



THE UNIVERSITY
of ADELAIDE

Efficient Fully-Convolutional Networks for Image Perception

Hao CHEN

A thesis submitted for the degree of
DOCTOR OF PHILOSOPHY
The University of Adelaide

February 4, 2021

Contents

Abstract	xi
Declaration of Authorship	xiii
Acknowledgements	xv
1 Introduction	1
1.1 Motivation	2
1.2 Contribution and Thesis Outline	3
2 Background	7
2.1 2D Perception Tasks	7
2.1.1 Static semantic segmentation	7
2.1.2 Dynamic semantic segmentation	7
2.1.3 Object Detection	8
2.1.4 Instance Segmentation	9
2.2 Neural Architecture Search	10
2.2.1 Differentiable NAS	10
2.2.2 Reinforcement Learning Based Methods	11
2.2.3 NAS for perception tasks	11
3 Smooth and Aligned Features for Real-Time Instance Segmentation	13
3.1 Introduction	13
3.2 Background	13
3.3 Methods	17
3.3.1 Aligned Mask Predictors	18
3.3.2 Smooth Features	20
3.4 Two-Stage Interpretation	22
3.5 Experiments	23
3.5.1 Ablation Experiments	24
3.5.2 Two-Stage Version	28
3.5.3 Main Results	28

3.6	Conclusion	29
4	BlendMask: Fully-Convolutional Framework for Instance Segmentation	31
4.1	Introduction	31
4.2	Background	32
4.3	Methodology	36
4.3.1	Overall pipeline	36
4.3.2	Blender module	37
4.3.3	Configurations and baselines	38
4.3.4	Semantics encoded in learned bases and attentions	38
4.4	Experiments	39
4.4.1	Ablation experiments	40
4.4.2	Main result	43
4.4.3	Discussions	46
4.4.4	Panoptic Segmentation	47
4.4.5	More Qualitative Results	48
4.4.6	Evaluating on LVIS annotations	48
4.5	Conclusion	49
5	DR1Mask: Segmenting Background for Free with Rank-1 Dynamic Convolution	53
5.1	Introduction	53
5.2	Background	54
5.3	Preliminaries	56
5.4	DR1Mask: Unified Panoptic Segmentation Network	57
5.4.1	Dynamic Rank-1 Convolution	57
5.4.2	DR1Mask for Instance Segmentation	58
	Top Layer	60
	DR1Basis	60
	Instance Prediction Module	61
5.4.3	Unified Panoptic Segmentation Training	64
5.5	Experiments	65
5.5.1	Dataset and Implementation Details	65
5.5.2	Ablation Experiments	65
5.5.3	Main Results	67
5.5.4	Conclusions	68

6	Dynamic Cell Search for Semantic Video Segmentation	71
6.1	Introduction	71
6.2	Background	71
6.3	Methodology	74
6.3.1	Input space	74
6.3.2	Search space	75
6.3.3	Finding optimal architectures	76
6.4	Experiments	77
6.4.1	Search	77
6.4.2	End-to-end Training	79
6.4.3	Details of Discovered Architectures	82
6.5	Discussion & Conclusions	84
7	NAS-FCOS: Searching Decoder for Object Detection	85
7.1	Introduction	85
7.2	Background	85
7.3	Methodology	87
7.3.1	Problem Formulation	87
7.3.2	Search Space	88
	FPN Search Space	89
	Prediction Head Search Space	90
	Searching for Head Weight Sharing	90
7.3.3	Search Strategy	91
7.4	Experiments	91
7.4.1	Implementation Details	91
	Searching Phase	91
	Full Training Phase	92
7.4.2	Search Results	94
7.4.3	Ablation Study	97
	Design of Reinforcement Learning Reward	97
	Effectiveness of Search Space	99
	Impact of Deformable Convolution	100
7.5	Conclusion	100
8	Conclusion	101
A	Details for Semantic Video Segmentation	103
A.1	Training Details of Static Baseline	103
A.2	Search Space Aggregation Operations	103

List of Figures

3.1	An illustration of the tower for dense instance segmentation . . .	17
3.2	Offsets computation and feature sampling for RoIPred	19
3.3	Different choices of mask predictors with three convolutions as mask tower	20
3.4	Two-stage implementation of RoIPred	23
4.1	Blending process	32
4.2	BlendMask pipeline	36
4.3	Detailed view of learned bases and attentions	39
4.4	Detailed comparison with other methods	50
4.5	BlendMask qualitative results	51
5.1	Dynamic rank-1 convolution	59
5.2	DR1Mask pipeline	59
5.3	Diagram of DR1Basis	61
5.4	An example of the position sensitive attention tensor	62
5.5	Factored attention components	63
6.1	Network structure for video segmentation	73
6.2	<i>arch2</i> [53]. ‘ <i>gap</i> ’ stands for global average pooling.	74
6.3	Average search metrics on CamVid and CityScapes datasets.	78
6.4	Average sampling proportion of operations	80
6.5	Inference results on the test set of CamVid.	81
6.6	Inference results on the validation set of CityScapes.	82
6.7	Visualisation of the discovered cells	83
7.1	An example of our NAS-FCOS decoder	89
7.2	Reward progression on the proxy task	93
7.3	The discovered FPN structure	94
7.4	Our discovered Head structure.	95
7.5	The trend of head weight sharing during search	95
7.6	Correlation between the search reward and final performance	97

7.7	Diagram of the relationship between FLOPs and AP with different backbones	98
7.8	Diagram of the relationship between parameters and AP with different backbones	98
7.9	Comparison of two different RL reward designs	99

List of Tables

1.1	Thesis contributions	4
3.1	Performance of box-aligned predictor	24
3.2	SmalMask kernel sizes difference	25
3.3	Box supervision schedules comparison	25
3.4	Comparison of level selections and sizes of interest	26
3.5	Improvements of anti-aliased downsampling	26
3.6	SmalMask two-stage results	26
3.7	SmalMask main results	27
3.8	SmalMask real-time results	27
4.1	Comparison of different strategies for merging top and bottom modules	40
4.2	Performances with different top/bottom resolutions	41
4.3	Performance with different number of basis	41
4.4	Performance with different bottom feature locations	42
4.5	Performance with different top interpolations	42
4.6	Performance with different bottom alignment	43
4.7	Other improvements	43
4.8	BlendMask main results	44
4.9	BlendMask real-time results	45
4.10	Panoptic segmentation results	47
4.11	Comparison with PointRend	49
5.1	Instance segmentation results with the dynamic factors removed.	66
5.2	Panoptic segmentation results with the dynamic factors removed.	66
5.3	Instance segmentation results with the contextual information from different positions.	67
5.4	Comparison of different instance prediction modules	67

5.5	Comparison of different padding strategies for panoptic segmentation. The baseline method is padding to 32x, divisibility of C5 from ResNet. Padding to 128x is for the divisibility of DR1Basis. 4 w/ aligned is padding the input size to 4x and applying our aligned upsampling strategy.	68
5.6	Comparison of different channel widths in DR1Basis for panoptic segmentation	68
5.7	Position sensitive attention for panoptic segmentation	69
5.8	DR1Mask instance segmentation results	69
5.9	DR1Mask panoptic segmentation results	69
6.1	Description of operations used in the search process.	75
6.2	Description of aggregation operations used in the search process.	76
6.3	Quantitative results on the test set of CamVid	81
6.4	Comparison with other video segmentation approaches on the val set of CityScapes	82
6.5	Number of parameters and average runtime (RT) comparison	83
7.1	Unary operations used in the search process.	89
7.2	Results on test-dev set of MS COCO	93
7.3	Comparison with other NAS methods	96
7.4	Comparisons between APs obtained under different search space with ResNet-50 backbone.	99

University of Adelaide

Abstract

Efficient Fully-Convolutional Networks for Image Perception

by Hao CHEN

Neural architecture search is widely applied to design networks to outperform manually designed architectures. However, it is not trivial to be directly applied to challenging perception tasks such as object detection since previous methods often rely on manually designed complex operations such as RoI pooling and RCNN heads. Thus, we look for universal fully-convolutional representations for perception tasks, which are easy to optimise and deploy because of their simple structures. They perform well on dense prediction tasks such as semantic segmentation, where the networks consist of a backbone module for visual feature extraction and a task-specific module for result generation. Designing the task-specific modules helps us understand how these networks tackle perception tasks and is also crucial for performance and efficiency improvements. However, fully-convolutional networks fall behind two-stage approaches on instance-level tasks such as object detection and instance segmentation. To solve this problem, we focus on designing fully-convolutional frameworks for instance detection tasks and study the task-specific structures and improve their performance by devising efficient neural architecture search algorithms. Our approach starts by designing fully-convolutional models for instance detection tasks. With deformable convolution, we tackle the local-incoherence problem for top-down instance segmentation, resulting in a fully-convolutional model with equivalent expressiveness as a typical two-stage model. We also propose BlendMask, a fully-convolutional instance segmentation network that is faster and more accurate than the state-of-the-art two-stage models. Then we demonstrate the benefit of having uniform representation by designing the first a panoptic segmentation network solving instance and semantic segmentation with a single branch. Targeting to improve the design of task-specific modules for fully-convolutional perception models, we devised efficient neural architecture search algorithms and applied them to video segmentation and object detection.

Declaration of Authorship

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Hao Chen

September 2020

Acknowledgements

First and foremost, I would like to thank my Ph.D. advisor, Professor Chunhua Shen. I have been very fortunate to work with him for the three years in Adelaide. He guided me through my journey in academic research. His expertise was invaluable for me to formulate my research directions and methodology. As a renowned professor, he is remarkably hard-working and humble. Chunhua's research foresight, technical depth, and commitment to the students is a valuable treasure for me.

I would also thank Professor Peng Wang from NWPU, previously a postdoc in our school. I want to thank him for his patient support and for all of the opportunities I was given to further my research.

I also thank all my closest collaborators during these years. Vladimir Nekrasov, Zhi Tian, Yuliang Liu, Ning Wang, Yang Gao, Kunyang Sun and Haokui Zhang. Vlad is the most efficient and disciplined researcher I've met of my age. Zhi is known for his Midas touch that everything he works on turns to be very influential. As one of the closest witnesser, I profoundly understand that the most critical reason of his success is his great attention to the details. Yuliang seizes every moment with me for academic discussions, which is my favorite small talk topic as well. It was a great honor to be able to work with you all. You are my dear friends. Every discussion is inspiring and pleasant. These rewarding and fun collaboration experiences have been a great motivation for me to work in the academic field. Good luck to you all.

I give my sincere thanks to all my other lab mates. They are the most diligent group of people I've met. Thank you for your contribution to the productive yet stress-free work environment. I enjoy everyday working around you guys.

I want to thank my wife for her unconditional support and emotional companionship. It was her proposal for me to apply for this PhD position. During these years, she managed to spend all her leaves from work here in Australia with me. I can accomplish what I did without her. And I give my sincere thanks to my parents, you have always been a great positive influence to me.

Finally, I would like to thank the funding support from the school of computer science.

Chapter 1

Introduction

Research in 2D machine perception involves designing algorithms for understanding scenes represented by 2D images, such as detecting and segmenting general objects. In recent years, deep neural network based methods have been the dominant approach because of its high expressiveness and the emergence of massive datasets. Neural networks for perceptions typically consist of two parts, a backbone module for visual feature extraction and a task-specific module for predicting the final results. Many researches focus on the design of task-specific modules, which is crucial for both the understanding of the tasks and performance/efficiency improvements.

These perception models have been playing a crucial part in many applications such as photography, augmented reality and autonomous driving and been widely embedded in many devices such as mobile phone and drones. They have to be computation and power efficient to be approachable to these low-end devices and have to be compact enough to fit into the storage, which requires great effort in the architecture design and acceleration.

Recently, neural architecture search (NAS) methods have attracted much attention and outperformed manually designed architectures. However, many of them focus on classification and cannot be directly applied to higher-level perception tasks. An additional task-specific head following the networks found for classification is required, to process the general visual features and generate the final predictions. Sometimes, these algorithms fail to generalize to the target perception task because they tend to overfit the simpler proxy task with smaller dataset where search is conducted. In addition, it is intuitive that the best models for classification may not be the best for higher-level perception tasks because perceptions tasks often require richer spatial information and representations with higher resolution.

Comparing to classification tasks, NAS on perception tasks such as semantic segmentation and object detection is not widely studied. Many perception tasks require more complicated network structures whose search space is difficult to

design. Particularly, these more complex tasks requires more data and longer training schedules, which makes the searching process inefficient.

Albeit being challenging, we find this area worth further studying for both empirical and theoretical reasons. In this thesis, we introduce our works on designing fully-convolutional frameworks for complex perception tasks e.g. instance segmentation and efficient NAS algorithms to optimize the structure of these perception networks. To make search possible for perception tasks, we design unified powerful yet simple representation for instance-level tasks suitable for the search space design. We further design acceleration strategies to make to searching process efficient on tasks such as segmentation and object detection.

1.1 Motivation

Even though many perception tasks have a similar goal of classifying the object in scenes, models for different tasks often have drastically different architectures. This leaves great difficulties if we attempt to merge the models in multi-task setting to share their intermediate features and reduce computational cost. For example, panoptic segmentation [1] is a task that combines semantic segmentation and instance segmentation, where each pixel is assigned to a class label and different instances have different labels. If we rely on a two-stage method for instance segmentation, we have to use a separate decoder for semantic segmentation, leaving no room for feature sharing.

Instead, using a fully-convolutional approach for perception task not only provides great flexibility for merging different heads in multi-task learning but also leaves great room for architecture design optimisation. We looking for universal fully-convolutional encoder-decoder framework for various perception tasks and design automated algorithms targeting task-specific decoder architecture optimisation.

Specifically, we formulate instance-level perception models in the form of fully-convolutional networks, which previous state-of-the-art is two-stage methods such as Mask R-CNN [2]. Reduce the expressiveness gap [3] and improve the performance to surpass their two-stage counterparts in the field of object detection and instance segmentation. These models have been widely adopted by the researchers and have been established as new strong baselines.

Based on these frameworks, we design NAS algorithms to search for compact and powerful task-specific modules, which is not trivial. Usually, NAS algorithms require expensive computation cost since evaluating each sample takes

a complete training. We proposed an acceleration strategy, namely RL-NAS [4] tailored for fully-convolutional sub-network search which speeds up the whole process dramatically. Based on RL-NAS, we are able to search for dynamic cells for video segmentation [5] and neck/head structures for object detection [6]. In addition, for different tasks, the search space and training strategies also need to be carefully chosen. Among the first researchers to explore NAS for perception tasks, we are happy to provide our empirical findings to the field as some guidelines.

1.2 Contribution and Thesis Outline

We design efficient models for various perception tasks with the help of neural architecture search. This thesis proposes a fully-convolutional framework for instance-level tasks and efficient searching algorithms for the task-specific modules, as listed in Table 1.1. The contributions of this thesis are:

- A comprehensive study of top-down fully-convolutional instance segmentation network. To address its major problem, we devise an intermediate model, called SmalMask, that reduces the gap between one-shot and two-stage by improving the local-coherence of mask prediction module with deformable convolutions.
- A fast and accurate fully-convolutional instance segmentation framework, called BlendMask, that enjoys the advantages of both previous top-down and bottom-up approaches, which surpasses its two-stage counterpart Mask R-CNN particularly in the mask quality and inference time consistency.
- The first unified fully-convolutional panoptic segmentation framework, called DR1Mask, that is twice more efficient than previous methods because a single very compact module is used for both instance segmentation and semantic segmentation.
- An reinforcement learning based neural architecture search algorithm, RL-NAS, for semantic segmentation, which focuses on finding efficient task-specific sub modules in fully-convolutional networks. In particular, we apply this algorithm to study the effect of dynamic cells in video segmentation networks.
- Equipped with this algorithm, we discovered a state-of-the-art object detection architecture, called NAS-FCOS, which is made possible by the

fully-convolutional one-stage framework, FCOS (Fully-Convolutional One-Shot Object Detection).

Perception Tasks	Conventional (complex)	Fully-Convolutional (easy to optimize)	NAS (efficient)
Semantic Segmentation	-	DeepLab [7]	RL-NAS [4], Chapter 6 [5]
Object Detection	Faster R-CNN [8]	<i>FCOS</i> [9]	Chapter 7 [6]
Instance Segmentation	Mask R-CNN [2]	Chapter 3 [3],4 [10]	Future work
Panoptic Segmentation	Panoptic FPN [1]	Chapter 5 [11]	Future work

TABLE 1.1. **Thesis contributions:** We propose simple and easy to optimize fully-convolutional frameworks for instance segmentation which surpass previous state-of-the-art conventional methods and NAS algorithms targeting on dynamic structure in video segmentation and object detection. I have contributed to the work in italic. Works in bold font are the ones of which I am the (co-) first author.

Chapter 2 provides the background for problems and techniques we use in this thesis, i.e. deep learning based 2D machine perception tasks and neural architecture search algorithms. We survey related fully-convolutional approaches for perception tasks including semantic segmentation, object detection and instance segmentation and algorithms for automatic network design. Among them, two of my collaborative works, RL-NAS [4] and FCOS [9] are the most important foundation for the work in this thesis.

Chapter 3 describes one major shortcoming for top-down instance segmentation models, i.e. local feature alignment and our proposed fix with deformable convolution. Resulting in a fully-convolutional structure, SmalMask, that has equivalent expressive power comparing to the two-stage counterpart. The content of this chapter is based primarily on [3].

Chapter 4 presents a better fully-convolutional framework for instance segmentation called BlendMask. It preserves the local-coherence with a bottom-up module and represent the instance-wise information with a top-down module. BlendMask breaks the mask resolution limitation of two-stage methods while being more efficient. The content of this chapter is based primarily on [10].

Chapter 5 demonstrates the benefit of fully-convolutional framework for perception task in multi-task learning. By designing a more efficient dynamic module, we present the first unified panoptic segmentation network, DR1Mask, that segments thing and stuff from a single feature. DR1Mask is two times

faster than previous SOTA approaches. The content of this chapter is based primarily on [11].

Chapter 6 describes an example of applying RL-NAS to video semantic segmentation. We study the choice of temporal propagation modules, aka dynamic cells empirically, by quantifying the distribution of such operations during the process of neural architecture search. In addition, we discovered an efficient architecture for video segmentation. The content of this chapter is based primarily on [5].

Chapter 7 presents our work of improving FCOS-based object detection modules with RL-NAS. We are able to efficiently search a top-performing detection architecture within 4 days using 8 V100 GPUs. The discovered architecture surpasses state-of-the-art object detection models by 1.5 to 3.5 points in AP on the COCO dataset. The content of this chapter is based primarily on [6].

In **Chapter 8** we summarize this thesis and discuss future directions for fully-convolutional perception networks.

Chapter 2

Background

2.1 2D Perception Tasks

2D perception tasks are targeted to interpret scenes similar to humans use their eyes. The tasks can vary in their level of concept and granularity. On one end there is dense prediction tasks which assign a prediction target to each pixel. On the other end there is instance-level tasks such as object detection which only requires sparse output signals related to the whole instances. Many tasks lie in between, requiring information from both sides, such as instance segmentation and human pose estimation. In this section, I will go through the basics and introduce recent approaches to these tasks.

2.1.1 Static semantic segmentation

Most recent approaches in static semantic segmentation have been exploiting fully convolutional neural networks [12]. Typical methods are based either on the encoder-decoder structure with skip-connections [12], [13], dilated convolutional layers [14]–[16], or the combination of the above [17]. Per-frame instantiations of these networks are usually computationally expensive, hence, several works have considered building light-weight segmentation architectures [18], [19]. Nevertheless, due to the lack of information propagation between frames, these networks perform poorly on videos and are unable to provide consistent results.

2.1.2 Dynamic semantic segmentation

One of the first lines of work in video segmentation has been built upon the usage of the optical flow [20], in which features extracted from the previous frame are propagated to the current one via warping. This usually results in a slight computational overhead, although as noted by Gadde *et al.* [21] an easily attainable noisy estimate of the optical flow still carries significant benefits.

Nevertheless, the optical flow does not fair well in situations when scenes are undergoing substantial changes with novel objects constantly appearing and multiple occlusions being present. Thus, Jain *et al.* [22] have proposed to combine the optical flow estimate with a relatively cheaper approximation of the current frame using a smaller network. Xu *et al.* [23] have chosen to assign different image regions to two different networks to process: while the first one - deep and slow - works on regions that have significantly changed, the second one - shallow - predicts new features based on the optical flow information. In a similar vein, Nilsson and Sminchisescu [24] have propagated labels from the previous frame at only those pixels where the optical flow estimate is reliable.

A seemingly different approach, proposed by Li *et al.* [25], instead predicts local convolutional kernels based on the low-level representation of the current frame that are applied on the prediction from the previous frame. Importantly, while the current estimate is being used for next frame, a more accurate one is being computed in parallel for future re-use.

In yet another line of work, Chandra *et al.* [26] have adapted Deep Gaussian Random Field [27] to handle temporal information by predicting besides unary and spatial pairwise terms also temporal pairwise terms, efficiently propagating features between frames.

2.1.3 Object Detection

The frameworks of deep neural networks for object detection can be roughly categorized into two types: one-stage detectors [28] and two-stage detectors [2], [8].

Two-stage detection frameworks first generate class-independent region proposals using a region proposal network (RPN), and then classify and refine them using extra detection heads. In spite of achieving top performance, the two-stage methods have noticeable drawbacks: they are computationally expensive and have many hyper-parameters that need to be tuned to fit a specific dataset.

In comparison, the structures of one-stage detectors are much simpler. They directly predict object categories and bounding boxes at each location of feature maps generated by a single CNN backbone.

Note that most state-of-the-art object detectors (including both one-stage detectors [28]–[30] and two-stage detectors [8]) make predictions based on anchor boxes of different scales and aspect ratios at each convolutional feature map location. However, the usage of anchor boxes may lead to high imbalance between object and non-object examples and introduce extra hyper-parameters.

Anchor-free object detection Recent advances in object detection unveil the possibilities of removing bounding box anchors [9], [31]–[34], largely simplifying the detection pipeline. This much simpler design improves the box average precision (AP^{bb}) by 2.7% comparing to its anchor-based counter-part RetinaNet [28]. One possible reason responsible for the improvement is that without the restrictions of predefined anchor shapes, targets are freely matched to prediction features according to their effective receptive field.

2.1.4 Instance Segmentation

Detect-then-segment instance segmentation The dominant instance segmentation paradigms take the two-stage methodology, first detecting the objects and then predicting the foreground masks on each of the proposals. The success of this framework partially is due to the alignment operation, RoIAlign [2], which provides local-coherence for the second-stage RoI heads missing in all one-stage top-down approaches. However, two issues exist in two-stage frameworks. For complicated scenarios with many instances, inference time for two-stage methods is proportional to the number of instances. Furthermore, the resolution for the RoI features and resulting mask is limited. We discuss the second issue in detail in Section 4.4.3.

