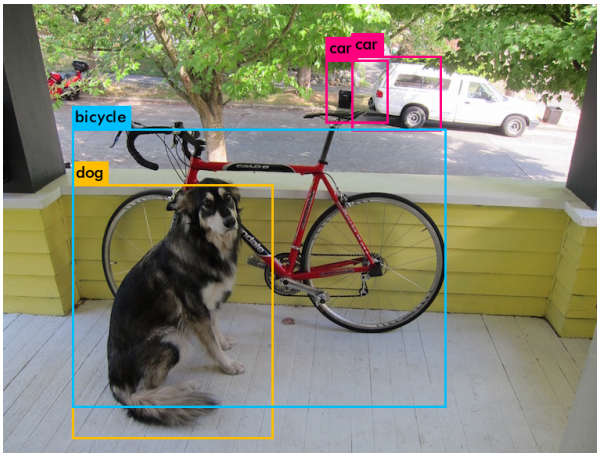
$$\alpha = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}}$$

$$W_{merged} = W \odot \alpha$$

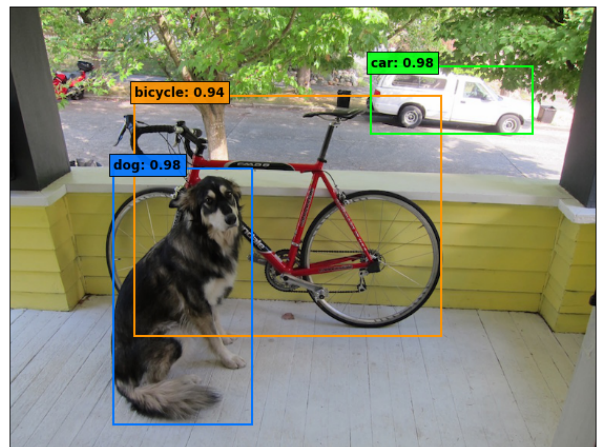
$$B_{merged} = B \times \alpha + (\beta - \mu \times \alpha)$$

Appendix C

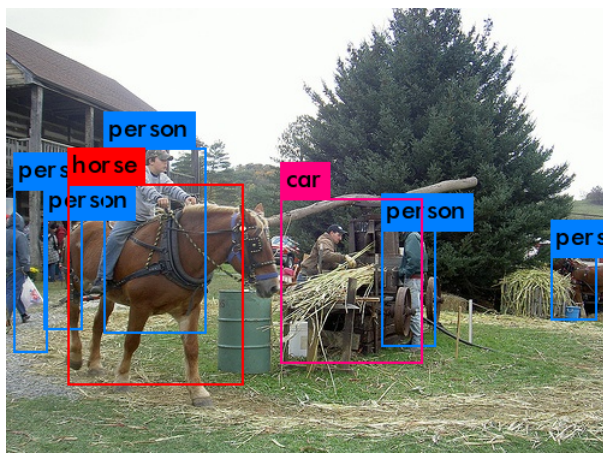
More Detection Examples: TinyYOLOv2 vs. Pelee