These problems can be partly solved by replacing a RoI head with a simple crop-and-assemble module. In FCIS, Li *et al.* [35] add a bottom module to a detection network, for predicting position-sensitive score maps shared by all instances. This technique was first used in R-FCN [36] and later improved in MaskLab [37]. Each channel of the k^2 score maps corresponds to one crop of $k \times k$ evenly partitioned grid tiles of the proposal. Each score map represents the likelihood of the pixel belongs to a object and is at a certain relative position. Naturally, a higher resolution for location crops leads to more accurate predictions, but the computation cost also increases quadratically. Moreover, there are special cases where FCIS representation is not sufficient. When two instances share center positions (or any other relative positions), the score map representation on that crop is ambiguous, it is impossible to tell which instance this crop is describing.

In YOLACT [38], an improved approach is used. Instead of using position-controlled tiles, a set of mask coefficients are learned alongside the box predictions. Then this set of coefficients guides the linear combination of cropped bottom mask bases to generate the final mask. Comparing to FCIS, the responsibility for predicting instance-level information is assigned to the top-level. We

argue that using scalar coefficients to encode the instance information is sub-optimal.

Bottom-up Beside the predict-then-segment approaches, it is also possible to segment instances from the bottom up. These methods takes a predict-then-group route. They first learn a instance-aware feature map for each pixel, then they apply some grouping methods to predict instances. A straight-forward grouping algorithm is clustering[39]. Depends on the properties of the learned instance-aware embeddings, it is possible to apply all kinds of grouping methods, such as graph-based algorithms [40]. Some proposal-based models are distantly related, as they perform patch-based grouping guided by detection results. FCIS [35] assembles score map crops from corresponding locations like InstanceFCN [41]; YOLACT [38] performs a weighted sum on the proposal regions of the embeddings. We limit our search to top-down dense instance segmentation for its simplicity.

2.2 Neural Architecture Search

2.2.1 Differentiable NAS

NAS methods aim to find high-performing architectures in an automated way. However, it has not been widely adopted in common deep learning research workflow since it is usually time consuming. We have seen great improvements from 24,000 GPU-days [42] to 0.2 GPU-day [43]. The trick is to first construct a supernet containing the complete search space and train the candidates all at once with bi-level optimization and efficient weight sharing [44], [45]. But the large memory allocation and difficulties in approximated optimization prohibit the search for more complex structures.

Recently researchers [46]–[48] propose to apply single-path training to reduce the bias introduced by approximation and model simplification of the supernet. DetNAS [49] follows this idea to search for an efficient object detection architecture. One limitation of the single-path approach is that the search space is restricted to a sequential structure. Single-path sampling and straight through estimate of the weight gradients introduce large variance to the optimization process and prohibit the search for more complex structures under this framework. Within this very simple search space, NAS algorithms can only make trivial decisions like kernel sizes for manually designed modules.

2.2.2 Reinforcement Learning Based Methods

Reinforcement learning (RL) [50] studies sequential decision making processes to maximize the cumulative rewards. It can be applied to perform gradient free optimisation. Here, we consider the reinforcement learning-based approach [51], where a separate recurrent neural network (controller) outputs a sequence of tokens describing an architecture that should provide highest score on the holdout validation set.

To speed up reward evaluation of RL-based NAS, the work of [4] proposes to use progressive tasks and other training acceleration methods. By caching the encoder features, they are able to train semantic segmentation decoders with very large batch sizes very efficiently. In the sequel of this paper, we refer to this technique as fast decoder adaptation.

2.2.3 NAS for perception tasks

Two results in static segmentation are worth mentioning: Chen *et al.* [52] used a random search to find a single set of operations (so-called ‘cell’) on the top of the DeepLab architecture [16], while Nekrasov *et al.* [53] exploited RL to find a cell together with the topological structure of the encoder-decoder type of architecture.

We borrow one of the architectures found by Nekrasov *et al.* as our static baseline, and extend their NAS approach for video segmentation. Since we are only searching for the dynamic component that connects different instantiations of the already pre-trained static segmentation network, we are able to train and evaluate each candidate in a short amount of time, the trait that is extremely important for all NAS methods.

However, directly applying this technique to object detection tasks does not enjoy similar speed boost, because they are either not in using a fully-convolutional model [54] or require complicated post processing that are not scalable with the batch size [28].

Object detection models are different from single-path image classification networks in their way of merging multi-level features and distributing the task to parallel prediction heads. Feature pyramid networks (FPNs) [1], [54]–[57], designed to handle this job, plays an important role in modern object detection models. NAS-FPN [55] targets on searching for an FPN alternative based on one-stage framework RetinaNet [28]. Feature pyramid architectures are sampled with a recurrent neural network (RNN) controller. The RNN controller

is trained with reinforcement learning (RL). However, the search is very time-consuming even though a proxy task with ResNet-10 backbone is trained to evaluate each architecture.

Since all these three kinds of research ([49], [55] and ours) focus on object detection framework, we demonstrate the differences among them that *DetNAS [49] aims to search for the designs of better backbones, while NAS-FPN [55] searches the FPN structure, and our search space contains both FPN and head structure.*

Statement of Authorship

Title of Paper	Smooth and Aligned Features for Real-Time Instance Segmentation
Publication Status	<input type="checkbox"/> Published <input type="checkbox"/> Accepted for Publication <input checked="" type="checkbox"/> Submitted for Publication <input type="checkbox"/> Unpublished and Unsubmitted work written in manuscript style
Publication Details	Prepared for CVPR 2020 submission

Principal Author

Name of Principal Author (Candidate)	Hao Chen
Contribution to the Paper	Experiment framework design and implementation, experiment conduction, writing method part of the paper.
Overall percentage (%)	70%
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.
Signature	<hr/>
Date	02/10/2020

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

- i. the candidate's stated contribution to the publication is accurate (as detailed above);
- ii. permission is granted for the candidate to include the publication in the thesis; and
- iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Zhi Tian
Contribution to the Paper	Discussion and writing revision.
Signature	<hr/>
Date	19/10/2020

Name of Co-Author	Chunhua Shen
Contribution to the Paper	Discussion and writing revision.
Signature	<hr/>
Date	19/10/2020

Chapter 3

Smooth and Aligned Features for Real-Time Instance Segmentation

3.1 Introduction

Recently, fully convolutional object detection methods have drawn much attention because of their simple yet neat structures. Unifying the framework of instance segmentation with fully convolutional object detectors is of great interest, which can lead to easier analysis and more flexible network design. In this chapter we draw connections between fully convolutional framework and the better engineered two-stage approach, Mask R-CNN. To reduce the performance gap between these two, we investigate the effect of local-coherence and positional information on instance segmentation. Where we carry out a thorough evaluation of current adaptive receptive field operations and anti-aliasing samplers. We propose a predictor module named RoIPred, which outperforms other dense mask predictors. Furthermore, we reduce redundant computation of the predictor module by casting it to a two-stage representation. The resulting model, SmalMask2, is a simple instance segmentation framework which enjoys the advantages of both sides, scalable design, efficient computation and improved performance. Our real-time model SmalMask2-RT is 2.7 mAP better than the YOLACT model while having similar inference time, and the full-fledged model SmalMask2+ attains 0.7 mAP higher while being $5\times$ more efficient than its TensorMask counterpart.

3.2 Background

The top performing object detectors and instance segmenters follow a two-stage paradigm. A region proposal network (RPN) first predicts proposal candidates, or regions of interest (RoIs). For the second stage, a group of light-weight

subnetworks, namely ‘heads’ refines the features inside RoIs and generate predictions [8]. In contrast, a dense object predictor/segmenter takes an end-to-end paradigm. It predicts the targets directly with a fully convolutional network [9], [28], [58]. Some of the most successful examples of the two frameworks shares some common points: they often use a feature pyramid network (FPN) to generate feature maps of different sizes to deal with objects of different scales.

Two-stage object detectors were considered to be slower than the dense detectors because of the per-region subnetwork computation [36] and more complex post-processing. Here we want to clarify further: it is not always true. In practice, with the typical number of foreground proposals being around 1000, modern two-stage models such as Faster R-CNN [8] can be both faster and more accurate than a typical one-stage model such as RetinaNet [28]¹, since the head computation is actually efficient for simple bounding box regression. Some one-stage methods move the head computation to a parallel tower after the feature pyramid, which can be more costly because the features are at a higher resolution than the head RoIs. There is a speed-accuracy trade-off by balancing the tower *vs.* head computation.

When it comes to instance segmentation, the head *vs.* tower speed-accuracy trade-off leans towards the head much more. As the task for each instance becomes more difficult, it is more efficient to focus on RoIs and make the head more complex. Otherwise, a large portion of computation could be wasted on background locations. But this does not put the effort of studying dense instance segmenters in vain. First, they are easier to be incorporated into modern one-stage object detectors[9], which are proved to be competitive in efficiency and performance. Second, the computation cost for the fully convolutional paradigm does not scale with the number of foreground objects as two-stage methods do. Last but not least, they provide a nice baseline for studying the dense feature learning in convolutional neural networks, which is our focus in this chapter.

DeepMask [59] is regarded as the first deep dense instance segmenter, which performs a per-pixel instance prediction on the feature maps. A convolution layer predicts mask logits with a fixed size as a flattened vector. This straightforward representation lays down the foundations and leave some challenging problems. The feature map the mask predictor samples are usually either misaligned with the target. This can happen when the feature is not capable of describing the whole instance.

¹Please refer to the model zoo of the official Detectron2 repository: https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md.

Dai *et al.* [41] propose to mitigate the feature misalignment by assembling features from different patches. Chen *et al.* [60] goes one step further by introducing a densely aligned representation for each location of the target instance mask. By aligning every pixel with the high-resolution target mask, their TensorMask is able to achieve performance similar to the two-stage Mask R-CNN. However, using the aligned representation introduces great representation redundancy which slows down the model severely.

Feature pyramids in dense object detectors suffer from position resolution degradation introduced by strided convolutions. For other dense prediction tasks, this can be dealt with by refining the features with an encoder-decoder framework. However, the dense instance mask prediction layer requires larger receptive field but usually has a much smaller sampling density than an RoI pooler (5×5 convolution *vs.* 14×14 RoIAlign). This requires the features to sample on to be more compact as well. In practice, we map targets to higher levels of FPN in dense object detectors than in their two-stage counter-parts. The downside of this is that comparing to lower levels, higher levels lack spatial information.

Because of these existing issues, it is difficult to design a dense instance segmenter as efficient as Mask R-CNN. In TensorMask [60], to match the performance, head computation is increased which makes the running time three times as long as Mask R-CNN.

In this chapter, we try to answer the following two questions:

1. What is the gap between dense instance segmenters and the two-stage approaches?
2. Can we reduce this gap and propose a form of algorithms in between these two which enjoy the advantages from both sides?

Our work is driven by the goal of trying to empower one-stage framework with the following two properties:

Aligned Features It is common in object detection networks to align features according to anchors or proposals. In two-stage frameworks, feature misalignment in RoI sampling [8] can be mitigated with RoIAlign [2]. For one-stage methods, the alignment process is often done with a convolution layer or its variations. RefineDet [61] uses several vanilla convolutions to refine features associated with anchors. Guided anchor instead uses the proposal shape to generate offset field for a deformable convolution layer [62], which is used to align the feature. Chen *et al.* [63] propose RoIConv, which is a deformable convolution with offsets calculated from an RPN network. This operation is

more aligned to the proposal, and is equivalent to a dense RoIAlign followed by a fully connected layer.

We perform an extensive investigate of different feature aligning layers. Our method is different from RoIConv in that we apply the aligned operation directly to the last prediction layer, which we argue helps preserving local-coherence. And we use ground-truth instead of prediction to generate the sampling locations. Experiments show that this increases stability to the local supervisions and improves the final performance.

Smooth Features In signal processing, a low-pass filter is often applied to the signal before downsampling to avoid aliasing. Zhang [64] applied this idea to convolutional neural networks to avoid unstable classification results acting to small image shifts. The conventional strided convolution is replaced by a convolution with stride 1, following a BlurPool operation, which is implemented as a strided blur convolution kernel. We apply this low-pass filter to our backbone network before the features getting downsampled by strided-convolutions.

We first conduct extensive experiments to investigate the above listed problems to search for the answers. Then we then compare our solution directly to its two-stage equivalent and analyze the pros and cons of each method. Our study and experiments proves that *by reducing the head computation, two-stage models can be more efficient than dense models while avoiding their representation issues*. Our result is a simple architecture, SmalMask (SMooth and ALigned Mask Predictor), closely tied to a state-of-the-art one-stage object detector, FCOS [9], by adding only one layer to the top for mask prediction. More specifically, our contributions are:

- A single layer remedy called RoIPred for aligning dense instance segmentation features is devised.
- We apply blurred convolution to reduce sampling alias in strided convolutions.
- Mask prediction quality is further improved.
- We incorporate this solution to both dense and two-stage paradigms, resulting in two models, SmalMask and SmalMask2.
- Learning from the weight-sharing and behavior in the dense model, we propose to use multi-scale poolers to deal with the scale imbalance problem in two-stage models.
- Comparing to previous best performing dense instance segmenter, TensorMask [60], SmalMask is five times faster, achieving on par mask APs.

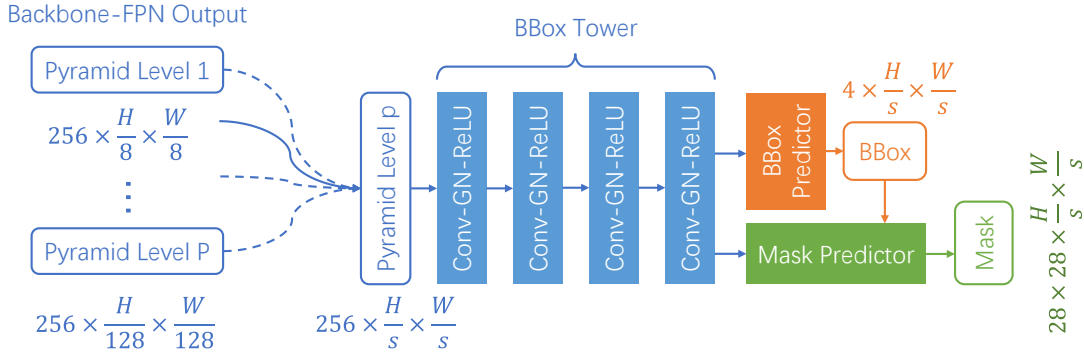


FIGURE 3.1. An illustration of the tower for dense instance segmentation. Blue and orange blocks are the original modules and features of FCOS. For the shared structure, we only add a one-layer mask predictor module (green) on top of the bounding box tower.

3.3 Methods

Our dense instance segmentation model SmalMask can be viewed as a hybrid of FCOS [9] and DeepMask [59].

FCOS uses Feature Pyramid Networks [54] as its backbone structure, using multi-scale features (pyramids) to predict objects of different sizes. This framework provides feature augmentation and normalizes target scales for different levels. This also plays a crucial role in dealing with object overlaps which can potentially create target ambiguity [9].

We add a dense mask prediction module to the feature pyramid of FCOS. While DeepMask generates mask of the same size as the input, we force this module to generate masks of fixed size (28×28), then project the predictions onto the corresponding detection results. Considering the computation efficiency, we design two versions of SmalMask:

- The **separate** SmalMask has an independent mask-tower followed by the mask prediction layer.
- The **shared** SmalMask adds minimal computation overhead to the FCOS towers. The mask prediction layer is added to the box-tower.

In Figure 3.1, we illustrate the mask prediction tower of shared SmalMask. For the separate structure, the mask head instead use another independent tower of four convolutions. In Section 3.3.1, we discuss the design of mask prediction layer to improve local-coherence.

Similar to other dense approaches, our baseline model is vulnerable to the addressed problems:

- **Local-coherence missing:** It is difficult for the dense feature to encode information of the entire mask, which requires some points to be aware of patterns very far from themselves. Moreover, the supervision is invariant to their relative positions inside the box.
- **Representation redundancy:** All locations within the box foreground has to predict the same mask repeatedly.
- **Positional information loss:** The highest level has a resolution of merely about ten pixels, which is very hard to preserve positional fidelity after traditional strided downsampling.

In the following sections, we introduce our fixes to these problems. More specifically, we devise an aligned mask predictor to retain local-coherence of the features before prediction and its two-stage equivalent to avoid representation redundancy. Anti-aliasing and sampling tricks are applied to reduce the downsampling information corruption almost for free.

3.3.1 Aligned Mask Predictors

Different from the feature adaptation layers in object detectors [61], [63], [65], which adjust the features during the tower computation, we want to keep the features aligned, i.e., coherent to their locations. So we directly apply feature alignment to the final prediction layer. We enable the predictor to adjust receptive field according to detections. We compare vanilla convolution to two types of deformable convolutions, DefConv, the one with unsupervised offset and modulations [66] and two other predictors with proposal-guided offsets.

We now introduce RoIPred, our attempt to fix the local-coherence with an aligned mask predictor. RoIPred is a deformable convolution with bounding box guided offsets. For an input feature $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$, the RoIPred generates mask predictions $\mathbf{M} \in \mathbb{R}^{N \times (M \times M) \times H \times W}$ given a set of bounding boxes $\mathbf{B} \in \mathbb{R}^{N \times 4 \times H \times W}$:

$$\mathbf{M} = \text{RoIPred}(\mathbf{X}, \mathbf{B}). \quad (3.1)$$

Given bounding box target or prediction $\mathbf{b} = (l, t, r, b) \in \mathbb{R}_{\geq 0}^4$, which is the distance from the point to the left, top, right, and bottom border, we compute

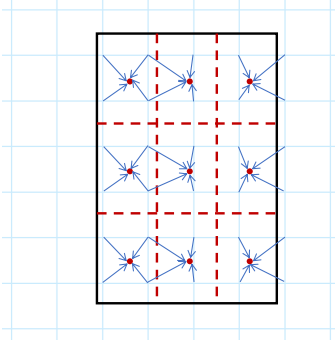


FIGURE 3.2. Offsets computation and feature sampling for RoIPred with kernel size 3×3 guided by the bounding box represented with black rectangle. The bounding box is evenly divided into nine crops. RoIPred uses the features interpolated at the crop centers. The offsets are computed accordingly.

the offsets $(\mathbf{o}_x, \mathbf{o}_y) \in \mathbb{R}^h \times \mathbb{R}^w$ for RoIPred with kernel size $h \times w$ as:

$$o_x(i) = \frac{(l+r)(2i+1)}{2w} - l \quad (3.2)$$

$$o_y(j) = \frac{(t+b)(2j+1)}{2h} - t, \quad (3.3)$$

where $(i, j) \in \{0, 1, \dots, w-1\} \times \{0, 1, \dots, h-1\}$. An example of sampling features for a 3×3 RoIPred is illustrated in Figure 3.2. This is equivalent to RoiAlign with bin size 1×1 .

Like RoiAlign, RoIPred can adjust its receptive field according to the region of interest. This makes the features pre-prediction to focus on their neighbour regions and thus help with retaining the local-coherence.

Where should we get the box supervisions, from the ground truth (gt) or the predicted detections? To answer this question, we design multiple teacher forcing schedules to choose ground truth or predictions as box supervision. The ‘w/o’ schedule always uses box predictions; The ‘random’ schedule use ground truth with probability $1-i/90,000$, where i is the current iteration number; The ‘fix’ schedule uses ground truth for the first 18k iterations; The ‘w/’ schedule always uses the ground truth.

We discover that *it is better to always use the ground truth*. Some may argue that using predictions at the later stage of training may help the model to generalize at inference time. However, it is not the case. Since only the ground truth box is aligned with our mask target, using ground truth boxes for RoIPred is the best for keep local-coherence.

Figure 3.3 shows the prediction tower structures with different mask predictor modules. All modules uses a 5×5 convolution module to predict a mask. RoIConv is the adaptation layer used by Chen *et al.* [63] and RoIPred is our proposed predictor. The difference is between RoIConv and RoIPred is that in RoIConv feature adaptation is before the convolution prediction, but in RoIPred, the alignment and prediction happens at the same time. We compare

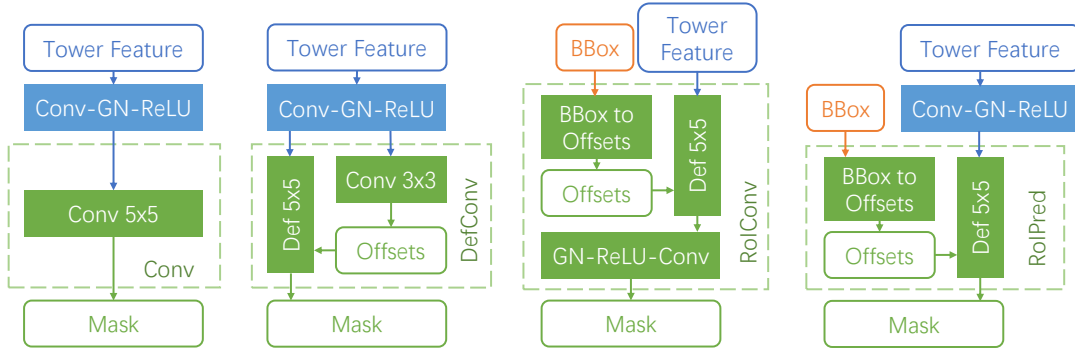


FIGURE 3.3. Different choices of mask predictors with three convolutions as mask tower. Conv is a normal convolution; DefConv is a normal deformable convolution without supervision for the offsets; RoIConv and RoIPred both use bounding box guided offsets. RoIConv [63] uses RPN predictions as offsets while RoIPred uses ground truth supervision. RoIConv has a deeper dense prediction layer following the align operation while RoIPred uses the align operation directly for prediction. The later is more reasonable because further convolution operations on the deformed feature map after alignment does not have clear meaning.

their performance in Section 3.5.1.

3.3.2 Smooth Features

In the dense object detector FCOS, a set of sizes of interest define the matching mechanism of objects to separate levels. A location is regarded as a positive sample for a certain level, if the location is inside a box and distance from it to the farthest edge of the box falls into the sizes of interest of that level, $[a, b)$. For a feature pyramid with P levels, the sizes of interest is defined as the positions of the p th level with stride $2^{p-1}s$ is defined as be $[2^{p-2}L, 2^{p-1}L)$, where L is the canonical length for level 1.

By mapping objects of different sizes to different levels, FCOS is able to learn a scale-invariant representation across the feature pyramid. However, it is very sensitive to the sizes of interest. If we assign an object to a level too low, the local-coherence is lost because of limited receptive field. On the other hand, if we assign it to a level too high, position information is lost because of the distortion in upsampling. In this section, we connect this phenomena with the position information loss and describe our solution to the importance of smooth features for our instance segmenter rooted signal processing techniques.

Downsampling by strided convolution and our RoIPred can introduce distortions to our sampled features. This artifact is called aliasing, which is common

when representing a high-resolution image at a lower resolution. The original features we predict masks from have sampling strides ranging from 8 to 128. The position information is gradually distorted moving from the lower to higher levels. Two challenges arise from this effect: 1) How can we reduce the position information loss? 2) How can we adjust our sampling strategy accordingly?

It is common to use low-pass filters such as sinc filter for spatial anti-aliasing in computer graphics [67]. This makes the signal smooth to spatial variance. Zhang [64] applied this technique to image classification and conditional image generation. The idea is to filter the feature with a convolutional blur kernel before downsampling. This makes the network more robust to input shift corruptions. We use this trick to avoid aliasing introduced during the downsampling. By making the sampled feature more smooth, we can prevent the corruption of position information. More specifically, we replace the strided convolutions $\text{Conv}_{K,s}$ in the backbone with $\text{BlurConv}_{K,s}$ where K is the kernel size and s is the stride:

$$\text{BlurConv}_{K,s}(\mathbf{X}) = \text{Blur}_{M,s} \circ \text{Conv}_{K,1}(\mathbf{X}), \quad (3.4)$$

where $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$ is the input feature and $\text{Blur}_{M,s}$ is a Gaussian blur kernel of size $M \times M$ and stride s . Thus, instead of downsampling directly with the original convolution, we first apply the convolution and then a blur filter without downsampling, then sample the features with stride s .

To better retain positional information, we make the mask predictor down to sample from lower levels, where the fine-grained information is rich. In FCOS [9], each level is assigned to a maximum length of object corresponding to this stride, restricting them to predict object bounding boxes and masks spanning from 8 to 16 pixels on the feature map. We hypothesize that the level of features for box regression is not optimal for mask prediction, we should move the mask prediction to lower levels where the instance details are better stored. But if we move them too low, the aliasing effect increases again, since this decreases the sampling frequency of the mask predictor. Therefore, we experiment moving the mask prediction lower only by one level.

We name this trick ‘lower-level mask sampling’. The formal definition is following. Given bounding boxes $\mathbf{B}_s \in \mathbb{R}^{N \times 4 \times \frac{W}{s} \times \frac{H}{s}}$, where s is the feature stride with regarding to the input size, to sample features $\mathbf{X}_{2^l s}$ from l layers lower, we use a RoIPred with stride 2^l :

$$\mathbf{M}_s = \text{RoIPred}_{2^l}(\mathbf{X}_{2^l s}, \mathbf{B}_s), \quad (3.5)$$

so that the mask prediction and locations are corresponding to the original features.

3.4 Two-Stage Interpretation

Because of the connections between RoIPred and RoIAlign, we can implement SmalMask in a two-stage fashion. The advantage of using two-stage framework is that we can save computations to only focus on the most likely foreground regions.

SmalMask2 We only transfer the shared SmalMask, to avoid adding an independent mask tower. The most straight-forward two-stage implementation of SmalMask is illustrated in Figure 3.4. We call this new model SmalMask2. In SmalMask2, RoIPred is replaced with a two-stage light-weight RoI head, SmalMask, which is a RoIAlign with sample ratio 1 followed by a fully connected layer. RoIPred and SmalMask have the same effect and their parameters are the same.

We only compute masks on the features inside the RoIs. RoIs are given by ground-truth bounding boxes during training and predicted boxes after post-processing during inference. Since out of the box losses are masked out and predictions are not selected, SmalMask2 has essentially the same computation as its dense counterpart SmalMask. The comparison between SmalMask and SmalMask2 has shed some light on the design of efficient instance segmenters. By reducing their head computation, two-stage models can be more efficient than dense models while avoiding the representation issues of the dense models.

We want to retain the level-matching mechanism in FCOS [9] which has been proved effective in the dense approach. So we match boxes by their longer sides instead of areas in the conventional RoIAlign: a box is assigned to a level if its longer side length falls into the sizes of interest of the corresponding level. For a feature pyramid with P levels, we set the sizes of interest of the p th level with stride $2^{p-1}s$ to be $[2^{p-2}L, 2^{p-1}L)$, where L is the canonical length for level 1. As discussed in Section 3.3.2, we can adjust the canonical length for the pooler to sample smooth features for mask prediction.

Multi-Scale Poolers Even though the features and operations are the same at inference phase, the two models are not equivalent during training. The most significant difference is the number of foreground features used in mask training. In the two-stage approach, each ground truth proposal is matched to one FPN level according to the pooler’s setting. But in the fully convolutional approach, it is typical for an object to be assigned to two levels, with the center locations

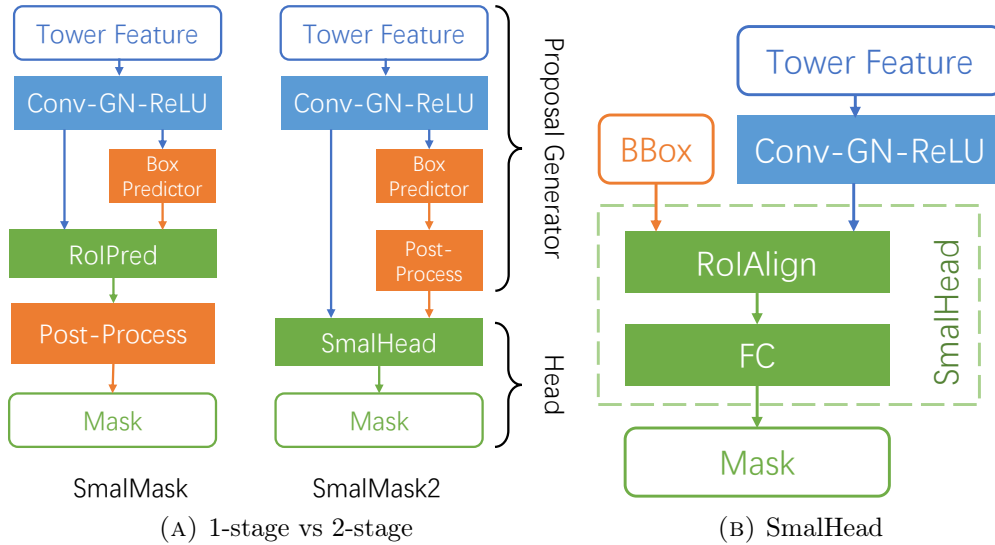


FIGURE 3.4. Two-stage implementation of RoIPred, SmalHead is an RoIAlign layer followed by a fully-connected layer.

on the lower one and border locations on the higher one. Since the tower weights are shared across FPN levels, training with features from multiple levels has a similar effect as within-network multi-scale augmentation [68].

Without this augmentation, our two-stage mask predictor suffers from underfitting. We propose a training time augmentation structure, termed multi-scale poolers, i.e., retaining this augmentation by creating two poolers with different canonical lengths during training. During inference, only the main pooler with larger canonical length is used. According to our empirical study, this is helpful in fixing the scale imbalance problem and has a similar effect as SNIPER [69].

SmalMask2-RT We design a compact version of our model, SmalMask2-RT, to compare with YOLACT [38], a recent real-time instance segmentation method: i) the number of convolution layers in the prediction head is reduced to three; ii) and we merge the classification tower and box tower into one by sharing their features.

3.5 Experiments

Our experiments are reported on the MSCOCO 2017 instance segmentation dataset [70]. It contains 123K images with 80-class instance labels. Our models are trained on the `train2017` split (115K images) and the ablation study is carried out on the `val2017` split (5K images). Final results are on `test-dev`. The evaluation metrics are COCO mask average precision (AP), AP at IoU 0.5 (AP_{50}), 0.75 (AP_{75}) and AP for objects at different sizes AP_S , AP_M , and AP_L .

Method	tower	AP	AP ₅₀	AP ₇₅
Conv	separate	29.2	52.3	29.5
	shared	29.0	52.3	28.7
DefConv	separate	29.7	52.6	30.4
	shared	29.6	52.6	29.9
RoIConv [63]	separate	30.6	53.4	31.5
RoIPred	separate	31.3	53.4	32.7
	shared	30.8	53.7	31.8

TABLE 3.1. **Performance of box-aligned predictor:** All predictors have kernel size 5×5 .

Training details Unless specified, for the ablation study, ImageNet pre-trained ResNet-50 [71] is used as our backbone network. We use the separate SmalMask with four consecutive 3×3 convolutions. All networks are trained with the $1 \times$ schedule of FCOS [9], i.e., we train our model for 90K iterations with batch size 16 in 4 GPUs and base learning rate 0.01 with constant warm-up of 1k iterations. The learning rate is reduced by a factor of 10 at iteration 60K and 80K. Input images are resized to have shorter side 800 and longer side at maximum 1333. All hyper-parameters are set to be the same with FCOS [9].

Testing details The unit for inference time is ‘ms’ in all our tables. For the ablation experiments, performance and time of our models are measured with one image per batch on one 1080Ti GPU.

3.5.1 Ablation Experiments

We investigate the effectiveness of our aligned mask module by carrying out ablation experiments on the follow variations.

Aligned mask predictors We compare the performance of aligned predictors described in Section 3.3.1. Table 3.1 shows the results of replacing the aligned RoIPred with normal convolution (Conv), deformable convolution (DefConv) and RoIConv. The proposed RoIPred shows obvious advantage over other predictors. We guess deformable convolution cannot handle the alignment problem for two reasons: 1) It does not have access to the accurate box information, so the align quality is lower. 2) Offsets are not perfectly aligned with the supervisions. Comparing to RoIConv which also uses the gt boxes, our RoIPred is still 0.7 mAP better. The reason we assume is that it is better to keep the features aligned until the prediction. The prediction layer in RoIConv after the aligned sampler can no longer refine its information because each location is independent.

Kernel	Time	AP	AP ₅₀	AP ₇₅
3×3	88	30.9	53.2	32.0
5×5	115	31.3	53.4	32.7
7×7	153	31.4	53.7	32.8

TABLE 3.2. **Kernel sizes difference:** The model we use for the experiments is the separate version of SmalMask.

TF?	AP	AP ₅₀	AP ₇₅
w/o	30.4	52.9	31.5
random	30.4	52.9	31.5
fix	30.4	53.0	31.5
w/	31.3	53.4	32.7

TABLE 3.3. **Box supervision schedules comparison:** TF is abbreviation for teacher forcing. The four configurations corresponds to our four schedules of when to use ground truth supervision.

What is the optimal kernel size? We measure the average inference time and performance for RoIPred with different kernel sizes. The results are shown in Table 3.2. By increasing the kernel size of RoIPred from 3×3 to 7×7 , the performance keeps improving. We do not try sizes larger than 7×7 because the improvement from 5×5 is very subtle, however, it increases the inference time by 38ms. We decide to use 5×5 because of the speed trade-off.

Should we use teacher forcing? We test the effectiveness of ground truth supervision for the offsets. Four different schedules are tested. The w/o schedule always uses box predictions; The random schedule use ground truth with probability $1 - i/90,000$, where i is the current iteration number; The fix schedule uses ground truth for the first 18k iterations; The w/ schedule always uses the ground truth. During training, we want the segmentation targets to be local-coherent. As shown in Table 3.3, the one that always uses ground truth has clear advantage to the others since it can always provide the most accurate segmentation targets. The other three schedules do not have clear difference from one another.

How to assign levels and sizes for mask predictions? We conjecture that mask prediction should be predicted from features with higher resolution comparing to bounding box prediction. So we conduct experiments changing the feature level assignments of the RoIPred predictors. Three configurations are tested, the original assignment (P3–P7), top level down by one (P3–P6, P6), and all levels except the first down by one (P3, P3–P6). As shown in Table 3.4, even though simply moving the top predictor does not help with the

Levels	L	AP	AP_S	AP_M	AP_L
P3–P7	64	31.3	15.5	34.4	43.5
P3–P6, P6	64	31.1	15.2	34.4	43.2
P3, P3–P6	64	31.7	15.3	34.9	44.3
P3, P3–P6 7×7	64	31.7	15.5	35.0	44.6
P3, P3–P6	128	31.4	15.2	35.1	43.7
P3, P3–P6	48	31.6	15.0	35.1	44.3

TABLE 3.4. **Comparison of level selections and sizes of interest:** The levels are the feature names from the pyramid where we predict masks from. P3–P7 is the baseline model using the same feature levels for box and mask predictions; P3–P6, P6 means moving the top level prediction from P7 to P6; P3, P3–P6 means moving all predictors except the first down by one level. We also include experiments with different canonical length L . All predictions use a 5×5 RoIPred as the predictor except the one specified 7×7 .

Operator	BatchNorm	AP	AP_{50}	AP_{75}
Conv	freeze	30.8	53.0	32.2
	train	31.4	53.8	32.4
BlurConv	freeze	31.4	54.1	32.8
	train	31.9	54.5	33.4

TABLE 3.5. **Improvements of anti-aliased downsampling:** Conv is the normal strided convolution. BlurConv is the operator described in eq. 3.4. In the second column, freeze means that batchnorm is not updated, and train means that batchnorm parameters are trained.

Model	L	AP	AP_S	AP_M	AP_L
SmalMask2	96	32.3	15.9	35.4	45.4
	128	32.1	16.1	35.3	44.9
	64, 128	32.5	16.5	35.6	45.1
SmalMask2-RT	128	29.0	12.9	31.9	43.0
	64, 128	29.5	11.8	32.4	44.1

TABLE 3.6. **Two-stage results:** Both models are trained with resize augmentation. SmalMask2 has shorter size [640, 800], and SmalMask2-Rt [440, 550]. Models are implemented with Detectron2. L is the canonical length of pooler(s) used during training. During testing, the pooler with largest L is used.

Method	Backbone	Epochs	Time	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Mask R-CNN*	R-50	72	> 90	36.8	59.2	39.3	17.1	38.7	52.1
TensorMask		72	> 380	35.5	57.3	37.4	16.6	37.0	49.1
SmalMask2+		36	76.0	36.1	59.0	38.2	21.5	39.0	44.5
Mask R-CNN*	R-101	72	> 118	38.3	61.2	40.8	18.2	40.6	54.1
TensorMask		72	> 400	37.3	59.5	39.5	17.5	39.3	51.6
SmalMask2+		36	97.9	38.0	60.8	40.6	22.5	40.8	47.5

TABLE 3.7. Comparison with state-of-the-art on the COCO-Things `test-dev`. Our SmalMask2+ models are based on Detectron2. Speed is measured with a single 1080Ti. We increase the receptive field for the R101 SmalMask2+ predictor from 5×5 to 7×7 .

Method	Backbone	NMS	Resolution	Time	AP ^{bb}	AP	AP ₅₀	AP ₇₅
YOLACT	R-101	Fast	550×550	34.2	32.5	29.8	48.3	31.3
			700×700	46.7	33.4	30.9	49.8	32.5
Mask R-CNN	R-50	Batched	$550 \times *$	63.4	39.1	35.3	56.5	37.6
SmalMask2-RT				38.3	39.0	32.5	54.1	34.1

TABLE 3.8. **Real-time setting comparison** of speed and accuracy with other state-of-the-art methods on COCO `val2017`. Metrics for YOLACT are obtained using their official code and trained model. Mask R-CNN and SmalMask models are trained and measured using Detectron2. Resolution $550 \times *$ means using shorter side 550 in inference.

accuracy, moving all predictors down in parallel improves the performance. This is because lower features have higher resolution for masks, and the number of objects assigned to each level is more balanced. We also notice that increasing the predictor kernel size after lower level sampling does not lead to significant improvement.

We also modify the canonical length L for different levels after we notice the advantage of using lower level features for prediction. From the original 64 to 80 (larger) and 48 (smaller). Also shown in Table 3.4, both modifications cannot further improve the performance, because these also affect the bounding box detection.

Smooth downsampling Replacing the strided convolutions in the backbone with BlurConv yields 0.6 AP improvement (Table 3.5). The improvement is consistent if we update the batch norm parameters, which results in 0.5 AP improvement.

3.5.2 Two-Stage Version

We train our two-stage version SmalMask2 and SmalMask2-RT with $1\times$ schedule. Both models do not have independent mask tower, and SmalMask2-RT has a single tower with three convolution layers for joint classification and box regression. Both models are trained with multi-scale augmentation, input for the RT models are smaller (550). We demonstrate the effect of multi-scale poolers in Table 3.6. SmalMask performs better with a single pooler with $L = 96$ than $L = 128$. But adding an auxiliary pooler with $L = 64$ increases the mask AP by 0.4 on SmalMask2 and 0.5 on SmalMask2-RT.

3.5.3 Main Results

We compare Mask R-CNN [2] and TensorMask [60] on COCO `test-dev` with our two models, SmalMask2+ with shorter side 800, and the real-time version SmalMask2-RT with shorter side 550. Both models utilize SmalHead for prediction and use BlurConv for downsampling. Since our ablation models are heavily under-fitted, we increase the training iterations to 270K ($3\times$ schedule), tuning learning rate down at 180K and 240K. Following Chen *et al.*'s strategy [60], we use multi-scale training with shorter side randomly sampled from [640, 800] and [440, 550] respectively.

As shown in Table 5.8, SmalMask2+ outperforms TensorMask using only half of the training iterations. The R101 model achieves better accuracy. SmalMask2 is also more efficient. Measured on V100 GPU, the best R-101 SmalMask2 runs at 0.07s/image, *vs.* TensorMask's 0.38s per image, *vs.* Mask R-CNN's 0.09s per image [60]. Since our SmalHead has only one FC layer, the additional time for complex scenes is nearly negligible. On the contrary, for two-stage Mask R-CNN, the head computation is much longer, and the inference time increases significantly if the number of predicted instances grows.

YOLOACT resizes all images to square, changing the aspect ratios of inputs. Also, a paralleled NMS algorithm called Fast NMS is used in YOLOACT. We do not adopt these two configurations because they are not conventionally used in instance segmentation researches. In YOLOACT, a speedup of 12ms is reported by using Fast NMS. We instead use the Batched NMS in Detectron2, which could be slower than Fast NMS but does not sacrifice the accuracy.

Results in Table 3.8 shows that *SmalMask2-RT is 11ms faster and 1.4 AP higher than YOLOACT-700*, making our model competitive under the real-time settings.

3.6 Conclusion

We revisit the relationships between dense and two-stage approaches for instance segmentation using a simple baseline duo, SmalMask and SmalMask2. Based on both models, we propose multiple solutions to improve the local-coherence and positional information for mask prediction, and their underlying connections are discovered. Our main findings are 1) many tricks for dense models and two-stage models are interchangeable 2) by reducing head computation, two-stage models can be more efficient than the dense models while avoiding their representation issues. Our SmalMask2 is a simple and effective framework for instance segmentation. It outperforms the more complex TensorMask while being five times faster. Furthermore the real-time version SmalMask2-RT achieves 32.5 mAP at 26 FPS evaluated on a single 1080Ti. We believe our SmalMask2 could serve as a simple baseline for many other instance-level dense prediction tasks.

Statement of Authorship

Title of Paper	BlendMask: Top-Down Meets Bottom-Up for Instance Segmentation
Publication Status	<input checked="" type="checkbox"/> Published <input type="checkbox"/> Accepted for Publication <input type="checkbox"/> Submitted for Publication <input type="checkbox"/> Unpublished and Unsubmitted work written in manuscript style
Publication Details	CVPR 20: 8570-8578

Principal Author

Name of Principal Author (Candidate)	Hao Chen
Contribution to the Paper	Coding, experiment and majority writing
Overall percentage (%)	70%
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.
Signature	<div style="display: flex; justify-content: space-between;"> _____ Date </div>

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

- i. the candidate's stated contribution to the publication is accurate (as detailed above);
- ii. permission is granted for the candidate to include the publication in the thesis; and
- iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Kunyang Sun
Contribution to the Paper	discussion, conducting initial experiments figures illustrations
Signature	<div style="display: flex; justify-content: space-between;"> _____ Date </div>

Name of Co-Author	Zhi Tian
Contribution to the Paper	discussion and writing revision
Signature	<div style="display: flex; justify-content: space-between;"> _____ Date </div>

Name of Co-Author	Chunhua Shen	
Contribution to the Paper	Discussion and writing revision	
Signature		Date

Name of Co-Author	Yongming Huang	
Contribution to the Paper	Discussion and writing revision	
Signature		Date

Name of Co-Author	Youliang Yan	
Contribution to the Paper	Discussion and writing revision	
Signature		Date 14/07/2020



Chapter 4

BlendMask: Fully-Convolutional Framework for Instance Segmentation

4.1 Introduction

Instance segmentation is one of the fundamental vision tasks. Recently, fully convolutional instance segmentation methods have drawn much attention as they are often simpler and more efficient than two-stage approaches like Mask R-CNN. To date, almost all such approaches fall behind the two-stage Mask R-CNN method in mask precision when models have similar computation complexity, leaving great room for improvement. In this chapter, we achieve improved mask prediction by effectively combining instance-level information with semantic information with lower-level fine-granularity. Our main contribution is a blender module which draws inspiration from both top-down and bottom-up instance segmentation approaches. The proposed BlendMask can effectively predict dense per-pixel position-sensitive instance features with very few channels, and learn attention maps for each instance with merely one convolution layer, thus being fast in inference. BlendMask can be easily incorporated with the state-of-the-art one-stage detection frameworks and outperforms Mask R-CNN under the same training schedule while being 20% faster. A light-weight version of BlendMask achieves 34.2% mAP at 25 FPS evaluated on a single 1080Ti GPU card. Because of its simplicity and efficacy, we hope that our BlendMask could serve as a simple yet strong baseline for a wide range of instance-wise prediction tasks.

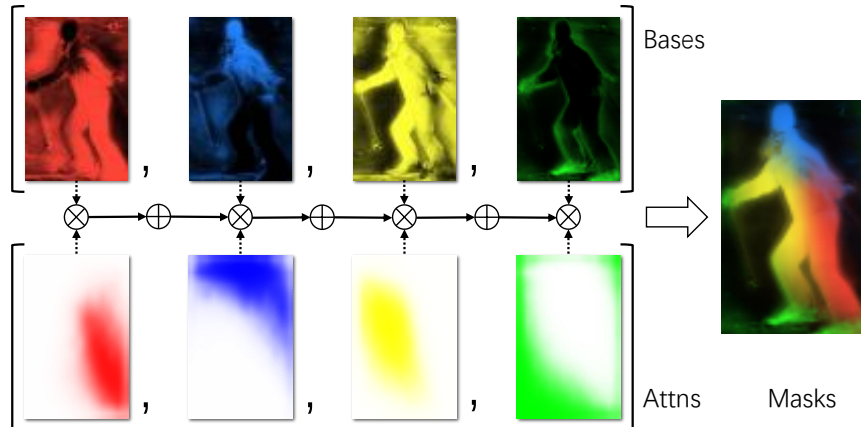


FIGURE 4.1. **Blending process.** We illustrate an example of the learned bases and attentions. Four bases and attention maps are shown in different colors. The first row are the bases, and the second row are the attentions. Here \otimes represents element-wise product and \oplus is element-wise sum. Each basis multiplies its attention and then is summed to output the final mask.