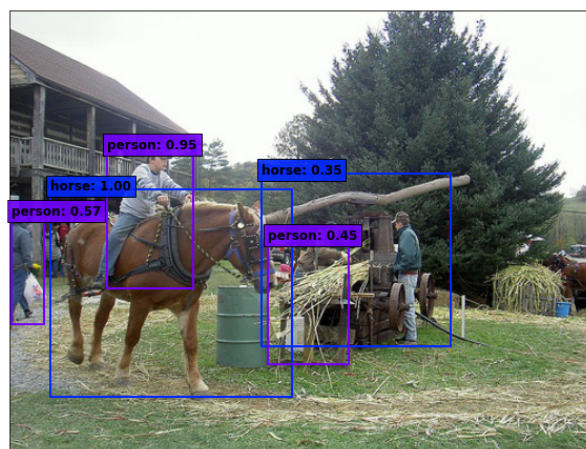
TinyYOLOv2



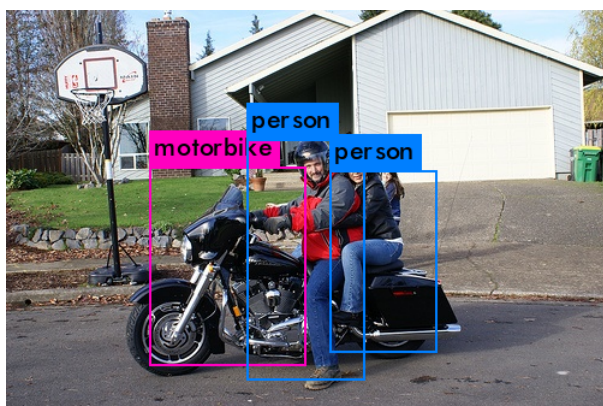
Pelee



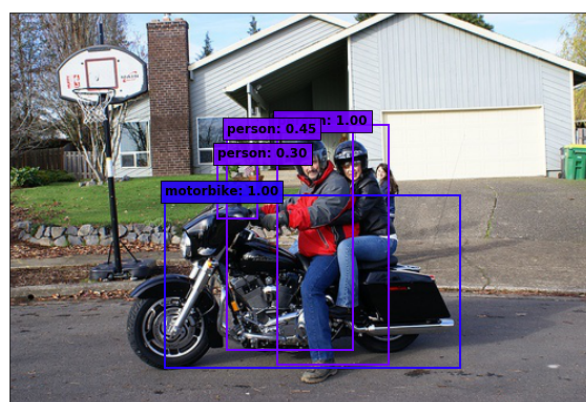
TinyYOLOv2



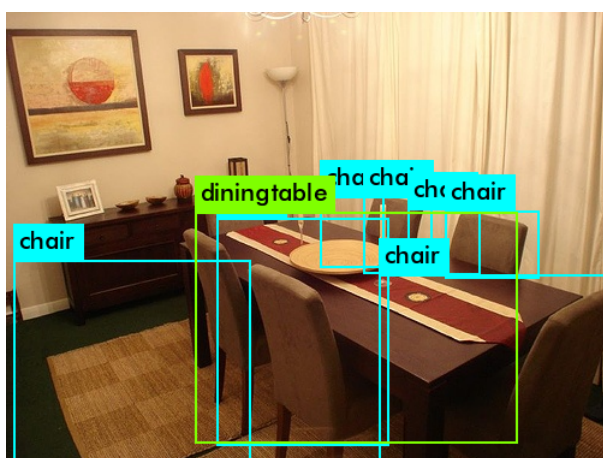
Pelee



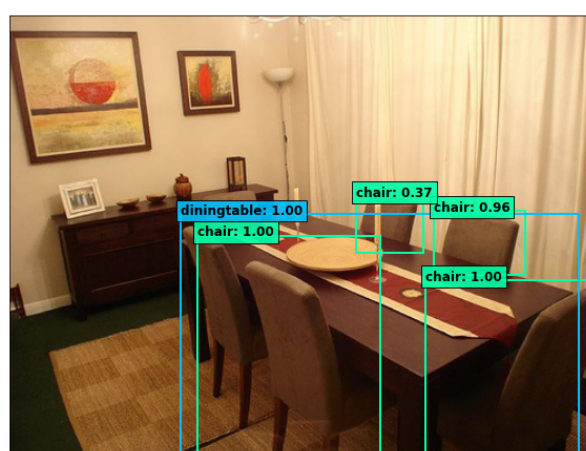
TinyYOLOv2



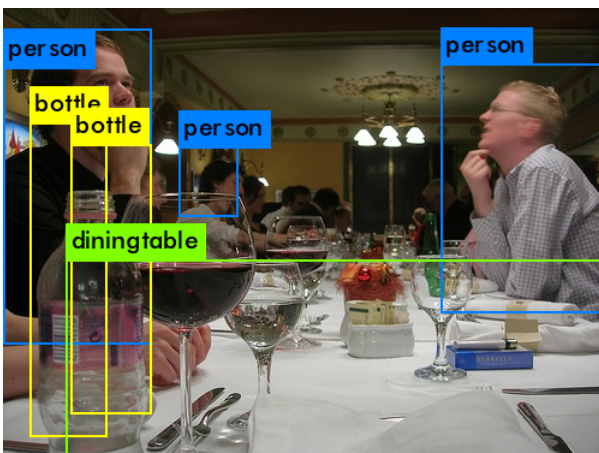
Pelee