4.2 Background

The top performing object detectors and segmenters often follow a two-stage paradigm. They consist of a fully convolutional network, region proposal network (RPN), to perform dense prediction of the most likely regions of interest (RoIs). A set of light-weight networks, a.k.a. heads, are applied to re-align the features of RoIs and generate predictions [8]. The quality and speed for mask generation is strongly tied to the structure of the mask heads. In addition, it is difficult for independent heads to share features with related tasks such as semantic segmentation which causes trouble for network architecture optimization.

Recent advances in one-stage object detection prove that one-stage methods such as FCOS can outperform their two-stage counterparts in accuracy [9]. Enabling such one-stage detection frameworks to perform dense instance segmentation is highly desirable as 1) models consisting of only conventional operations are simpler and easier for cross-platform deployment; 2) a unified framework provides convenience and flexibility for multi-task network architecture optimization.

Dense instance segmenters can date back to DeepMask [59], a top-down approach which generates dense instance masks with a sliding window. The representation of mask is encoded into a one-dimensional vector at each spatial location. Albeit being simple in structure, it has several obstacles in training that prevent it from achieving superior performance: 1) local-coherence between

features and masks is lost; 2) the feature representation is redundant because a mask is repeatedly encoded at each foreground feature; 3) position information is degraded after downsampling with strided convolutions.

The first issue was studied by Dai *et al.* [41], who attempt to retain local-coherence by keeping multiple position-sensitive maps. This idea has been explored to its limits by Chen *et al.* [60], who proposes a dense aligned representation for each location of the target instance mask. However, this approach trades representation efficiency for alignment, making the second issue difficult to resolve. The third issue prevents heavily downsampled features to provide detailed instance information.

Recognizing these difficulties, a line of research takes a bottom-up strategy [40], [72], [73]. These methods generate dense per-pixel embedding features and use some techniques to group them. Grouping strategies vary from simple clustering [39] to graph-based algorithms [40] depending on the embedding characteristics. By performing per-pixel predictions, the local-coherence and position information is well retained. The shortcomings for bottom-up approaches are: 1) heavy reliance on the dense prediction quality, leading to sub-par performance and fragmented/joint masks; 2) limited generalization ability to complex scenes with a large number of classes; 3) requirement for complex post-processing techniques.

In this chapter, we consider hybridizing top-down and bottom-up approaches. We recognize two important predecessors, FCIS [35] and YOLACT [38]. They predict instance-level information such as bounding box locations and combine it with per-pixel predictions using cropping (FCIS) and weighted summation (YOLACT), respectively. We argue that *these overly simplified assembling designs may not provide a good balance for the representation power of top- and bottom-level features.*

Our objective is similar to TensorMask [60], which also discusses the alignment problem in dense instance segmenters and tries to reduce the gap between dense and two-stage approaches. TensorMask uses a proposal-free segmentation framework, making the dense prediction of large masks more compute-intensive. Its proposed align representation requires each point to encode redundant segmentation information for neighbourhood predictions, which makes the model run three times slower than Mask R-CNN. Different from their approach, we choose a proposal-based framework, more similar to Mask R-CNN and we reuse same set of features for all dense mask predictions within the same object, which makes our model efficient.

To break through these limitations, we propose a new proposal-based mask

generation framework, termed BlendMask. The top- and bottom-level representation workloads are balanced by a blender module.

We base of instance segmentation network on a simple anchor-free object detector FCOS. The benefits of using anchor-free object detector are twofold. First, it is important to map target sizes with proper pyramid levels to fit the effective receptive field for the features. Second, *removing anchors enables us to assign heavier duties to the top-level instance prediction module without introducing overall computation overhead*. For example, inferring shape and pose information alongside the bounding box detection would take about eight times more computation for anchor-based frameworks than ours.

This makes it intractable for anchor based detectors to balance the top *vs.* bottom workload (i.e., learning instance-aware maps¹ *vs.* bases). We assume that this might be the reason why YOLACT can only learn one single scalar coefficient for each prototype/basis given an instance when computation complexity is taken into account. *Only with the use of anchor-free bounding box detectors, this restriction is removed*.

Both levels are guaranteed to describe the instance information within their best capacities. As shown in our experiments in Section 4.4, our blender module improves the performance of bases combination methods comparing to YOLACT and FCIS by a large margin without increasing computation complexity.

Refining coarse masks with lower-level features BlendMask merges top-level coarse instance information with lower-level fine-granularity. This idea resembles MaskLab [37] and Instance Mask Projection (IMP) [74], which concatenates mask predictions with lower layers of backbone features. The differences are clear. Our coarse mask acts like an attention map. The generation is extremely light-weight, without the need of using semantic or positional supervision, and is closely tied to the object generation. As shown in Section 4.3.4, our lower-level features have clear contextual meanings, even though not explicitly guided by bins or crops. Further, our blender does not require a subnet on top of the merged features as in MaskLab [37] and IMP [74], which makes our method more efficient. In parallel to this work recent two single shot instance segmentation methods have shown good performance [75], [76].

Higher-level features correspond to larger receptive field and can better capture overall information about instances such as poses, while lower-level features preserve better location information and can provide finer details. One of the focuses of our work is to investigate ways to better merging these two in fully

¹Attention maps for BlendMask and simple weight scalars for YOLACT.

convolutional instance segmentation. More specifically, we generalize the operations for proposal-based mask combination by enriching the instance-level information and performing more fine-grained position-sensitive mask prediction. We carry out extensive ablation studies to discover the optimal dimensions, resolutions, alignment methods, and feature locations. Concretely, we are able to achieve the followings:

- We devise a flexible method for proposal-based instance mask generation called blender, which incorporate rich instance-level information with accurate dense pixel features. In head-to-head comparison, our blender surpasses the merging techniques in YOLACT [38] and FCIS [35] by 1.9 and 1.3 points in mAP on the COCO dataset respectively.
- We propose a simple architecture, BlendMask, which is closely tied to the state of the art one-stage object detector, FCOS [9], by adding moldiest computation overhead to the already simple framework.
- One obvious advantage of BlendMask is that its inference time does not increase with the number of predictions as conventional two-stage methods do, which makes it more robust in real-time scenarios.
- The performance of BlendMask achieves mAP of 37.0% with the ResNet-50 [77] backbone and 38.4% mAP with ResNet-101 on the COCO dataset, outperforming Mask R-CNN [2] in accuracy while being about 20% faster. We set new records for fully convolutional instance segmentation, surpassing TensorMask [60] by 1.1 points in mask mAP with only half training iterations and $1/5$ inference time.

To our knowledge, BlendMask may be the first algorithm that can outperform Mask R-CNN in both mask AP and inference efficiency.

- BlendMask can naturally solve panoptic segmentation without any modification (refer to Section 4.4.4), as the bottom module of BlendMask can segment ‘*things and stuff*’ simultaneously.
- Compared with Mask R-CNN’s mask head, which is typically of 28×28 resolution, BlendMask’s the bottom module is able to output masks of much higher resolution, due to its flexibility and the bottom module not being strictly tied to the FPN. Thus BlendMask is able to produce masks with more accurate edges, as shown in Figure 4.4. For applications such as graphics, this can be very important.
- The proposed BlendMask is general and flexible. With minimal modification, we can apply BlendMask to solve other instance-level recognition tasks such as keypoint detection.

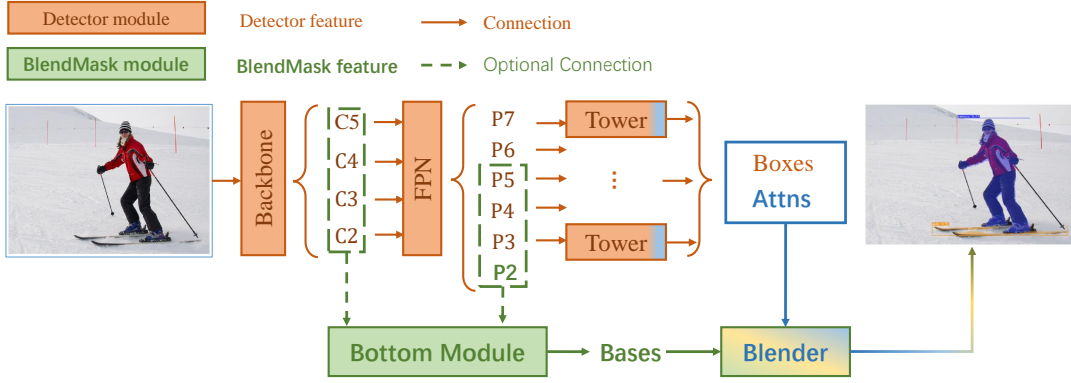


FIGURE 4.2. **BlendMask pipeline** Our framework builds upon the state-of-the-art FCOS object detector [9] with minimal modification. The bottom module uses either backbone or FPN features to predict a set of bases. A single convolution layer is added on top of the detection towers to produce attention masks along with each bounding box prediction. For each predicted instance, the blender crops the bases with its bounding box and linearly combine them according the learned attention maps. Note that the Bottom Module can take features either from ‘C’, or ‘P’ as the input.

4.3 Methodology

4.3.1 Overall pipeline

BlendMask consists of a detector network and a mask branch. The mask branch has three parts, a bottom module to predict the score maps, a top layer to predict the instance attentions, and a blender module to merge the scores with attentions. The whole network is illustrated in Figure 4.2.

Bottom module Similar to other proposal-based fully convolutional methods [35], [38], we add a bottom module predicting score maps which we call bases, \mathbf{B} . \mathbf{B} has a shape of $N \times K \times \frac{H}{s} \times \frac{W}{s}$, where N is the batch size, K is the number of bases, $H \times W$ is the input size and s is the score map output stride. We use the decoder of DeepLabV3+ in our experiments. Other dense prediction modules should also work without much difference. The input for the bottom module could be backbone features like conventional semantic segmentation networks [7], or the feature pyramids like YOLACT and Panoptic FPN [1].

Top layer We also append a single convolution layer on each of the detection towers to predict top-level attentions \mathbf{A} . Unlike the mask coefficients in YOLACT, which for each pyramid with resolution $W_l \times H_l$ takes the shape of $N \times K \times H_l \times W_l$, our \mathbf{A} is a tensor at each location with shape $N \times (K \cdot M \cdot$

$M) \times H_l \times W_l$, where $M \times M$ is the attention resolution. With its 3D structure, our attention map can encode instance-level information, e.g. the coarse shape and pose of the object. M is typically smaller² than the mask predictions in top-down methods since we only ask for a rough estimate. We predict it with a convolution with $K \cdot M \cdot M$ output channels. Before sending them into the next module, we first apply FCOS [9] post-process to select the top D box predictions $P = \{\mathbf{p}_d \in \mathbb{R}_{\geq 0}^4 | d = 1 \dots D\}$ and corresponding attentions $A = \{\mathbf{a}_d \in \mathbb{R}^{K \times M \times M} | d = 1 \dots D\}$.

Blender module is the key part of our BlendMask. It combines position-sensitive bases according to the attentions to generate the final prediction. We discuss this module in detail in the next section.

4.3.2 Blender module

The inputs of the blender module are bottom-level bases \mathbf{B} , the selected top-level attentions A and bounding box proposals P . First we use RoIPooler in Mask R-CNN [2] to crop bases with each proposal \mathbf{p}_d and then resize the region to a fixed size $R \times R$ feature map \mathbf{r}_d .

$$\mathbf{r}_d = \text{RoIPool}_{R \times R}(\mathbf{B}, \mathbf{p}_d), \quad \forall d \in \{1 \dots D\}. \quad (4.1)$$

More specifically, we use sampling ratio 1 for RoIAlign, i.e. one bin for each sampling point. The performance of using nearest and bilinear poolers are compared in Table 4.6. During training, we simply use ground truth boxes as the proposals. During inference, we use FCOS prediction results.

Our attention size M is smaller than R . We interpolate \mathbf{a}_d from $M \times M$ to $R \times R$, into the shapes of $R = \{\mathbf{r}_d | d = 1 \dots D\}$.

$$\mathbf{a}'_d = \text{interpolate}_{M \times M \rightarrow R \times R}(\mathbf{a}_d), \quad \forall d \in \{1 \dots D\}. \quad (4.2)$$

Then \mathbf{a}'_d is normalize with softmax function along the K dimension to make it a set of score maps \mathbf{s}_d .

$$\mathbf{s}_d = \text{softmax}(\mathbf{a}'_d), \quad \forall d \in \{1 \dots D\}. \quad (4.3)$$

²The largest M we try is 14.

Then we apply element-wise product between each entity \mathbf{r}_d , \mathbf{s}_d of the regions R and scores S , and sum along the K dimension to get our mask logit \mathbf{m}_d :

$$\mathbf{m}_d = \sum_{k=1}^K \mathbf{s}_d^k \circ \mathbf{r}_d^k, \quad \forall d \in \{1 \dots D\}, \quad (4.4)$$

where k is the index of the basis. We visualize the mask blending process with $K = 4$ in Figure 4.1.

4.3.3 Configurations and baselines

We consider the following configurable hyper-parameters for BlendMask:

- R , the bottom-level RoI resolution,
- M , the top-level prediction resolution,
- K , the number of bases,
- bottom module input features, it can either be features from the backbone or the FPN,
- sampling method for bottom bases, nearest-neighbour or bilinear pooling,
- interpolation method for top-level attentions, nearest neighbour or bilinear upsampling.

We represent our models with abbreviation R_K_M. For example, 28_4_4 represents bottom-level region resolution of 28×28 , 4 number of bases and 4×4 top-level instance attentions. By default, we use backbone features C3 and C5 to keep aligned with DeepLabv3+ [7]. Nearest neighbour interpolation is used in top-level interpolation, for a fair comparison with FCIS [35]. Bilinear sampling is used in the bottom level, consistent with RoIAlign [2].

4.3.4 Semantics encoded in learned bases and attentions

By examining the generated bases and attentions on val2017, we observe this pattern. On its bases, BlendMask encodes two types of local information, 1) whether the pixel is on an object (semantic masks), 2) whether the pixel is on certain part of the object (position-sensitive features).

The complete bases and attentions projected onto the original image are illustrated in Figure 4.3. The first two bases (red and blue) detects points on the upper-right and bottom-left parts of the objects. The third (yellow) base activates on points more likely to be on an object. The fourth (green) base only activates on the borders of objects. Position-sensitive features help us separate

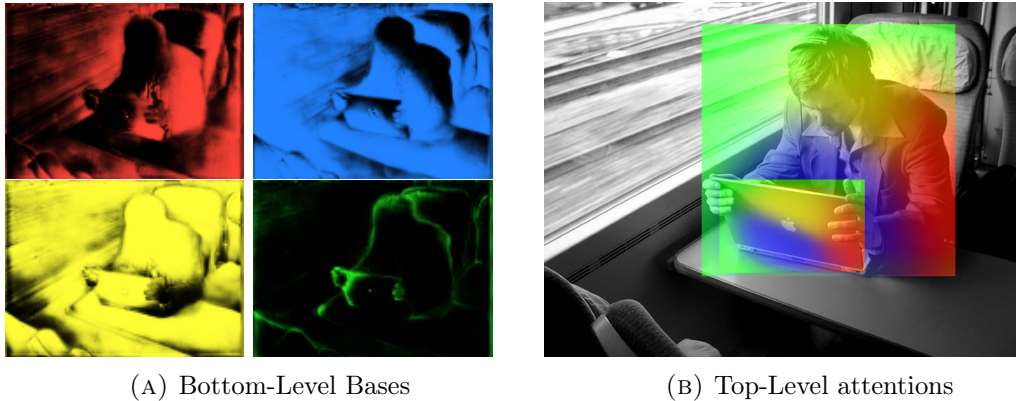


FIGURE 4.3. **Detailed view of learned bases and attentions.** The left four images are the bottom-level bases. The right image is the top-level attentions. Colors on each position of the attentions correspond to the weights of the bases, indicating from which part of which base is the mask assembled.

overlapping instances, which enables BlendMask to represent all instances more efficiently than YOLACT [38]. The positive semantic mask makes our final prediction smoother than FCIS [35] and the negative one can further suppress out-of-instance activations. We compare our blender with YOLACT and FCIS counterparts in Table 4.1. BlendMask can learn more accurate features than YOLACT and FCIS with much fewer number of bases (4 *vs.* 32 *vs.* 49, see Section 4.4.2).

4.4 Experiments

Our experiments are reported on the MSCOCO 2017 instance segmentation dataset [70]. It contains 123K images with 80-class instance labels. Our models are trained on the `train2017` split (115K images) and the ablation study is carried out on the `val2017` split (5K images). Final results are on `test-dev`. The evaluation metrics are COCO mask average precision (AP), AP at IoU 0.5 (AP_{50}), 0.75 (AP_{75}) and AP for objects at different sizes AP_S , AP_M , and AP_L .

Training details Unless specified, ImageNet pre-trained ResNet-50 [71] is used as our backbone network. DeepLabv3+ [7] with channel width 128 is used as our bottom module. For ablation study, all the networks are trained with the $1\times$ schedule of FCOS [9], i.e., 90K iterations, batch size 16 on 4 GPUs, and base learning rate 0.01 with constant warm-up of 1k iterations. The learning rate is reduced by a factor of 10 at iteration 60K and 80K. Input images are resized to have shorter side 800 and longer side at maximum 1333. All hyperparameters are set to be the same with FCOS [9].

Method	AP	AP ₅₀	AP ₇₅
Weighted-sum	29.7	52.2	30.1
Assembler	30.3	52.5	31.3
Blender	31.6	53.4	33.3

TABLE 4.1. **Comparison of different strategies for merging top and bottom modules.** Here the model used is 28_4_4. Weighted-sum is our analogy to YOLACT, reducing the top resolution to 1×1 . Assembler is our analogy to FCIS, where the number of bases is increased to 16, matching each of the region crops without the need of top-level attentions.

Testing details The unit for inference time is ‘ms’ in all our tables. For the ablation experiments, performance and time of our models are measured with one image per batch on one 1080Ti GPU.

4.4.1 Ablation experiments

We investigate the effectiveness of our blender module by carrying out ablation experiments on the configurable hyperparameters in Section 4.3.3.

Merging methods: Blender vs. YOLACT vs. FCIS Similar to our method, YOLACT [38] and FCIS [35] both merge proposal-based bottom regions to create mask prediction. YOLACT simply performs a weighted sum of the channels of the bottom regions; FCIS assembles crops of position-sensitive masks without modifications. Our blender can be regarded as a generalization where both YOLACT and FCIS merging are special cases: The blender with 1×1 top-level resolution degenerates to YOLACT; and FCIS is the case where we use fixed one-hot blending attentions and nearest neighbour top-level interpolation.

Results of these variations are shown in Table 4.1. Our blender surpasses the other alternatives by a large margin. We assume the reason is that other methods lack instance-aware guidance on the top. By contrast, our blender has a fine-grained top-level attention map, as illustrated in Figure 4.3.

Top and bottom resolutions: We measure the performances of our model with different top- and bottom-level resolutions, trying bottom pooler resolution R being 28 and 56, with R/M ratio from 14 to 4. As shown in Table 4.2, by increasing the attention resolution, we can incorporate more detailed instance-level information while keeping the running time roughly the same. Notice that the gain slows down at higher resolutions revealing limit of detailed information on the top-level. So we don’t include larger top settings with R/M ratio smaller than 4.

R	M	Time	AP	AP_S	AP_M	AP_L
28	2	72.7	30.6	14.3	34.1	42.5
	4	72.9	31.6	14.8	35.2	45.0
	7	73.9	32.0	15.3	35.6	45.0
56	4	72.9	32.5	14.9	36.1	46.0
	7	74.1	33.1	15.1	36.6	47.7
	14	77.7	33.3	16.3	36.8	47.4

TABLE 4.2. **Resolutions:** Performance by varying top-/bottom-level resolutions, with the number of bases $K = 4$ for all models. Top-level attentions are interpolated with nearest neighbour. Bottom module uses backbone features C3, C5. The performance increases as the attention resolution grows, saturating at resolutions of near 1/4 of the region sizes.

Different from two-stage approaches, increasing the bottom-level bases pooling resolution does not introduce much computation overhead. Increasing it from 28 to 56 only increases the inference time within 0.2ms while mask AP increases by 1 point. In further ablation experiment, we set $R = 56$ and $M = 7$ for our baseline model if not specified.

Number of bases: YOLACT [38] uses 32 bases concerning the inference time. With our blender, the number of bases can be further reduced, to even just one. We report our models with number of bases varying from 1 to 8. Different from normal blender, the one-basis version uses sigmoid activation on both the base and the attention map. Results are shown in Table 4.3. Since instance-level information is better represented with the top-level attentions, we only need 4 bases to get the optimal accuracy. $K = 4$ is adopted by all subsequent experiments.

Bottom feature locations: backbone vs. FPN We compare our bottom module feature sampling locations. By using FPN features, we can improve the performance while reducing the running time (see Table 4.4). In later experiments, if not specified, we use P3 and P5 of FPN as our bottom module input.

K	AP	AP_{50}	AP_{75}
1	30.6	52.9	31.6
2	31.2	53.4	32.3
4	33.1	54.1	34.9
8	33.0	53.9	34.9

TABLE 4.3. **Number of bases:** Performances of 56_K_7 models. For the configuration of one basis, we use sigmoid activation for both top and bottom features. Our model works with a small number of bases.

Features	M	Time (ms)	AP	AP ₅₀	AP ₇₅
C3, C5	7	74.1	33.1	54.1	34.9
	14	77.7	33.3	54.1	35.3
P3, P5	7	72.5	33.3	54.2	35.3
	14	76.4	33.4	54.3	35.5

TABLE 4.4. **Bottom feature locations:** Performance with bottom resolution 56×56 , 4 bases and bilinear bottom interpolation. C3, C5 uses features from backbone. P3, P5 uses features from FPN.

Interpolation	M	AP	AP ₅₀	AP ₇₅
Nearest	7	33.3	54.2	35.3
	14	33.4	54.3	35.5
Bilinear	7	33.5	54.3	35.7
	14	33.6	54.6	35.6

TABLE 4.5. **Top interpolation:** Performance with bottom resolution 56×56 , 4 bases and bilinear bottom interpolation. Nearest represents nearest-neighbour upsampling and bilinear is bilinear interpolation.

Interpolation method: nearest vs. bilinear In Mask R-CNN [2], RoIAlign plays a crucial role in aligning the pooled features to keep local-coherence. We investigate the effectiveness of bilinear interpolation for bottom RoI sampling and top-level attention re-scaling. As shown in Table 4.5, changing top interpolation from nearest to bilinear yields a marginal improvement of 0.2 AP.

The results of bottom sampling with RoIPool [78] (nearest) and RoIAlign [2] (bilinear) are shown in Table 4.6. For both resolutions, the aligned bilinear sampling could improve the performance by almost 2AP. Using aligned features for the bottom-level is more crucial, since it is where the detailed positions are predicted. Bilinear top and bottom interpolation are adopted for our final models.