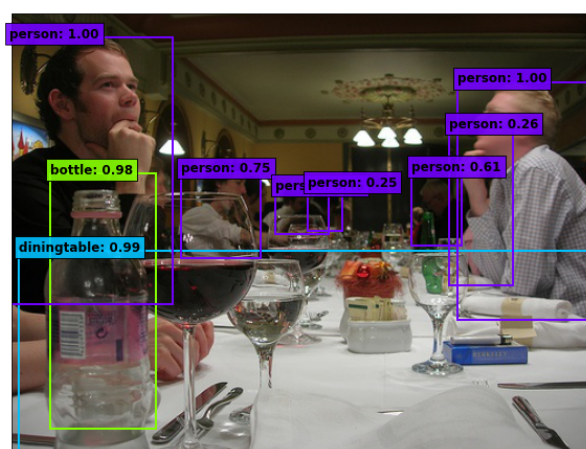
TinyYOLOv2



Pelee



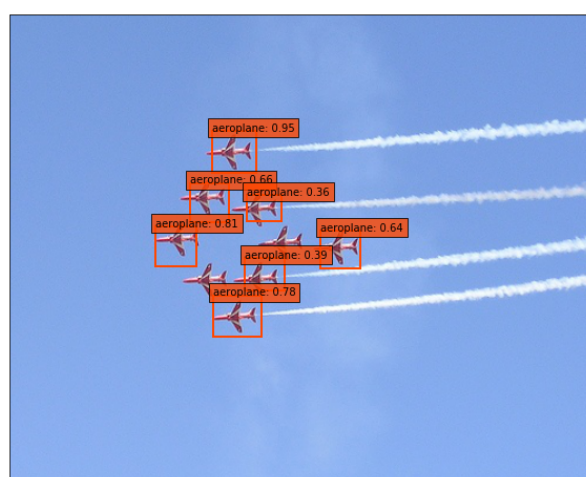
TinyYOLOv2



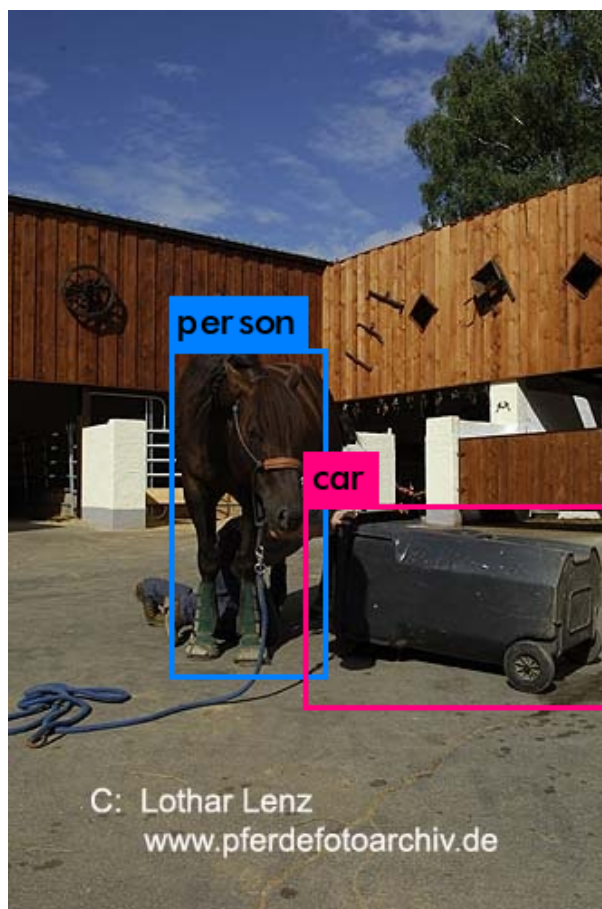
Pelee



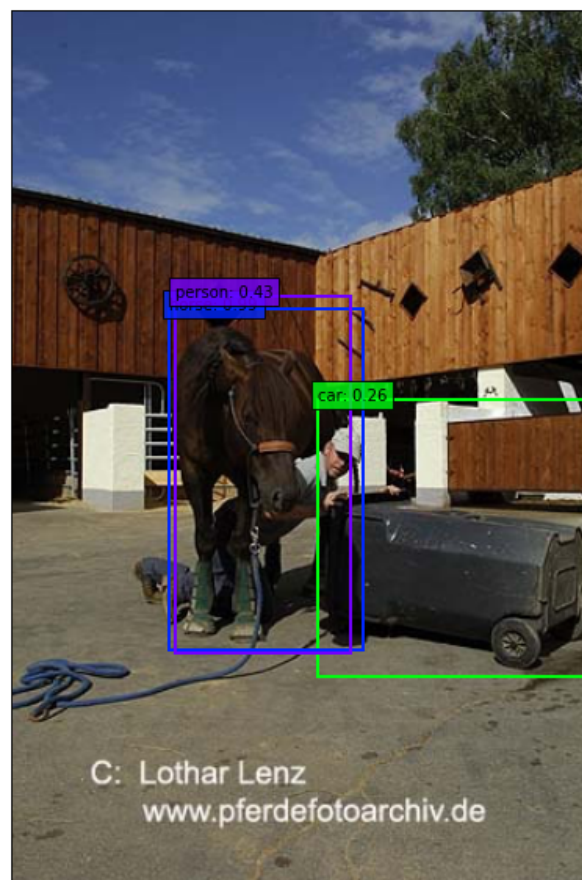
TinyYOLOv2



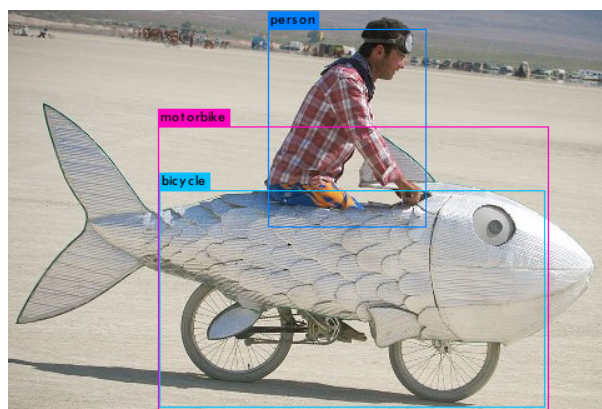
Pelee



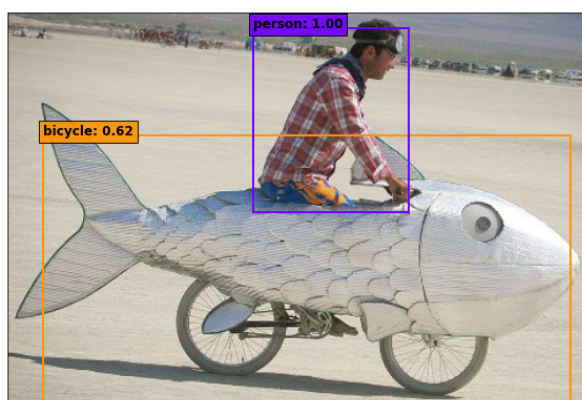
TinyYOLOv2



Pelee



TinyYOLOv2



Pelee

Curriculum Vitae

Name: Jun Wang

Post-Secondary Education and Degrees: Wuhan University
Wuhan, Hubei, China
1997 - 2001 B.Sc. Computer Science

Related Work Experience: Teaching Assistant
The University of Western Ontario
2016 - 2017
Research Assistant
The University of Western Ontario
2015 - 2018