Other improvements: We experiment on other tricks to improve the performance. First we add auxiliary semantic segmentation supervision on P3 similar to YOLACT [38]. Then we increase the width of our bottom module from 128 to 256. Finally, we reduce the bases output stride from 8 to 4, to produce higher-quality bases. We achieve this by using P2 and P5 as the bottom module input. Table 4.7 shows the results. By adding semantic loss, detection and segmentation results are both improved. This is an interesting effect since the instance segmentation task itself does not improve the box AP. Although all tricks contribute to the improvements, we decide to not use larger basis resolution because it slows down the model by 10ms per image.

Alignment	R	M	AP	AP ₅₀	AP ₇₅
Nearest	28	7	30.5	53.0	31.6
	56	14	31.9	53.6	33.4
Bilinear	28	7	32.4	54.4	34.5
	56	14	33.6	54.6	35.6

TABLE 4.6. **Bottom Alignment:** Performance with 4 bases and bilinear top interpolation. Nearest represents the original RoIPool in Fast R-CNN [78] and bilinear is the RoIAlign in Mask R-CNN [2].

Bottom	Time (ms)	AP ^{bb}	AP	AP ₅₀	AP ₇₅
DeepLabV3+	76.5	38.8	33.6	54.6	35.6
+semantic	76.5	39.2	34.2	54.9	36.4
+128	78.5	39.1	34.3	54.9	36.6
+s/4	86.4	39.2	34.4	55.0	36.8
Proto-P3	85.2	39.0	34.4	54.9	36.8
Proto-FPN	78.8	39.1	34.4	54.9	36.8

TABLE 4.7. **Other improvements:** We use 56_4_14x14 with bilinear interpolation for all models. ‘+semantic’ is the model with semantic supervision as auxiliary loss. ‘+128’ is the model with bottom module channel size being 256. ‘+s/4’ means using P2,P5 as the bottom input. Decoders in DeepLab V3+ and YOLACT (Proto) are compared. ‘Proto-P3’ has channel width of 256 and ‘Proto-FPN’ of 128. Both are trained with ‘+semantic’ setting.

We also implement the protonet module in YOLACT [38] for comparison. We include a P3 version and an FPN version. The P3 version is identical to the one used in YOLACT. For the FPN version, we first change the channel width of P3, P4, and P5 to 128 with a 3×3 convolution. Then upsample all features to s/8 and sum them up. Following are the same as P2 version except that we reduce convolution layers by one. Auxiliary semantic loss is applied to both versions. As shown in Table 4.7, changing the bottom module from DeepLabv3+ to protonet does not modify the speed and performance significantly.

4.4.2 Main result

Quantitative results We compare BlendMask with Mask R-CNN [2] and TensorMask [60] on the COCO test-dev dataset³. We use 56_4_14 with bilinear top interpolation, the DeepLabV3+ decoder with channel width 256 and P3,

³To make fair comparison with TensorMask, the code base that we use for main result is maskrcnn_benchmark. Recently released Detectron2 fixed several issues of maskrcnn_benchmark (ROIAlign and paste_mask) in the previous repository and the performance is further improved.

Method	Backbone	Epochs	Aug.	Time (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Mask R-CNN[2]		12	✓	97.0	34.6	56.5	36.6	15.4	36.3	49.7
Mask R-CNN*		72	✓	97+	36.8	59.2	39.3	17.1	38.7	52.1
TensorMask[60]	R-50	72	✓	400+	35.5	57.3	37.4	16.6	37.0	49.1
BlendMask		12		78.5	34.3	55.4	36.6	14.9	36.4	48.9
BlendMask		36	✓	78.5	37.0	58.9	39.7	17.3	39.4	52.5
BlendMask*		36	✓	74.0	37.8	58.8	40.3	18.8	40.9	53.6
Mask R-CNN		12		118.1	36.2	58.6	38.4	16.4	38.4	52.1
Mask R-CNN*		36	✓	118+	38.3	61.2	40.8	18.2	40.6	54.1
TensorMask		72	✓	400+	37.3	59.5	39.5	17.5	39.3	51.6
SOLO [76]		72	✓	-	37.8	59.5	40.4	16.4	40.6	54.2
+deform convs [76]	R-101	72	✓	-	40.4	62.7	43.3	17.6	43.3	58.9
BlendMask		36	✓	101.8	38.4	60.7	41.3	18.2	41.5	53.3
BlendMask*		36	✓	94.1	39.6	61.6	42.6	22.4	42.2	51.4
+deform convs (interval = 3)		60	✓	105.0	41.3	63.1	44.6	22.7	44.1	54.5

TABLE 4.8. **Quantitative results** on COCO test-dev. We compare our BlendMask against Mask R-CNN and TensorMask. Mask R-CNN* is the modified Mask R-CNN with implementation details in TensorMask [60]. Models with ‘aug.’ uses multi-scale training with shorter side range [640, 800]. Speed for Mask R-CNN 1× and BlendMask are measured with `maskrcnn_benchmark` on a single 1080Ti GPU. BlendMask* is implemented with Detectron2, the speed difference is caused by different measuring rules. ‘+deform convs (interval = 3)’ uses deformable convolution in the backbone with interval 3, following [79].

Method	Backbone	NMS	Resolution	Time (ms)	AP ^{bb}	AP	AP ₅₀	AP ₇₅
YOLOACT	R-101	Fast	550 × 550	34.2	32.5	29.8	48.3	31.3
YOLOACT		Fast	700 × 700	46.7	33.4	30.9	49.8	32.5
BlendMask-RT		Batched	550 × *	47.6	41.6	36.8	61.2	42.4
Mask R-CNN	R-50	Batched	550 × *	63.4	39.1	35.3	56.5	37.6
BlendMask-RT				36.0	39.3	35.1	55.5	37.1

TABLE 4.9. **Real-time setting comparison** of speed and accuracy with other state-of-the-art methods on COCO **val2017**. Metrics for YOLOACT are obtained using their official code and trained model. Mask R-CNN and BlendMask models are trained and measured using **Detectron2**. Resolution 550 × * means using shorter side 550 in inference. Our fast version of BlendMask significantly outperforms YOLOACT in accuracy with *on par* execution time.

P5 input. Since our ablation models are heavily under-fitted, we increase the training iterations to 270K (3× schedule), tuning learning rate down at 180K and 240K. Following Chen *et al.*'s strategy [60], we use multi-scale training with shorter side randomly sampled from [640, 800]. As shown in Table 4.8, our BlendMask outperforms both the modified Mask R-CNN with deeper FPN and TensorMask using only half of their training iterations.

BlendMask is also more efficient. Measured on a V100 GPU, the best R-101 BlendMask runs at 0.07s/im, *vs.* TensorMask's 0.38s/im, *vs.* Mask R-CNN's 0.09s/im [60]. Furthermore, a typical running time of our blender module is merely 0.6ms, which makes the additional time for complex scenes nearly negligible. On the contrary, for two-stage Mask R-CNN with more expensive head computation, the inference time increases by a lot if the number of predicted instances grows.

Real-time setting We design a compact version of our model, BlendMask-RT, to compare with YOLOACT [38], a real-time instance segmentation method: i) the number of convolution layers in the prediction head is reduced to three, ii) and we merge the classification tower and box tower into one by sharing their features. We use Proto-FPN with four convolution layers with width 128 as the bottom module. The top FPN output P7 is removed because it has little effect on the detecting smaller objects. We train both BlendMask-RT and Mask R-CNN with the ×3 schedule, with shorter side randomly sampled from [440, 550].

There are still two differences in the implementation comparing to YOLOACT. YOLOACT resizes all images to square, changing the aspect ratios of inputs. Also, a paralleled NMS algorithm called Fast NMS is used in YOLOACT. We do not adopt these two configurations because they are not conventionally used in

instance segmentation researches. In YOLACT, a speedup of 12ms is reported by using Fast NMS. We instead use the Batched NMS in Detectron2, which could be slower than Fast NMS but does not sacrifice the accuracy. Results in Table 4.9 shows that *BlendMask-RT is 7ms faster and 3.3 AP higher than YOLACT-700*. Making our model also competitive under the real-time settings.

Qualitative results We compare our model with the best available official YOLACT and Mask R-CNN models with ResNet-101 backbone. Masks are illustrated in Figure 4.4. Our model yields higher quality masks than Mask R-CNN. The first reason is that we predicts 56×56 masks while Mask R-CNN uses 28×28 masks. Also our segmentation module mostly utilizes high resolution features that preserve the original aspect-ratio, where Mask R-CNN also uses 28×28 features.

Note that YOLACT has difficulties discriminating instances of the same class close to each other. BlendMask can avoid this typical leakage. This is because its top module provides more detailed instance-level information, guiding the bases to capture position-sensitive information and suppressing the outside regions.

4.4.3 Discussions

Comparison with Mask R-CNN Similar to Mask R-CNN, we use RoIPooler to locate instances and extract features. We reduce the running time by moving the computation of R-CNN heads before the RoI sampling to generate position-sensitive feature maps. Repeated mask representation and computation for overlapping proposals are avoided. We further simplify the global map representation by replacing the hard alignment in R-FCN [36] and FCIS [35] with our attention guided blender, which needs ten times less channels for the same resolution.

Another advantage of BlendMask is that it can produce higher quality masks, since our output resolution is not restricted by the top-level sampling. Increasing the RoIPooler resolution of Mask R-CNN will introduce the following problem. The head computation increases quadratically with respect to the RoI size. Larger RoIs requires deeper head structures. Different from dense pixel predictions, RoI foreground predictor has to be aware of whole instance-level information to distinguish foreground from other overlapping instances. Thus, the larger the feature sizes are, the deeper sub-networks is needed.

Furthermore, it is not very friendly to real-time applications that the inference time of Mask R-CNN is proportional to the number of detections. By

Method	Backbone	PQ	SQ	RQ	PQ Th	PQ St
Panoptic-FPN [1]	R-50	41.5	79.1	50.5	48.3	31.2
BlendMask		42.5	80.1	51.6	49.5	32.0
Panoptic-FPN [1]	R-101	43.0	80.0	52.1	49.7	32.9
BlendMask		44.3	80.1	53.4	51.6	33.2

TABLE 4.10. **Panoptic results** on COCO val2017. Panoptic-FPN results are from the official Detectron2 implementation, which are improved upon the original published results in [1].

contrast, our blender module is very efficient (0.6ms on 1080 Ti). The additional inference time required after increasing the number of detections can be neglected.

Our blender module is very flexible. Because our top-level instance attention prediction is just a single convolution layer, it can be an almost free add-on to most modern object detectors. With its accurate instance prediction, it can also be used to refine two-stage instance predictions.

4.4.4 Panoptic Segmentation

We use the semantic segmentation branch of Panoptic-FPN [1] to extend BlendMask to the panoptic segmentation, which is a combination of semantic and instance segmentation. Its metric panoptic quality (PQ) captures performance for all classes (stuff and things). We use annotations of COCO 2018 panoptic segmentation task. All models are trained on train2017 subset and tested on val2017. We train our model with the default FCOS [9] 3× schedule with scale jitter (shorter image side in [640, 800]). To combine instance and semantic results, we use the same strategy as in Panoptic-FPN, with instance confidence threshold 0.2 and overlap threshold 0.4.

Results are reported in Table 4.10. Our model is consistently better than its Mask R-CNN counterpart, Panoptic-FPN. We assume there are three reasons. First, our instance segmentation is more accurate, this helps with both thing and stuff panoptic quality because instance masks are overlaid on top of semantic masks. Second, our pixel-level instance prediction is also generated from a global feature map, which has the same scale as the semantic prediction, thus the two results are more consistent. Last but not least, since the our bottom module shares structure with the semantic segmentation branch, it is easier for the network to share features during the closely related multi-task learning.

4.4.5 More Qualitative Results

We visualize qualitative results of Mask R-CNN and BlendMask on the validation set in Fig. 4.5. Four sets of images are listed in rows. Within each set, the top row is the Mask R-CNN results and the bottom is BlendMask. Both models are based on the newly released Detectron2 with use R101-FPN backbone. Both are trained with the 3× schedule. The Mask R-CNN model achieves 38.6% AP and ours 39.5% AP.

Since this version of Mask R-CNN is a very strong baseline, and both models achieve very high accuracy, it is very difficult to tell the differences. To demonstrate our advantage, we select some samples where Mask R-CNN has trouble dealing with. Those cases include:

- Large objects with complex shapes (Horse ears, human poses). Mask R-CNN fails to provide sharp borders.
- Objects in separated parts (tennis players occluded by nets, trains divided by poles). Mask R-CNN tends to include occlusions as false positive or segment targets into separate objects.
- Overlapping objects (riders, crowds, drivers). Mask R-CNN gets uncertain on the borders and leaves larger false negative regions. Sometimes, it assigns parts to the wrong objects, such as the last example in the first row.

Our BlendMask performs better on these cases. 1) Generally, BlendMask utilizes features with higher resolution. Even for the large objects, we use stride-8 features. Thus details are better preserved. 2) As shown in previous illustrations, our bottom module acts as a class agnostic instance segmenter which is very sensitive to borders. 3) Sharing features with the bounding box regressor, our top module is very good at recognizing individual instances. It can generate attentions with flexible shapes to merge the fine-grained segments of bottom module outputs.

4.4.6 Evaluating on LVIS annotations

To quantify the high quality masks generated by BlendMask, we compare our results with on the higher-quality LVIS annotations [80]. Our model is compared to the best high resolution model we are aware of, recent PointRend [81], which uses multiple subnets to refine the local features to get higher resolution mask predictions. The description of the evaluation metric can be found in [81]. Table 4.11 shows that the evaluation numbers will improve further given more

Method	Backbone	resolution	COCO AP	LVIS AP*
Mask R-CNN	X101-FPN	28×28	39.5	40.7
PointRend	X101-FPN	224×224	40.9	43.4
BlendMask	R-101+dcni3	56×56	41.1	44.1

TABLE 4.11. **Comparison with PointRend.** Mask R-CNN and PointRend results are quoted from Table 5 of [81]. Our model is the last model in Table 4.8. Our model is 0.2 points higher on COCO and 0.7 points higher on LVIS annotations. Here LVIS AP* is COCO mask AP evaluated against the higher-quality LVIS annotations.

accurate ground truth annotations. Our method can benefit from the accurate bottom features and surpasses the high-res PointRend results.

4.5 Conclusion

We have devised a novel blender module for instance-level dense prediction tasks which uses both high-level instance and low-level semantic information. It is efficient and easy to integrate with different main-stream detection networks.

Our framework BlendMask outperforms the carefully-engineered Mask R-CNN without bells and whistles while being 20% faster. Furthermore, the real-time version BlendMask-RT achieves 34.2% mAP at 25 FPS evaluated on a single 1080Ti GPU card. We believe that our BlendMask is capable of serving as an alternative to Mask R-CNN [2] for many other instance-level recognition tasks.

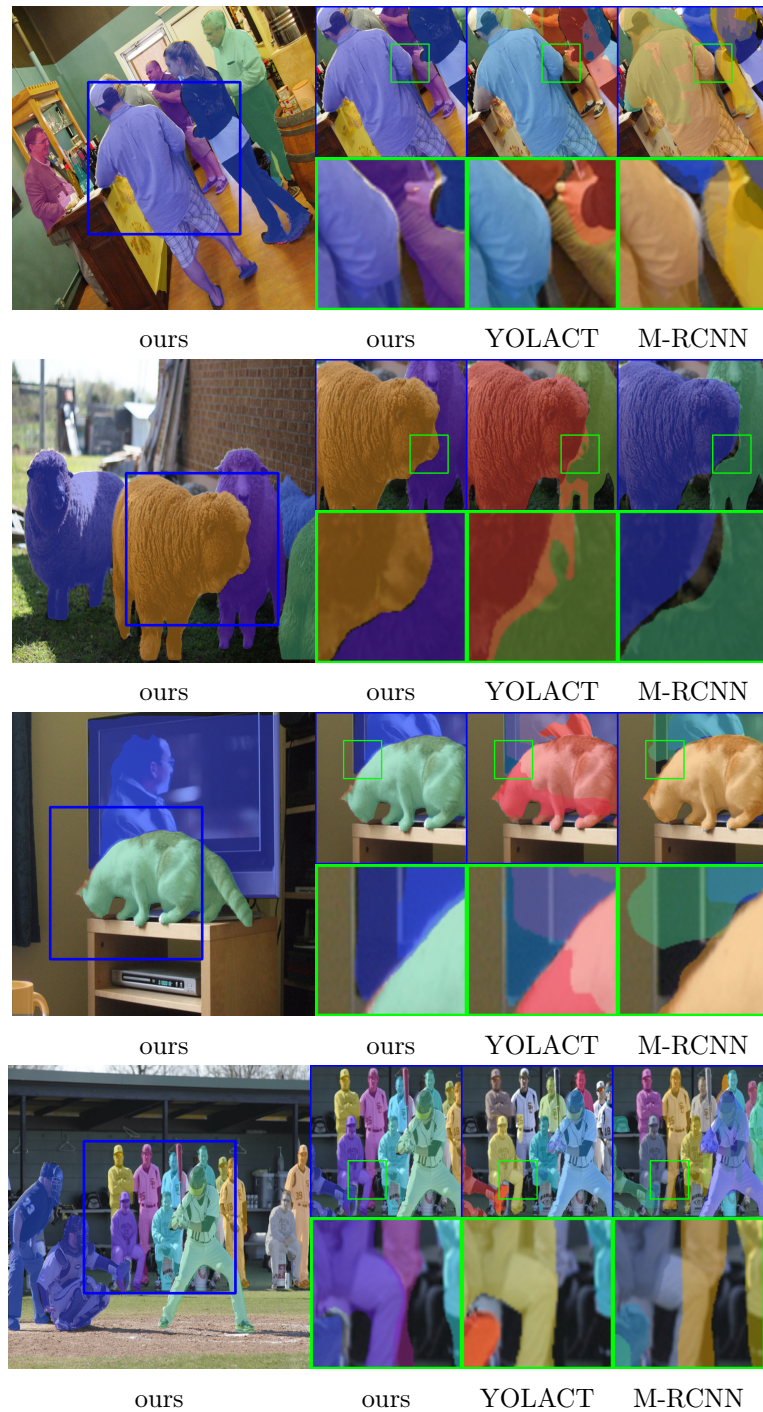


FIGURE 4.4. **Detailed comparison with other methods.** The large image on the left side is the segmentation result of our method. We further zoom in our result and compare against YOLACT [38] (31.2% mAP) and Mask R-CNN [2] (36.1% mAP) on the right side. Our masks are overall of higher quality.



FIGURE 4.5. Selected results of Mask R-CNN (top) and BlendMask (bottom). Both models are based on Detectron2. The Mask R-CNN model is the official $3\times$ R101 model with 38.6 AP. BlendMask model obtains 39.5 AP. Best viewed in digital format with zoom.

Statement of Authorship

Title of Paper	Segmenting Background for Free with Rank-1 Dynamic Convolution
Publication Status	<input type="checkbox"/> Published <input type="checkbox"/> Accepted for Publication <input type="checkbox"/> Submitted for Publication <input checked="" type="checkbox"/> Unpublished and Unsubmitted work written in manuscript style
Publication Details	Prepared for CVPR 2021 submission

Principal Author

Name of Principal Author (Candidate)	Hao Chen
Contribution to the Paper	Experiment framework design and implementation, experiment conduction, writing method part of the paper.
Overall percentage (%)	70%
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.
Signature	<hr/>
Date	19/10/2020

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

- the candidate's stated contribution to the publication is accurate (as detailed above);
- permission is granted for the candidate to include the publication in the thesis; and
- the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Zhi Tian
Contribution to the Paper	Discussion and writing revision.
Signature	<hr/>
Date	19/10/2020

Name of Co-Author	Chunhua Shen
Contribution to the Paper	Discussion and writing revision.
Signature	<hr/>
Date	19/10/2020

Chapter 5

DR1Mask: Segmenting Background for Free with Rank-1 Dynamic Convolution

5.1 Introduction

As discussed in 4, fully-convolutional one-stage networks have shown superior performance comparing to two-stage frameworks for instance segmentation as typically they can generate higher-quality predictions with less computation. In addition, their simple design opens up new opportunities for joint multi-task learning. In this chapter, we demonstrate that adding a single classification layer for semantic segmentation, fully-convolutional instance segmentation networks can achieve state-of-the-art panoptic segmentation quality. This is made possible by our novel dynamic rank-1 convolution (DR1Conv), a novel dynamic module that can efficiently merge high-level context information with low-level detailed features which is beneficial for both semantic and instance segmentation. Our model is 10% faster and 1 mAP more accurate than previous SOTA instance segmentation network BlendMask for instance segmentation.

Surprisingly, DR1Mask can perform panoptic segmentation by adding a single layer. It is the first panoptic segmentation framework that relying on a shared feature map for both instance and semantic segmentation according to our knowledge. Not only our framework is much more efficient, two times faster than two-branch approaches, the unified framework opens up opportunity for using the same context module to improve the performance for both tasks.

5.2 Background

The two-stage instance segmentation methods, most notably Mask R-CNN [2] have difficulties generating high quality features efficiently. They utilize a sub-network to segment the foreground instance from each proposals generated by the region proposal network. Thus, the predicted mask resolution is restricted by the second stage input size, typically 14×14 . Simply increasing this resolution will increase the computation overhead quadratically and make the network hard to train.

Fully-convolutional instance segmentation models can predict high-resolution masks efficiently because the features are shared across all predictions. The key breakthrough of these methods is the discovery of a dynamic module to merge of high level instance-wise features and the low level detail features. Recent successful methods such as YOLACT [38], BlendMask [10] and CondInst [82] choose to merge these two streams at the final prediction stage. The merging mechanism in these models is similar to a self-attention, computing an inner-product between the high-level and low-level features.

Similarly in semantic segmentation, researchers have found that incorporating higher level context information is crucial for the performance. Earlier attempts such as global average pooling [7] and ASPP [83] target on increasing the receptive field of single operations. More recent methods exploit second-order structures closely related to self-attention [84], [85].

These closely related structures indicate there is possibility to unify the context module for semantic and instance segmentation. The task of panoptic segmentation [1] introduces a new metric for joint evaluation of these two tasks. However, the dominate approaches still rely on two separate networks for stuff and thing segmentation. This approach has limited prospect both in research and in practice.

First, a model topping the instance segmentation leaderboard does not necessarily indicate it is most suitable for panoptic segmentation. The current metric for instance segmentation, mean average precision (mAP), is heavily biased towards whole object detection and not sensitive to subtle instance boundary misclassifications. To better discriminate the mask quality for instance segmentation, we need to include metrics from related segmentation tasks. A framework inherently compatible for both semantic and instance segmentation can provide better feature sharing, which is more promising to also yield better performance.

Second, adopting a unified model for these two tasks can substantially reduce the representation redundancy. A fully-convolutional structures are easier to be compressed and optimized for target hardware. This could open up opportunities for embedding panoptic segmentation algorithms in platforms with low computational resources or real-time requirement and be applied in fields such as autonomous driving, augmented reality and drone controls.

However, the features of segmentation branch in previous fully-convolutional models such as BlendMask [10] and CondInst [82] cannot be easily used for semantic segmentation because they typically contains very few channels, prohibiting them to encode rich class sensitive information. Furthermore, the parameters of their dynamic modules do not scale up to wider basis features, leading to very inefficient training and inference on a wider basis output such as 64. Thus, the dynamic modules are often limited to the final prediction module on very compact basis features.

In this work, we propose a novel unified, high-performing fully-convolutional panoptic segmentation framework called DR1Mask. This is made possible by our new way of merging higher level and local features for segmentation, dynamic rank-1 convolution (DR1Conv), which is efficient even on high dimensional feature maps and can be applied to the intermediate layers and increase the performance of both semantic and instance segmentation, leading to much more efficient computation. More specifically, our contributions are:

- DR1Conv, a novel computation efficient contextual feature merging operation that can improve the performance of instance and semantic segmentation at the same time.
- An efficient embedding for instance segmentation based on tensor decomposition, which adds almost no computation but can improve the instance segmentation prediction by 1 mAP.
- Unified semantic and instance segmentation framework Dr1Mask that achieves SOTA on both instance and panoptic segmentation benchmarks.
- Our model can produce complete panoptic segmentation results using only the running time for previous best networks to finish instance segmentation. Generating the extra stuff segmentation costs only one layer that is almost for free. It only takes half the running time for previous best fully-convolutional framework Panoptic-DeepLab [86] while scoring 8 PQ higher.

5.3 Preliminaries

Panoptic segmentation tackles the problem of classifying every pixel in the scene that assign different labels for different instances. Mainstream methods often take a two stream approach, using two separate networks to handle the stuff (semantic) and thing (instance) segmentation and focus on devising methods to fuse these two predictions [87], [88]. This is partly because previous state-of-the-art instance segmentation networks follow a two-stage paradigm which is not compatible with semantic segmentation pipeline. Panoptic-DeepLab [86] uses bottom-up structure for both tasks but still has two separate decoders. In addition, since it tackles instance segmentation with a bottom-up approach, the model cannot scale to complex dataset such as COCO and its performance falls behind two-stage methods. According to our knowledge, we are the first to use a single branch for both semantic and instance segmentation, with the only difference being the last prediction layers.

Dynamic networks Neural networks can dynamically modify its own weights or topology based on inputs on the fly. Dynamic networks are used in natural language processing to implement dynamic control flow for adaptive input structures [89]. The mechanism to mask out a subset of network connections is called dynamic routing, which has been used in various models for computation reduction [90], [91] and continual learning [92]. Dynamically changing the weights of network operations can be regarded as a special case of feature-wise transformation [93]. The most common form is channel-wise weight modulation in batch norm [94] and linear layers [95]. This is widely used to incorporate contextual information in vision language [95], image generation [94] and many other domains. Many of these dynamic mechanisms take a second-order form on the input and have very similar effect as self-attention.

Recently, many networks have adopted some variant of attention mechanism in both semantic and instance segmentation. For **semantic segmentation**, it is used to learn a context encoding [96] or pairwise relationship [97]. **Fully-convolutional instance segmentation** networks use a dynamic module to merge instance information with high-resolution features. The design usually involves applying a dynamically generated operator, which essentially is a generalized self-attention module. The module of YOLACT [38] takes a vector embedding as the instance-level information and applies a channel-wise weighted sum on the cropped features. BlendMask [10] extends the embedding into a 3D tensor, adding two spatial dimensions for the position-sensitive features of the instance. Most recently, CondInst [82] represents instance context with a set

of dynamically generated convolution weights. Different from these approaches which are only applied once during prediction, we aggregate multi-scale context information at different stages with an efficient dynamic module.

To keep the number of dynamic parameters in the instance embedding in a manageable scale, previous methods [10], [82] reduce the bottom feature channel width to a very small number, e.g. 4. Even though it is enough for class agnostic instance segmentation, this prohibits sharing the bottom output for semantic segmentation. Instead, we design efficient instance prediction module for much wider features, which in return also benefits the stuff segmentation quality.

BatchEnsemble [98] uses a low-rank factorization of convolution parameters for efficient model ensemble. In this paper, we adopt this technique to generate efficient dynamic modules. We factorize a weight matrix \mathbf{W}' as a static matrix \mathbf{W} and a dynamic low-rank mask \mathbf{M} ,

$$\mathbf{W}' = \mathbf{W} \circ \mathbf{M}, \text{ where } \mathbf{M} = \mathbf{a}\mathbf{b}^T, \quad (5.1)$$

where $\mathbf{W}', \mathbf{W}, \mathbf{M} \in \mathbb{R}^{m \times d}$, $\mathbf{a} \in \mathbb{R}^m$, $\mathbf{b} \in \mathbb{R}^d$ and \circ is element-wise product. This factorization is efficient for matrix-vector product computation. A forward pass with this dynamic layer can be formulated as

$$\begin{aligned} \mathbf{y} &= \mathbf{W}'\mathbf{x} = (\mathbf{W} \circ \mathbf{a}\mathbf{b}^T)\mathbf{x} \\ &= (\mathbf{W}(\mathbf{x} \circ \mathbf{a})) \circ \mathbf{b}, \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{y} \in \mathbb{R}^m$ are the input and output vector respectively. Thus, this matrix-vector product can be computed as element-wise multiplying \mathbf{a} and \mathbf{b} before and after multiplying \mathbf{W} respectively. This formulation also extends to other linear operations such as tensor product and convolution. Dusenberry *et al.* [99] use this factorization for efficient Bayesian posterior sampling in Rank-1 BNN.

5.4 DR1Mask: Unified Panoptic Segmentation Network

5.4.1 Dynamic Rank-1 Convolution

We extend the factorization in Equation 5.3 to convolutions. Different from BatchEnsemble [98] and Rank-1 BNN [99], we want the low-rank factors \mathbf{a} and \mathbf{b} to preserve position information of 2D images. Thus, we follow the

tensor representation in BlendMask [10] and densely predict two feature maps $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{C \times H \times W}$ as the dynamic low-rank factors. For different locations, we generate different dynamic parameters \mathbf{W}'_{hw} from the corresponding locations of \mathbf{A}, \mathbf{B} . We apply dynamic matrix-vector multiplication at position (h, w) as

$$\mathbf{y}_{hw} = \mathbf{W}'_{hw} \mathbf{x}_{hw} = (\mathbf{W}(\mathbf{x}_{hw} \circ \mathbf{a}_{hw})) \circ \mathbf{b}_{hw}, \quad (5.2)$$

where \mathbf{a}_{hw} and \mathbf{b}_{hw} are elements in the dynamic tensors \mathbf{A} and \mathbf{B} . This can be interpreted as element-wise multiplying the context tensors before and after the static linear operator.

Extending this to convolution, the dynamic rank-1 convolution (DR1Conv) $\text{Conv}_{\mathbf{W}'}$ with static parameters \mathbf{W} at location (h, w) takes an input patch \mathbf{x}_{hw} and dynamic features \mathbf{a}_{hw} and \mathbf{b}_{hw} and outputs patch \mathbf{y}_{hw} :

$$\mathbf{y}_{hw} = (\text{Conv}_{\mathbf{W}}(\mathbf{x}_{hw} \circ \mathbf{a}_{hw})) \circ \mathbf{b}_{hw}. \quad (5.3)$$

DR1Conv has very efficient tensorized computation. Given \mathbf{X} and two dynamic tensors \mathbf{A}, \mathbf{B} with the same shape, DR1Conv outputs \mathbf{Y} with the following equation:

$$\mathbf{Y} = \text{DR1Conv}_{\mathbf{A}, \mathbf{B}}(\mathbf{X}) = \text{Conv}(\mathbf{X} \circ \mathbf{A}) \circ \mathbf{B}, \quad (5.4)$$

where all tensors have the same size. This is implemented as element-wise multiplying the dynamic factors \mathbf{A}, \mathbf{B} before and after the static convolution respectively. The structure of DR1Conv is shown in Figure 5.1.

We argue that DR1Conv is essentially different from naive channel-wise modulation. The two related factors \mathbf{A}, \mathbf{B} combine to gain much stronger expressive power while being as efficient in computation. As shown in later ablation experiment results in Table 5.1, the combination of these two dynamic factors yields higher improvement than the increments of the two factors individually added together.

5.4.2 DR1Mask for Instance Segmentation

DR1Conv can be integrated into fully-convolutional instance segmentation networks. We base our model on the two-stream framework of YOLACT [38] and BlendMask [10] and use DR1Conv as the contextual block to merge instance-level and segmentation features. Besides the usual output of previous methods, bounding box \mathbf{p}_i and instance embedding \mathbf{e}_i for instance i , our top-down branch

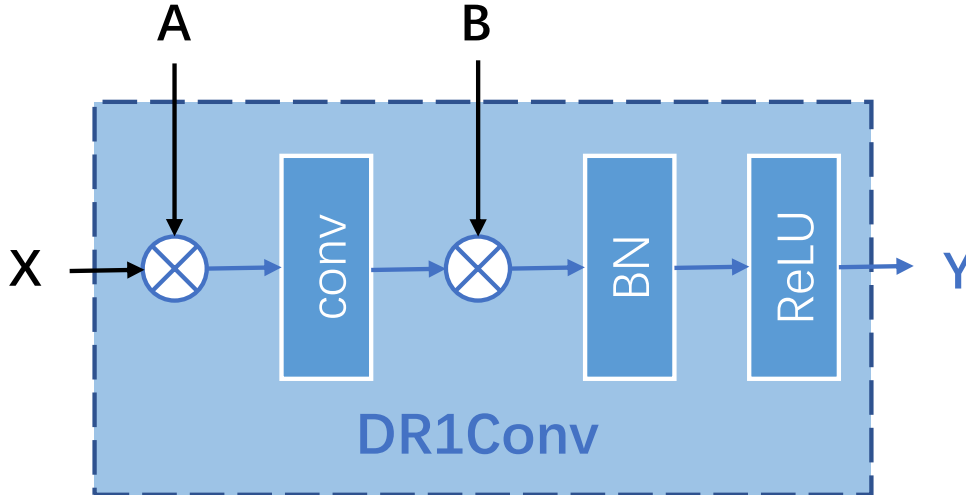


FIGURE 5.1. Diagram of dynamic rank-1 convolution (DR1Conv). \otimes denotes element-wise multiplication. Tensors \mathbf{A} , \mathbf{B} are the dynamic factors encoding the contextual information. Each modulated the channels of the feature before and after the convolution operation. \mathbf{X} is the input and \mathbf{Y} is the output. All tensors have the same size.

also generates a multi-scale conditional feature pyramid $\{\mathbf{C}_l = [\mathbf{A}_l, \mathbf{B}_l]\}$. The bottom-up branch aggregates the information from the backbone pyramid and $\{\mathbf{C}_l\}$ to generate the final prediction. The framework of our model, DR1Mask is shown in Figure 7.1.

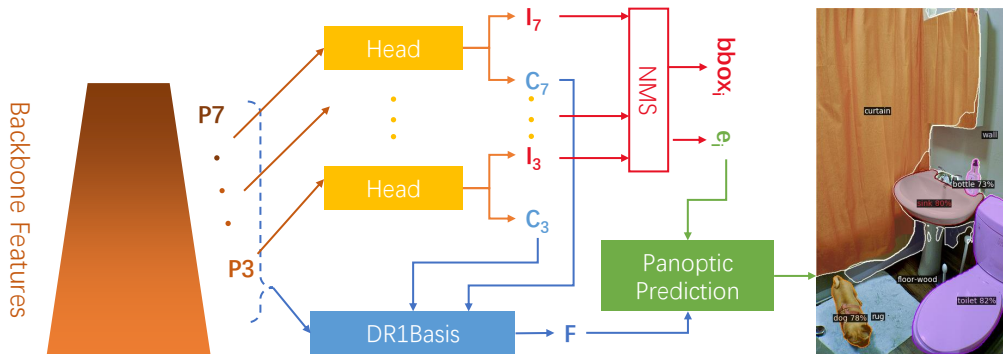


FIGURE 5.2. **DR1Mask pipeline** Our model follows the typical two-stream framework with one branch extracting instance-level features (heads) and the other for pixel-level prediction (DR1Basis). These two branches are connected by DR1Basis which is an inverted pyramid network consists of DR1Convs. A unified prediction layer can be appended to directly generate the panoptic segmentation output.

The three key components of our model are the top layer, DR1Basis and the unified prediction layer. We will introduce them one by one.

Top Layer

The top layer produces the instance-wise contextual information $\{\mathbf{C}_l\}$ and the instance embeddings $\{\mathbf{E}_l\}$. It is a single convolution layer added to the object detection tower of FCOS [9]. The conditional feature pyramid $\{\mathbf{C}_l\}$ has the same resolution as corresponding backbone FPN outputs. Given FPN output \mathbf{P}_l , the top-down branch computes these features with the following equation:

$$\{\mathbf{C}_l, \mathbf{E}_l\} = \text{Top}(\text{Tower}(\mathbf{P}_l)), l = 3, 4, \dots, 7 \quad (5.5)$$

where \mathbf{C}_l , \mathbf{E}_l and \mathbf{P}_l are tensors with the same spatial resolution. \mathbf{C}_l can be further split into the two dynamic tensors \mathbf{A}_l and \mathbf{B}_l in Equation 5.4.1. They are the dynamic factors in DR1Basis which we will later introduce in Section 5.4.2.

The densely predicted \mathbf{E}_l along with other instance features such as class labels and bounding boxes are later filtered into a set containing only the positive proposals, $\{\mathbf{e}_i\}$. The instance embedding can take various forms, a vector [38], a tensor [10] or a set of convolution weights [82]. We will introduce our novel prediction module in Section 5.4.2.

DR1Basis

We name our bottom-up branch DR1Basis because it is built with DR1Conv as the basic block. It aggregates the FPN features $\{\mathbf{P}_l\}$ and contextual features $\{\mathbf{C}_l\}$ and produces the basis features for segmentation prediction like an inverted pyramid. Starting from the highest level features with the smallest resolution, at each step l for $l = 7, 6 \dots, 3$, it uses a DR1Conv to merge \mathbf{P}_l and \mathbf{C}_l and upsample the result by a factor of 2:

$$\mathbf{F}_l = \text{DR1Conv}_{\mathbf{A}_l, \mathbf{B}_l}(\text{Conv}_{3 \times 3}(\mathbf{P}_l) + \uparrow_2(\mathbf{F}_{l+1})), \quad (5.6)$$

where $\mathbf{F}_8 = 0$ and \uparrow_2 is upsampled by a factor of 2 and $\mathbf{A}_l, \mathbf{B}_l$ are from \mathbf{C}_l split evenly along the channel dimension. We first reduce the channel width of \mathbf{P}_l with a 3×3 convolution. Then the channel width is kept the same throughout the computation. In practice, we found that for instance segmentation, 32 channels are enough ¹. The computation graph for DR1Basis is shown in Figure 5.3.

¹For semantic segmentation, performance get even better with 64 channels

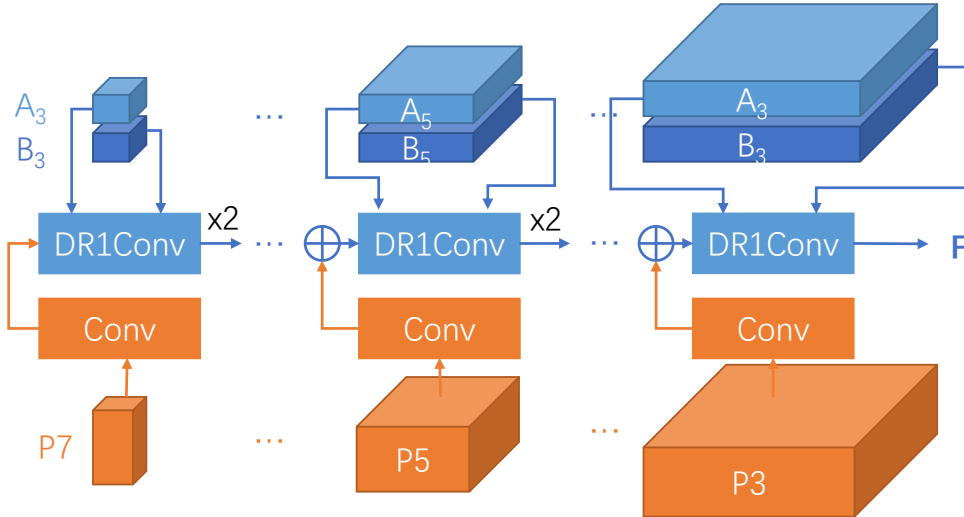


FIGURE 5.3. **DR1Basis** is an inverted pyramid network which consists a sequence of DR1Convs.

This makes our DR1Basis very compact, using only 1/4 of the channels of the corresponding block in BlendMask. In experiments, we found this makes our model 6% faster while achieving even higher accuracy.

Instance Prediction Module

Similar to other crop-then-segment models, we first crop a region \mathbf{F}_i from the DR1Basis output \mathbf{F} according to the detected bounding box \mathbf{b}_i using RoIAlign [2]. Then the crops from different bases are combined into the final instance foreground mask guided by the instance embedding \mathbf{e}_i . YOLACT [38] simply performs a channel-wise weighted sum with a vector embedding. BlendMask [10] improved the mask quality by extending the embedding spatially but the number of bases is limited.

In practice, we provide two different choices targeting different scenarios. For instance segmentation, we learn a low-rank decomposition for the attention tensor in [10]. Full attention in BlendMask has $4 \times 14 \times 14$ parameters. The first dimension is the number of bases and the last two are spatial resolution. There are two issues with this approach. First, 196 parameters per channel prohibits applying this to a wider basis output. Second, using a linear layer to generate so many parameters is not very efficient. In addition, noticing that the attention maps generated by BlendMask are usually very coarse (see Figure 5.4), we assume the representation is largely redundant.

We propose a new instance prediction module, called **factored attention**, which has less parameter but can take in much wider basis features. We split the embedding into two parts $\mathbf{e}_i = [\mathbf{t}_i : \mathbf{s}_i]$. The first step is projection. We use

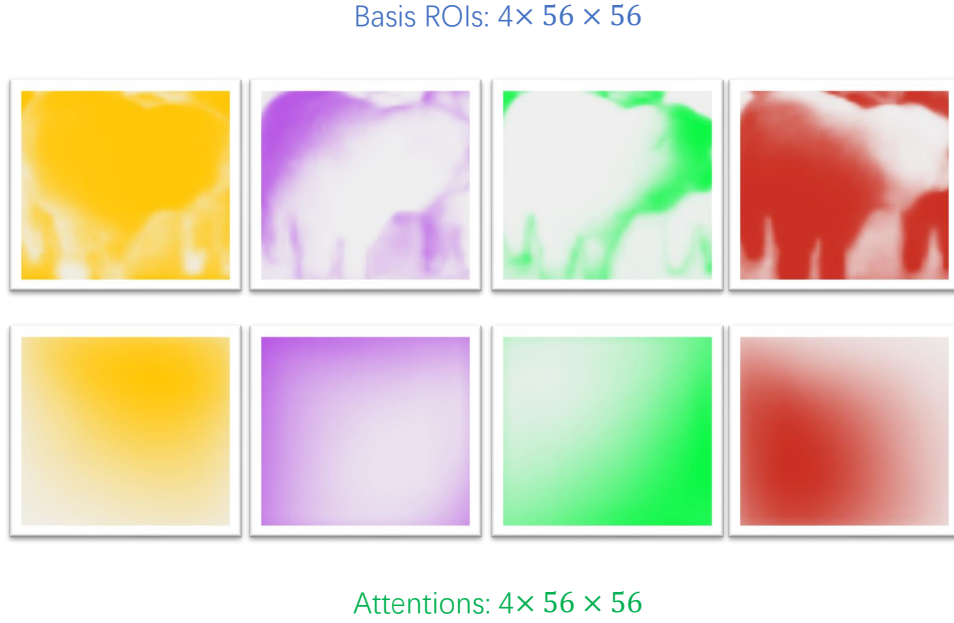


FIGURE 5.4. **An example of the position sensitive attention tensor** Illustrated are the mask bases and their corresponding attention maps generated by BlendMask [10]. Even though have a relatively large resolution, the attention usually does not contain fine-grained patterns indicating there is redundancy in its representation.

\mathbf{t}_i as the (flattened) weights of a 1×1 convolution which projects the cropped bases \mathbf{F}_i with D channels and spatial resolution 56×56 into a lower dimension, K :

$$\mathbf{R}_i = \mathbf{T}_i * \mathbf{F}_i \quad (5.7)$$

where \mathbf{T}_i is the reshaped convolution weights, $*$ is the convolution operator and \mathbf{R}_i is a tensor with shape $K \times 56 \times 56$. This makes \mathbf{t}_i a vector of length $D \times K$. We choose $K = 4$ to match the design choice of BlendMask. Similar to BlendMask, \mathbf{R}_i and the full-attention \mathbf{Q}^i are element-wise multiplied and summed along the first dimension to get the instance mask result.

To get an efficient attention representation, we decompose the $4 \times 14 \times 14$ full attention \mathbf{Q}_i in the following way. First, we split it along the first dimension into $\{\mathbf{Q}_{ki} | k = 1, \dots, 4\}$. Then each $\mathbf{Q}_{ki} \in \mathbb{R}^{14 \times 14}$ is decomposed into two matrices $\mathbf{U}_k, \mathbf{V}_k \in \mathbb{R}^{4 \times 14}$ and a diagonal matrix $\Sigma_{ki} \in \mathbb{R}^{4 \times 4}$:

$$\mathbf{Q}_{ki} = \mathbf{U}_k^T \Sigma_{ki} \mathbf{V}_k. \quad (5.8)$$

Thus, only the diagonal values in σ_k^i are generated from the top module, \mathbf{U}_k

and \mathbf{V}_k are shared by all instances and learned as network parameters. This reduces the instance embedding parameters from 784 to 16 while still enabling us to form position-sensitive attention shapes. For each k and row d in \mathbf{U}_k and \mathbf{V}_k , the outer product $\mathbf{u}_{kd}^T \mathbf{v}_{kd}$ can be considered as one of the components of \mathbf{Q}_{ki} . We visualize all components learned by our network in Figure 5.5. The factored attention has similar flexibility as the full attention in Figure 5.4 but much more parameter efficient.

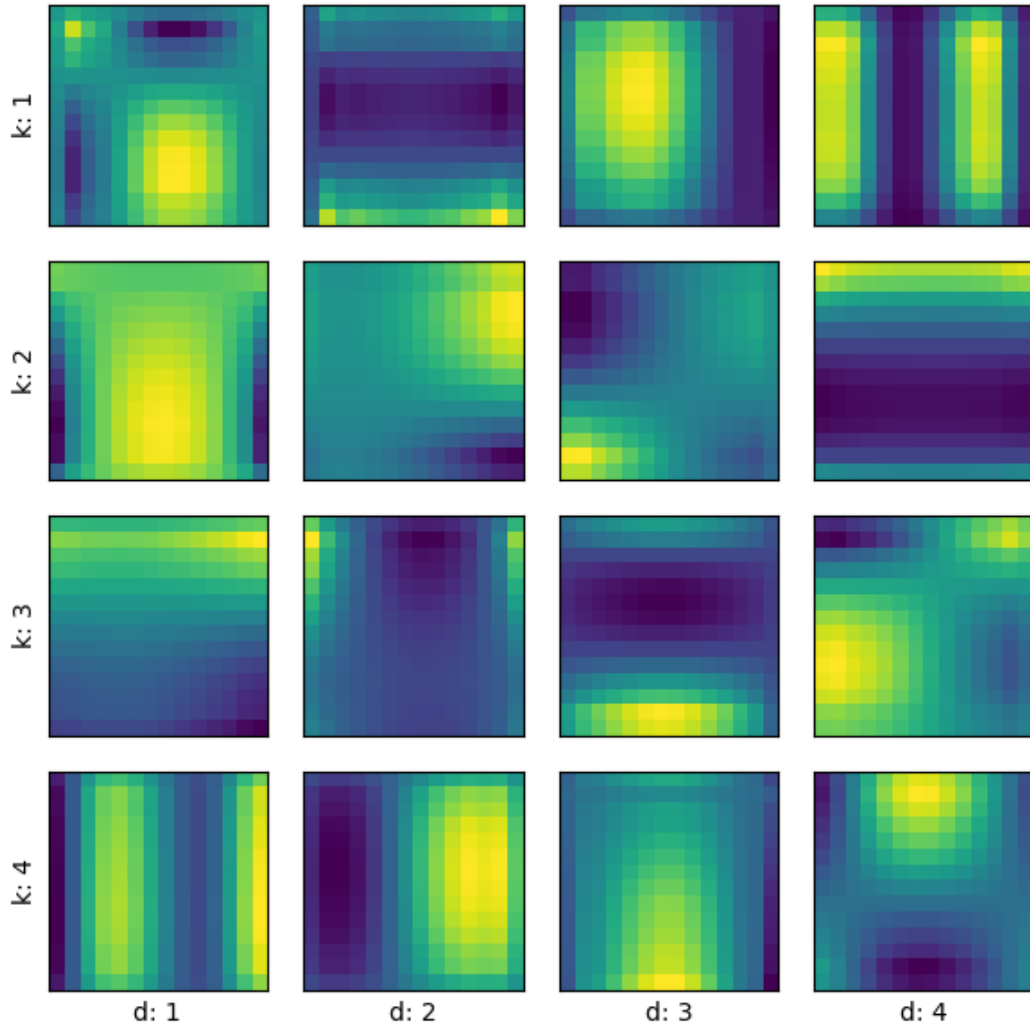


FIGURE 5.5. **Factored attention components** The k th row corresponds to k th basis $\mathbf{R}_i k$ and the d th column corresponds to the d th row of \mathbf{U}_k and \mathbf{V}_k . Each attention map can be considered as the k th slice of \mathbf{Q}_{ki} generated by a one-hot instance embedding with the d th element valued 1.

For panoptic segmentation, we compute the mean of all embedding vectors for the same instance. For details, please refer to Section 5.4.3.

5.4.3 Unified Panoptic Segmentation Training

We add minimal modifications to DR1Mask for panoptic segmentation: a unified panoptic segmentation layer which is simply a 1×1 convolution f_{pano} transforming the output \mathbf{F} of DR1Basis into panoptic logits with C channels. The first C_{stuff} channels are for semantic segmentation and the rest C_{thing} channels are for instance segmentation.

We split the weights for f_{pano} along the columns into two matrix $\mathbf{W}_{pano} = [\mathbf{W}_{stuff}, \mathbf{W}_{thing}]$. The first $D \times C_{stuff}$ parameters \mathbf{W}_{stuff} are static parameters. C_{stuff} is a constant equals to the number of stuff classes in the dataset, i.e., 53 for COCO dataset.

The rest $D \times C_{thing}$ parameters \mathbf{W}_{thing} are dynamically generated. During training, we choose C_{thing} to be the number of ground truth instances in the sample. For each target instance i , there can be $N_i \geq 0$ instance embeddings $\{\mathbf{e}_i^{(n)}\}$ in the network assigned to it. For instance segmentation, these embeddings are supervised separately. However, for panoptic segmentation, we have to map them into a single embedding $\bar{\mathbf{e}}_i \in \mathbb{R}^D$. Then the C_{thing} embeddings are concatenated into the dynamic weights \mathbf{W}_{thing} :

$$\mathbf{W}_{thing} = [\bar{\mathbf{e}}_1, \bar{\mathbf{e}}_2, \dots, \bar{\mathbf{e}}_{C_{thing}}]. \quad (5.9)$$

The panoptic prediction can be computed with a matrix multiplication $\mathbf{Y}_{pano} = \mathbf{W}_{pano}^T \mathbf{F}$.

The position sensitive attention introduced in Section 5.4.2 can also be integrated into an end-to-end framework. However, we find that using position sensitive attention causes the bases to encode too much instance-wise position information which leads to sub-par semantic segmentation results.

We have to be careful about directly applying the backbone features from instance segmentation networks for semantic segmentation. To align the features for FPN computation, it is common for instance segmentation networks to pad along the borders to make the feature resolution divisible by the stride so that feature sizes will be consistent after downsampling/upsampling. In practice, we observe that border padding can cause inaccurate segmentation near the padded edges. We fix this by changing the input size divisibility from 32 to 4. To keep the features aligned after upsampling, we crop the right and bottom edges of the upsampled features if the sizes does not match.

5.5 Experiments

5.5.1 Dataset and Implementation Details

We evaluate our algorithm on the MSCOCO 2017 dataset [70]. It contains 123K images with 80 thing categories and 53 stuff categories for instance and semantic segmentation respectively. We train our models on the train split with 115K images and carry out ablation studies on the validation split with 5K images. The final results are reported on the `test-dev` split, whose annotations are not publicly available.

Following the common practice, ImageNet pre-trained ResNet-50 [71] is used as our backbone network. Channel width of the DR1Basis is 32. For ablation studies unless specified, all the networks are trained with the $1\times$ schedule of BlendMask [10], i.e., 90K iterations, batch size 16 on 4 GPUs, and base learning rate 0.01 with linear warm-up of 1k iterations. The learning rate is reduced by a factor of 10 at iteration 60K and 80K. Input images are resized to have shorter side randomly selected between [640, 800] and longer side at maximum 1333. The auxiliary semantic segmentation loss weight is 0.3. All hyperparameters are set to be the same with BlendMask [10]. We implement our models based on the open-source project AdelaiDet ².

To measure the running time, we run the models with batch size 1 on the whole COCO val2017 split using one GTX 1080Ti GPU. We calculate the time from the start of model inference to the time final predictions are generated, including the post-processing stage.

5.5.2 Ablation Experiments

Effectiveness of dynamic factors DR1Conv has two dynamic components **A** and **B**. As shown in Equation 5.4.1, they each has the effect of channel-wise modulation pre-/post- convolution respectively. By removing both of them, our basis module becomes a vanilla FPN. We train networks with each of these two components masked out. The instance prediction module used for both tasks is the vector embedding in YOLACT [38]. Results are shown in Table 5.1 and Table 5.2. **A** and **B** each has slight improves on AP_{50} and AP_{75} but combining them improves all metrics significantly. Table 5.2 shows that DR1Conv can improve both the thing and stuff segmentation qualities.

Context feature position The contextual information **C** is computed with the features from the box tower of FCOS [9], same as the features for instance

²<https://github.com/aim-uofa/AdelaiDet>

Method	AP	AP ₅₀	AP ₇₅
Baseline	34.7	55.5	36.8
w/ A	34.9	55.9	36.8
w/ B	34.9	55.6	37.0
w/ A, B	35.2	56.1	37.5

TABLE 5.1. Instance segmentation results with the dynamic factors removed.

Method	PQ	PQ Th	PQ St
w/o A, B	38.7	45.9	28.0
w/ A, B	40.0	46.8	29.9

TABLE 5.2. Panoptic segmentation results with the dynamic factors removed.

embedding. We assume this is important to keep the instance representation consistent. To examine this effect, we move the top layer for contextual information computation to the FPN outputs and class towers. Results are shown in Table 5.3. With the features from FPN outputs, **A** and **B** are bot computed directly from **X** (see Figure 5.4.1). This badly hurts the segmentation performance, AP₇₅ especially, even worse than the vanilla baseline without dynamism. This proves that the correspondence between instance embedding and the contextual information is important.

Efficiency of the factored attention We compare the performance and efficiency of different instance prediction modules in Table 5.4. Our factored attention module is almost as efficient as the channel-wise modulation and can achieve the best performance.

We also notice that **border padding** can affect the performance of semantic segmentation performance. The structure difference between our basis module and common semantic segmentation branch is that we have incorporated high-level feature maps with strides 64 and 128 for contextual information embedding. We assume this leads to a dilemma over the padding size. Smaller padding size will make the features spatially misaligned across levels. However, too large padding size will make it very inefficient. Making a 800×800 image divisible by 128 will increase 25% unnecessary computation cost on the borders. We tackle this problem by introducing a new upsampling strategy with is spatially aligned with the downsampling mechanism of strided convolution and reduce the padding size to the output stride, i.e. 4 in our implementation. Results are shown in Table 5.5. Our aligned upsampling strategy requires the minimal padding size while being significantly better in semantic segmentation quality PQSt.

Position	AP	AP ₅₀	AP ₇₅
None	34.7	55.5	36.8
FPN	34.2	55.5	36.0
class tower	34.5	55.6	36.5
box tower	35.2	56.1	37.5

TABLE 5.3. Instance segmentation results with the contextual information from different positions.

Attention	Time (ms)	AP	AP ₅₀	AP ₇₅
Vector	68.7	35.2	56.1	37.5
Full	72.0	36.2	56.7	38.7
Factored	69.2	36.3	56.9	38.8

TABLE 5.4. Comparison of different instance prediction modules. Vector is channel-wise vector attention in YOLACT [38]; full is the 3D full attention tensor in BlendMask [10] and factored is the factored attention introduced in Section 5.4.2.

Choosing a proper **channel width of the basis module** also important for the panoptic segmentation accuracy. A more compact basis output of size 32 does not affect the class agnostic instance segmentation result but will leads to much worse semantic segmentation quality, which has to discriminate 53 different classes. To accurately measure the influence of different channel widths and making sure all models are fully trained, we train different models with the 3x schedule. Results are shown in Table 5.6. Doubling the channel width from 32 to 64 can improve the semantic segmentation quality by 2.1. However, a wider channel of 128 does not have comparable increment in performance.

Position sensitive attention for panoptic segmentation Unfortunately, even though beneficial for instance segmentation, we discover that position sensitive attention has negative effect for panoptic segmentation. It enforces the bases to perform position sensitive encoding on all classes, even for stuff regions, which is unnecessary and misleading. The panoptic performance for different instance prediction modules are shown in Table 5.7. Using factored attention makes the semantic segmentation quality drop 2.6.

5.5.3 Main Results

Quantitative results We compare DR1Mask with recent instance and panoptic segmentation networks on the COCO `test-dev` split. We increase the training iterations to 270K (3× schedule), tuning learning rate down at 180K and 240K. All instance segmentation models are implemented with the same code

Divisibility	PQ	PQ Th	PQ St
32	39.5	46.5	28.8
128	39.9	46.5	30.0
4 w/ aligned	40.0	46.8	29.9

TABLE 5.5. Comparison of different padding strategies for panoptic segmentation. The baseline method is padding to 32x, divisibility of C5 from ResNet. Padding to 128x is for the divisibility of DR1Basis. 4 w/ aligned is padding the input size to 4x and applying our aligned upsampling strategy.

Width	PQ	PQ Th	PQ St
32	41.8	49.1	30.7
64	42.9	49.5	32.9
128	42.8	49.5	32.8

TABLE 5.6. Comparison of different channel widths in DR1Basis for panoptic segmentation. All models are with a ResNet-50 backbone and are trained with the 3x schedule.

base, Detectron2³ and the running time is measured on the same machine with the same setting. We use multi-scale training with shorter side randomly sampled from [640, 800]. Results on instance segmentation and panoptic segmentation are shown in Table 5.8 and Table 5.9 respectively. Our model achieves the best performance and is also the most efficient among them all. For panoptic segmentation, DR1Mask is two times faster than the main stream separate frameworks. Particularly, the running time bottleneck for UPSNet [87] is not in the backbone but in the stuff/thing prediction branches and the final fusion stage, which makes the R-50 model almost as costly as the R-101 DCN model.

5.5.4 Conclusions

In this chapter, we use DR1Conv as an efficient way to compute second-order relations between features. This is remotely related to self-attention, where a score to control feature aggregation is computed with an inner product. Recently, low rank approximations have also been proposed for efficient self-attention computation [100], [101]. Our method is different from the self-attention based context modules such as non-local block [84] and axial-attention [102] in that we are aggregating different semantic rather than spatial information. In another word, not using the same feature for query and key computation is crucial to our approach.

³<https://github.com/facebookresearch/detectron2>

Method	PQ	PQ Th	PQ St
Vector	40.0	46.8	29.9
Factored	39.0	46.8	27.3

TABLE 5.7. Position sensitive attention for panoptic segmentation. Vector is the baseline model with vector instance embeddings. Factored is the position sensitive factored attention in Section 5.4.2.

Method	Backbone	Time (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Mask R-CNN [2]	R-50	74	37.5	59.3	40.2	21.1	39.6	48.3
BlendMask [10]		73	38.1	59.5	41.0	21.3	40.5	49.3
CondInst [82]		72	38.7	60.3	41.5	20.7	41.0	51.3
DR1Mask		69	38.3	59.6	41.2	21.1	40.4	50.0
Mask R-CNN[2]	R-101		38.8	60.9	41.9	21.8	41.4	50.5
BlendMask		94	39.6	61.6	42.6	22.4	42.2	51.4
CondInst		93	40.1	61.9	43.0	21.7	42.8	53.1
DR1Mask		89	39.8	61.6	42.9	21.9	42.4	51.9
DR1Mask*		98	41.2	63.2	44.5	22.6	43.8	54.7

TABLE 5.8. Instance segmentation results on COCO test-dev. Models with * have deformable convolutions in the backbone.

Method	Backbone	Time (ms)	PQ	SQ	RQ	PQ Th	PQ St
Panoptic-FPN [1]	R-50	89	41.5	79.1	50.5	48.3	31.2
UPSNet [87]		233	42.5	78.2	52.4	48.6	33.4
SOGNet [88]		248	43.7	78.7	53.5	50.6	33.1
Panoptic-DeepLab		149	35	-	-	-	-
BlendMask		96	42.5	80.1	51.6	49.5	32.0
DR1Mask		79	42.9	79.8	52.0	49.5	32.9
Panoptic-DeepLab	Xception-71	-	41.4	-	-	45.1	35.9
Panoptic-FPN [1]	R-101	111	43.6	79.7	52.9	51.0	32.6
BlendMask		117	44.5	80.7	53.8	52.1	33.0
UPSNet*		237	46.3	79.8	56.5	52.7	36.8
DR1Mask		99	44.5	80.7	53.8	51.7	33.5
DR1Mask*		109	46.1	81.5	55.3	53.1	35.5

TABLE 5.9. Panoptic results on COCO. R-50 models are evaluated on val2017 split and R-101 models are evaluated on test-dev. All models are evaluated with the official code and the best models publicly available on the same machine. Panoptic-DeepLab does not provide trained models on COCO. We measure its speed by running the Cityscapes pretrained model on COCO val2017. Models with * have deformable convolutions in the backbone.


In recent years, computer vision technologies have been improved both on the research and engineering sides. Researchers are breaking the traditional

boundaries of different vision tasks with deep learning, as simpler models have better ability to tackle multiple tasks. We wish this work to serve as an example to unify the semantic/instance/panoptic segmentation tasks.

Statement of Authorship

Title of Paper	Architecture Search of Dynamic Cells for Semantic Video Segmentation
Publication Status	<input checked="" type="checkbox"/> Published <input type="checkbox"/> Accepted for Publication <input type="checkbox"/> Submitted for Publication <input type="checkbox"/> Unpublished and Unsubmitted work written in manuscript style
Publication Details	WACV 2020; 1959-1968


Principal Author

Name of Principal Author (Candidate)	Hao Chen
Contribution to the Paper	Experiment framework design and implementation, experiment conduction, writing method part of the paper.
Overall percentage (%)	50%
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.
Signature	 Date 02/10/2020


Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

- the candidate's stated contribution to the publication is accurate (as detailed above);
- permission is granted for the candidate to include the publication in the thesis; and
- the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Vladimir Nekrasov
Contribution to the Paper	Searching space design, method discussion and writing majority of the paper.
Signature	 Date 05/10/2020

 9D05322315B54D3...

Name of Co-Author	Chunhua Shen
Contribution to the Paper	Discussion and writing revision.
Signature	 Date 02/10/2020

Name of Co-Author	Ian D. Reid		
Contribution to the Paper	Discussion and writing revision.		
Signature		Date	2/10/2020

Chapter 6

Dynamic Cell Search for Semantic Video Segmentation

6.1 Introduction

In semantic video segmentation the goal is to acquire consistent dense semantic labelling across image frames. To this end, recent approaches have been reliant on manually arranged operations applied on top of static semantic segmentation networks - with the most prominent building block being the optical flow able to provide information about scene dynamics. Related to that is the line of research concerned with speeding up static networks by approximating expensive parts of them with cheaper alternatives, while propagating information from previous frames. In this chapter, we attempt to come up with generalisation of those methods, and instead of manually designing contextual blocks that connect per-frame outputs, we propose a neural architecture search solution, where the choice of operations together with their sequential arrangement are being predicted by a separate neural network. We showcase that such generalisation leads to stable and accurate results across common benchmarks, such as CityScapes and CamVid datasets. Importantly, the proposed methodology takes only 2 GPU-days, finds high-performing cells and does not rely on the expensive optical flow computation.

6.2 Background

Human beings are well-equipped by evolution to quickly observe changes in dynamic environments. From merely few seconds of studying an unknown scene, we are able to coherently map out its main constituents. In contrast, static semantic segmentation networks would perform poorly in such conditions, and may as well produce contradictory predictions across the frames. Therefore,

the question arises of how to make the static models suitable for segmenting continuously evolving scenes?

One well-known approach would be to use the optical flow that describes the motion in the scene between adjacent frames [20], [21]. The optical flow calculation tends to be expensive and also comes with several notable disadvantages, among which its inability to deal with occlusions and newly appeared objects. Nevertheless, as shown by Gadde *et al.* [21], a relatively poor estimate of the optical flow may still carry significant benefits, not the least of which lies in computational savings.

Alternatively, one may choose to model which information must be propagated across the frames, e.g. with the help of a recurrent neural network with memory units [24]. Even more biologically plausible are the models that compute different features at various time-scales [103], in a vein similar to neural spikes. Naturally, this comes with its own set of disadvantages, most notably the difficulty of choosing an appropriate scheduling regime for updating individual parts of the network.

Yet another complementary line of work focuses on approximating an expensive per-frame forward pass with cheaper alternatives: e.g. Li *et al.* [25] predicted local filters to be applied on the segmentation prediction from the previous frame, while Jain *et al.* [22] used a larger network for key frames and directly employed a smaller one for consecutive frames. Such savings may allow to re-use more expensive optical flow methods without a significant slowdown, but the choice of key frames can be crucial and not readily justifiable.

Looking closely at the aforementioned approaches for video semantic segmentation, one may notice an easily discernible pattern: a typical video segmentation network predicts a labelling of the current frame based on the information propagated from the previous one and hidden representations of the current one (Fig. 6.1). While seemingly obvious, it possesses certain variations depending on the goal - e.g. whether efficiency, or real-time performance is desired. Importantly, what we would like to emphasise here is that, while technically sound, all the current approaches have been manually designed and have not considered any interplay between different building blocks.

On the other hand, Neural Architecture Search (NAS) approaches [4], [42], [55] have been showing impressive results on automatically discovering top-performing neural network architectures in large-scale search spaces. Compared to manual designs, NAS methods are data-driven instead of experience-driven, and hence need much less human intervention. As defined in [104], the workflow

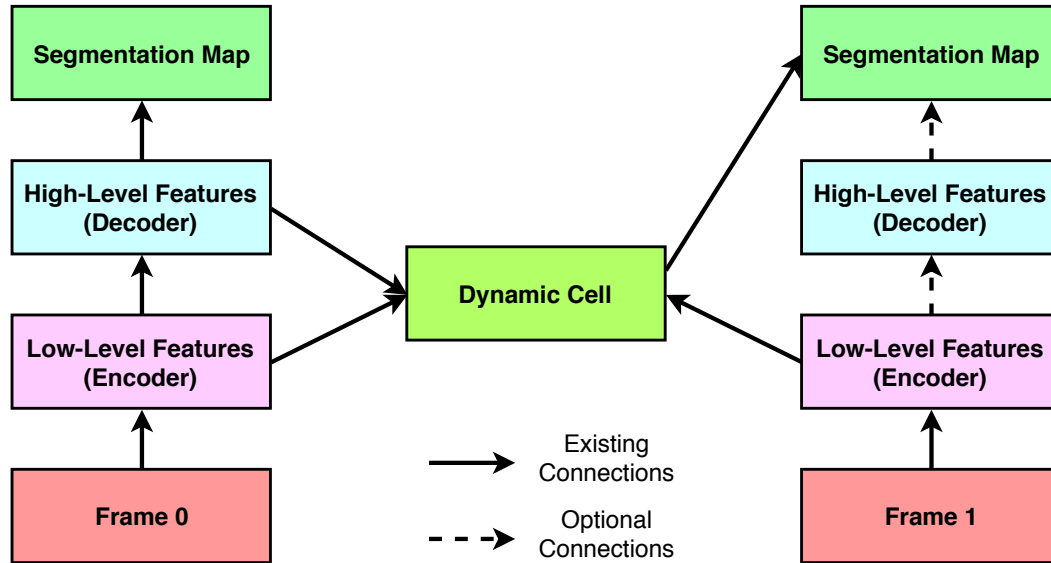


FIGURE 6.1. Semantic video segmentation approaches tend to comprise a dynamic cell that takes as inputs the information from the previous and current frames, and outputs the segmentation mask. For example, the dynamic cell can calculate the optical flow [21], or predict convolutional filters [25]. In this chapter we use NAS to discover novel and high-performing dynamic cells.

of NAS can be divided into the following three processes: 1) sampling architecture from a search space following some search strategies; 2) evaluating the performance of the sampled architecture; and 3) updating the parameters based on the performance.

Starting from that general pattern we instead propose to leverage the neural architecture search (NAS) [51] methodology to find contextual blocks that enhance a per-frame segmentation network with dynamic components. This motivation is justified by recent results achieved using NAS on such tasks as image classification [105], [106], language modelling [107] and static semantic segmentation [52], [53], that oftentimes outperform manually designed networks. We build upon those results and adapt current approaches in a way suitable for handling the dynamic nature of dense per-pixel classification. To the best of our knowledge, we are the first to consider the application of NAS to the task of video semantic segmentation.

Our automated approach comes with certain benefits, concretely:

- i.) it considers a larger span of initial building blocks than any previous work,
- ii.) it empirically evaluates different design structures and finds most promising ones, and
- iii.) it requires only few GPU-days to find a set of high-performing structures.

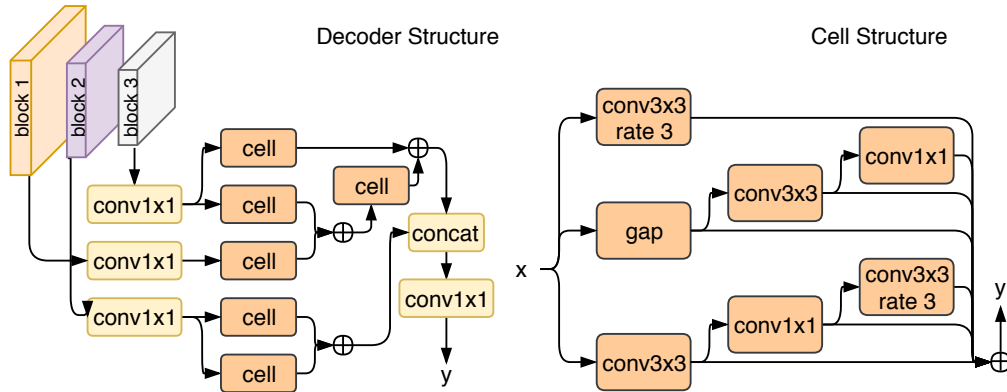


FIGURE 6.2. *arch2* [53]. ‘gap’ stands for global average pooling.

Furthermore, although we do not consider it in this chapter, the proposed methodology can further be extended to take into account different specific objectives (even non-differentiable), such as runtime [108].

6.3 Methodology

As noted in introduction and depicted in Fig. 6.1, we attempt to generalise previous solutions for video semantic segmentation in such a way that NAS methods become readily applicable. To this end, we look for a single cell that connects representations from the previous frame and enhances current predictions without a significant overhead. What follows is the description of the input space (Sect. 6.3.1), the search space (Sect. 6.3.2), and the search approach (Sect. 6.3.3).

6.3.1 Input space

We consider the *arch2* network from the work of Nekrasov *et al.* [53]. It is an encoder-decoder type of the segmentation network with the encoder being a light-weight classifier (MobileNet-v2 [109]), and the decoder being an automatically discovered structure presented in Fig. 6.2. This architecture strikes a fine balance between accuracy and runtime, both being important characteristics for semantic video segmentation. Here it should be noted that the application of our methodology is not directly tied to a concrete architecture and can be easily adapted to work with other networks.

In the proposed setup, the static network is applied end-to-end on the first frame and three outputs are being recorded: an intermediate representation - in this case, the encoder’s output with the resolution of $\frac{1}{32}$ of the input image

(*layer 4*), the decoder’s output before (*dec*) and after the final classifier (*pred*) - both with resolutions of $\frac{1}{8}$ of the input image and with 64 and C numbers of channels, correspondingly, where C is the number of output classes. For the second frame, we record three outputs from the encoder only - two intermediate ones with the resolutions of $\frac{1}{8}$ (*layer 2*) and $\frac{1}{16}$ (*layer 3*), respectively, and the final one with the resolution of $\frac{1}{32}$ (*layer 4*).

We rely on the dynamic cell, the layout of which will be described below, to predict the semantic labelling of the current frame given 5 inputs: *layer 4* and *dec* from the previous frame, and *layers 2–3–4* from the current one. This way, we do not have to execute the decoder part of the static segmentation network on the current frame (thus decreasing latency), at the same time re-using information from the previous frame. The output of the dynamic cell serves as the input *dec* for the next frame.

6.3.2 Search space

We rely on an LSTM-based controller to predict a sequence of operations together with locations where they should be applied in order to form a dynamic cell [53]. Concretely, we first choose two layers out of the provided five (with replacement), two corresponding operations that need to be applied on each of them, and an aggregation operation that combines two inputs into a single output. On the next step, we repeat this process, but now we are sampling two layers out of six possible, with the aggregated result being added into the sampling pool. This process can be repeated multiple times, with the final output being formed by the concatenation of all non-sampled aggregated results.

We rely on a similar set of operations as for static segmentation (Table 6.1), and in order to enable the dynamic cell to apply convolutional filters on irregular grids, we also include deformable 3×3 convolution [66].

ID	Description
0	separable conv 3×3
1	global average pooling followed by upsampling and conv 1×1
2	separable conv 3×3 with dilation rate 3
3	separable conv 5×5 with dilation rate 6
4	skip-connection
5	deformable 3×3 convolution

TABLE 6.1. Description of operations used in the search process.

While Nekrasov *et al.* [53] simply summed up two different inputs at each step, here to compensate for the dynamic nature of our problem we consider a set of aggregation operations given in Table 6.2.

ID	Description
0	summation with per-channel learnable weights per each input
1	channel-wise concatenation of two inputs followed by conv 1×1 to reduce the number of channels to the original size
2	(weight) predictive operation, where the first input becomes a set of spatial convolutional filters (weights) applied on the second one
3	bilinear sampling of the first input, where an affine grid is predicted based on the values of the second input [110]
4	3D-convolution where two inputs are stacked together forming a new dimension with $2 \times 3 \times 3$ convolution applied on top
5	dense attention: i.e. element-wise multiplication between the first input and the sigmoid-activated second one

TABLE 6.2. Description of aggregation operations used in the search process.

Based on the previous works, we conjecture that this set of operations will be sufficient for the task of video segmentation, and we provide experimental results to support this claim. Please see the full code definitions of each operation in the supplementary material.

6.3.3 Finding optimal architectures

We assume that there exists a video dataset that comes with segmentation annotations for at least a subset of consecutive frames. From it, we build pairs (or triplets) of frames such that in each sequence all the frames following the first one are always annotated. As commonly done, we further divide this set into two disjoint parts - meta-train and meta-val. We further assume an existence of the static segmentation network pre-trained on this dataset¹ - in particular, *arch2* from [53]. As mentioned above, we chose this particular architecture due to its compactness and low latency.

The controller samples a structure of the dynamic cell which we train on the meta-train set and evaluate on meta-val. As done in [53], we consider the geometric mean of three metrics as the validation score: mean intersection-over-union (*mIoU*), frequency-weighted IoU (*fwIoU*) and mean-pixel accuracy (*mAcc*). This score is used by the controller to update its weights, and the

¹Please refer to the appendix for the details on pre-training of static segmentation networks.

process is repeated multiple times. After that, one can either sample several cells from the trained controller, or simply choose best found cells that achieved highest results during the search process.

6.4 Experiments

We conduct all our experiments on two popular video segmentation benchmark datasets - CamVid [111] and CityScapes [112].

The first one, *CamVid*, comprises 701 RGB images of resolution 480×360 densely annotated into 11 categories. Following previous work by [113], we use the dataset splits of 367 images for training, 101 - for validation and 233 - for testing. We train generated architectures with batches of examples each comprising 3 consecutive frames.

The *CityScapes* dataset contains 5000 high-resolution 2048×1024 images densely labelled with 19 semantic classes - 2975 for training, 500 for validation and 1525 for testing, respectively. In addition to that, raw unannotated frames extracted from videos are also provided. For each annotated example, we add an image frame that precedes it and train architectures with batches of sequences of length 2, in which the second frame is always annotated.

In each case, we initialise the decoder’s output *dec* on the first frame using the pre-trained static segmentation network, and rely on the dynamic cell at all following frames in the sequence as described in Sect. 6.3.1. To update the dynamic cell weights, we sum up cross-entropy loss terms at each frame after the first one and back-propagate the gradients.

For both, search and training, we exploit a single V100 GPU with 32GB of memory.

6.4.1 Search

For searching we only employ the training splits of each dataset. We further divide each randomly in 2 non-overlapping sets - meta-train (90%) and meta-val (10%). We pre-compute all required outputs from the pre-trained static network and store them in memory. The static network is kept unchanged during the whole search process. Each generated architecture is trained on the meta-train split and evaluated on meta-val. We keep track of average performance and apply early stopping halfway through the training if the generated architecture is un-promising as done in [53].

Our controller is a two-layer LSTM with 100 hidden units randomly initialised from uniform distribution [53]. The controller is trained with PPO [114] with the learning rate of $1e-4$. To reduce the size of generated cells, we set the number of emitted layers (each layer is a string of five tokens as described in Sect. 6.3.2) to 5 on CamVid and to 4 on CityScapes.

For *CamVid*, we train predicted cells on mini-batches of 48 sequences for 20 epochs with the learning rate of $8e-3$ and the Adam learning rule [115]. Each image–segmentation mask pair in the sequence is cropped to 600 with the shorter side being mean-padded to 400. No transformations are applied to the validation sequences.

For *CityScapes*, we train for 10 epochs with 48 sequences each cropped to 512×512 with the longer side being resized to 1024.

Results

We visualise the progress of each metric together with the reward signal on each dataset in Fig. 6.3. Although the rewards are not directly comparable between the datasets, the growth dynamics on both datasets signal that the controller is able to discover better architectures throughout the search process across all the metrics.

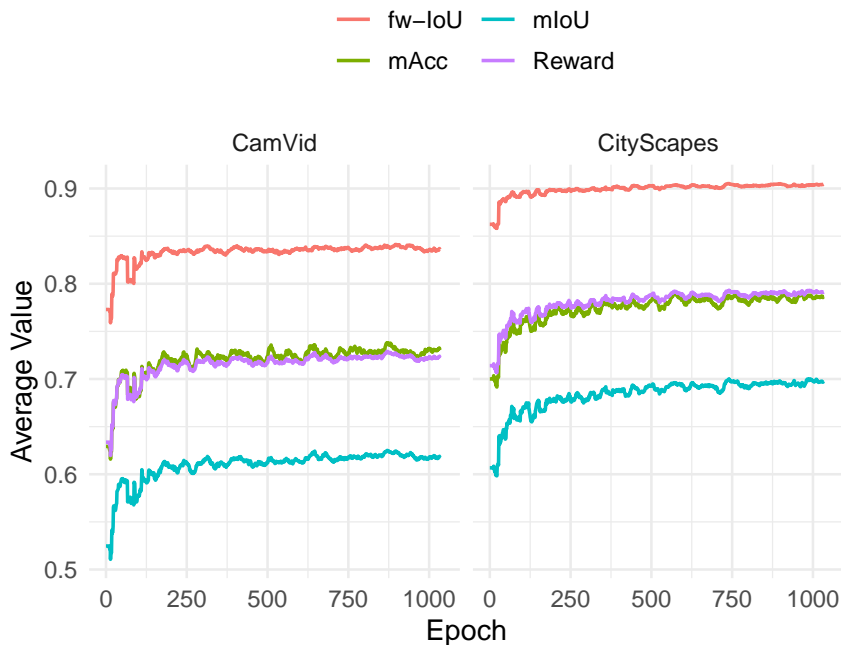


FIGURE 6.3. Average search metrics on CamVid and CityScapes datasets.

We further look at the distributions of sampled operations, aggregation operations and input layers plotted on Fig. 6.4. On both datasets, global average pooling and separable 5×5 convolution with dilation rate 6 are sampled less frequently than other operations, potentially indicating that these layers could be omitted from the search process. On average, the controller trained on CityScapes prefers sampling deformable convolution (Fig. 6.4a), while the CamVid one - separable 3×3 convolution (Fig. 6.4d).

In terms of aggregation operations, the dynamics between two controllers vary significantly: the CamVid-based controller tend to rely on dense attention, while omitting the predictive operation (Fig. 6.4e). In contrast, the CityScapes controller is more likely to apply bilinear sampling on an affine grid, and to ignore predictive operation together with dense attention (Fig. 6.4b).

When sampling the input layers, the controllers behave similarly: in particular, both tend to skip *layer* 4 from the previous and current frames. The CityScapes controller extensively uses information from the previous *dec* layer (Fig. 6.4c), while the CamVid one - from *layer* 2 of the current frame (Fig. 6.4f). This may well imply that on CityScapes the final predictions on the current frame change only slightly with respect to the previous frame.

Importantly, these observations indicate that two controllers trained on two different datasets exhibit various patterns, potentially capturing dataset-specific attributes in order to discover better performing architectures.

6.4.2 End-to-end Training

We further select top-2 performing dynamic cells on each dataset to train end-to-end on full training sets for longer.

In particular, for *CamVid*, we pre-train the dynamic cell only with Adam and the learning rate of $8e-3$ for 10 epochs with the batch size of 16 sequences. Then we decrease the cell’s learning rate in half, and fine-tune the whole architecture (i.e., with the per-frame segmentation network) end-to-end for 100 epochs - the static network weights are updated using SGD with momentum of 0.9 and the learning rate of $5e-4$. Each sample in the batch is cropped to 600×600 with the shorter side being padded to 400.

On *CityScapes* we pre-train for 200 epochs with the batch size of 16 sequences and fine-tune end-to-end for 200 epochs. Each example in the batch is cropped to 769×769 .

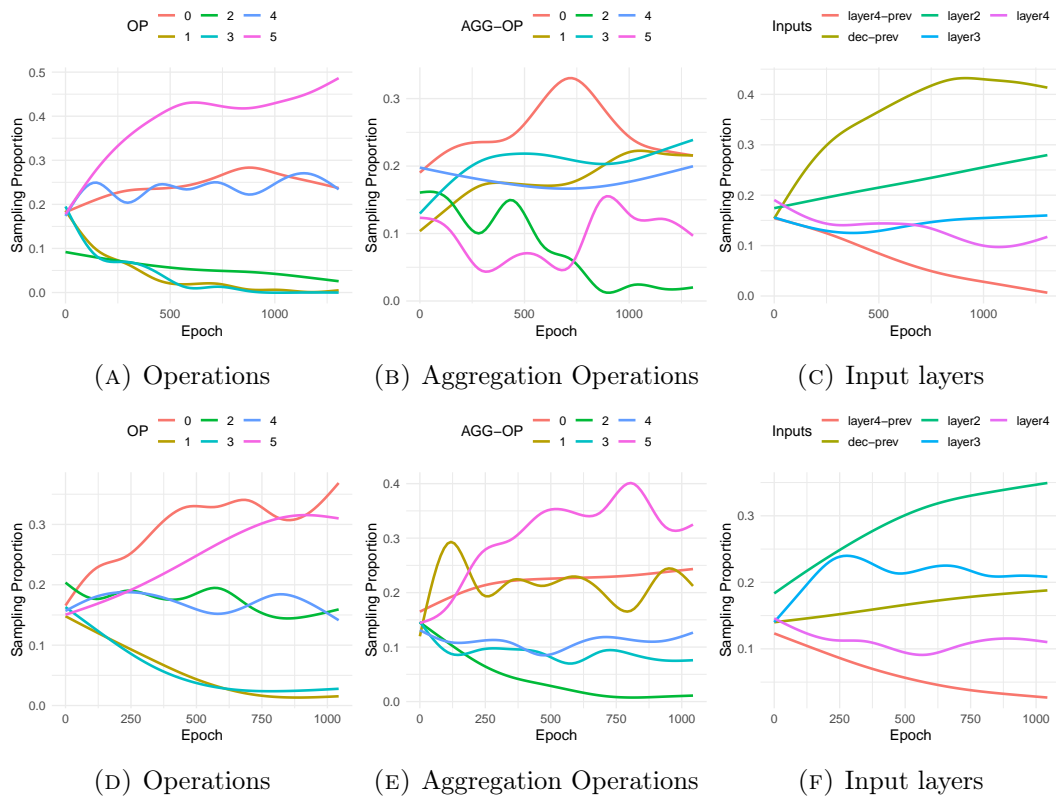


FIGURE 6.4. Average sampling proportion of operations, aggregation operations and input layers on CityScapes (a-c) and CamVid (d-f). Please refer to Tables 6.1 and 6.2 for the description of operations.

CamVid Results

We provide quantitative results on CamVid in Table 6.3. The inclusion of dynamic cells in both cases leads to an improvement over baseline by more than 1%. Importantly, with the exclusion of first frame in the sequence, we do not rely on expensive computations involving the static decoder.

Both our models perform comparably to other state-of-the-art video segmentation networks even though the backbone that we rely on - MobileNet-v2 [109] - is much smaller in comparison to ResNet-101 [116] exploited by Chandra *et al.* [26], or DilatedNet [117] - by Gadde *et al.* [21] and GRFP [24]. Furthermore, we did not make any use of higher-resolution images of 960×720 to further improve our scores.

We further visualise a few qualitative examples in Fig. 6.5. The dynamic cell enables the network to effectively propagate information about thin structures, such as poles, which makes the resultant segmentation masks consistent in contrast to the per-frame baseline (rows 1–5). Furthermore, the multi-frame segmentation network is able to track objects across neighbouring frames (rows

Method	mIoU,%	mAcc,%	gAcc,%	tIoU,%
per-frame baseline	65.3	76.1	90.8	41.4
w/ cell0	66.6	77.6	91.1	42.6
w/ cell1	66.9	78.5	90.1	42.4
GRFP [24]	66.1	-	-	-
Chandra <i>et al.</i> [26]	67.0	-	-	-
Gadde <i>et al.</i> [21]	67.1	-	-	36.6

TABLE 6.3. Quantitative results on the test set of CamVid. We report mean IoU (mIoU). Note that our method uses MobileNet-v2 as the encoder network.

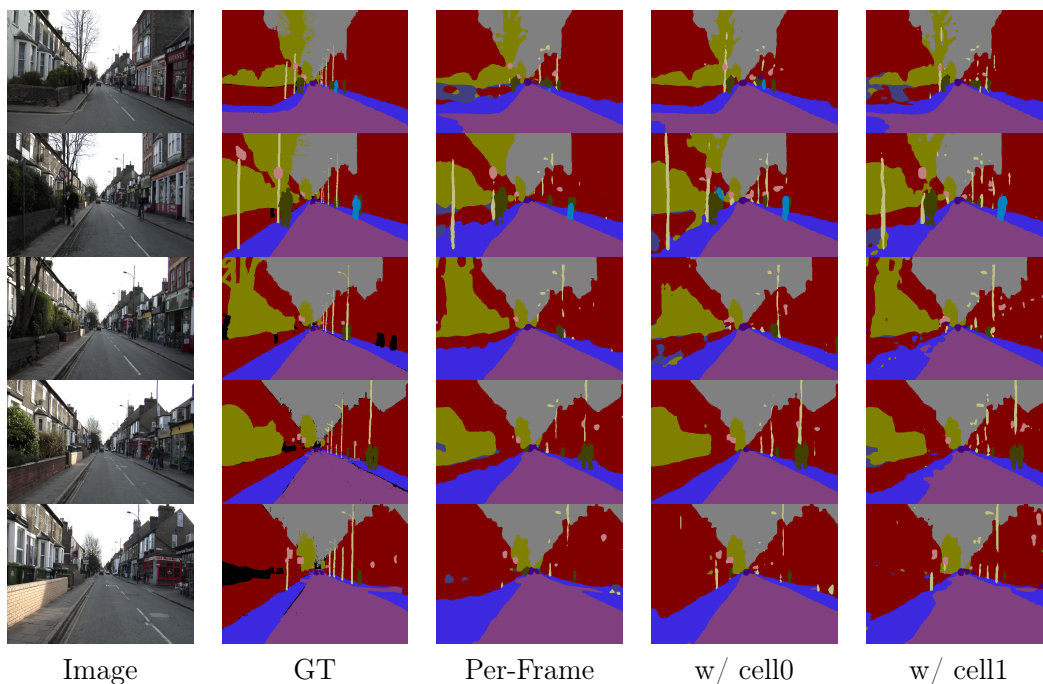


FIGURE 6.5. Inference results on the test set of CamVid.

1–2).

CityScapes Results

We include the validation results of two discovered cells on CityScapes in Table 6.4. Once again, both dynamic cells are able to outperform the per-frame baseline by 1.2%. Furthermore, our models achieve favourable results in comparison to other video segmentation methods, all of which employ significantly larger backbones and, with the exclusion of Li *et al.* [25], all rely on the optical flow computation. Note also that Gadde *et al.* [21] improved over their respective static baseline by 1.2%, too, while introducing a non-negligible overhead of 40ms; and Li *et al.* [25] compromised more than 3% of the baseline score in

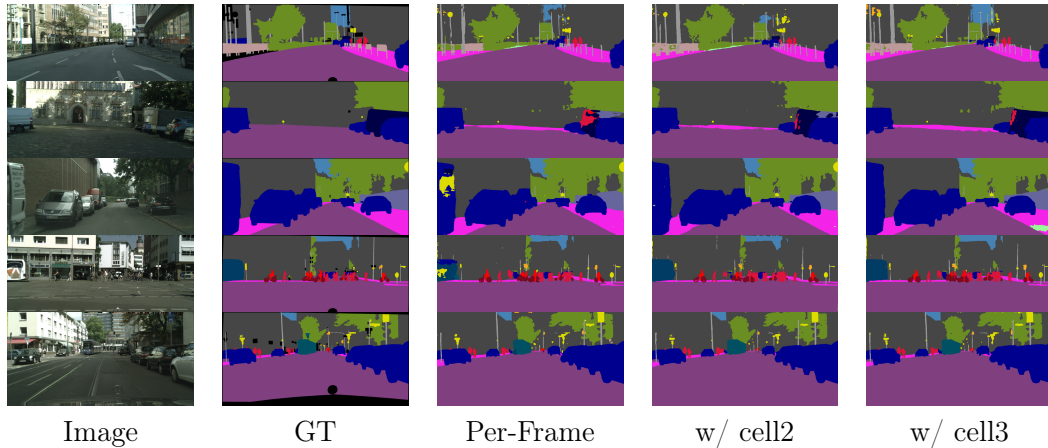


FIGURE 6.6. Inference results on the validation set of CityScapes.

order to reduce the latency. In contrast, we overcame our static baseline and decreased the runtime (Table 6.5).

Method	mIoU, %	Acc, %	tIoU, %
per-frame baseline	74.4	82.6	40.1
w/ cell2 ²	75.6	84.4	41.5
w/ cell3 ³	75.6	83.7	41.5
GRFP(5) [24]	69.5	-	-
Xu <i>et al.</i> [23]	70.4	-	-
Li <i>et al.</i> [25]	76.8	-	-
Gadde <i>et al.</i> [21]	80.6	-	42.1

TABLE 6.4. Comparison with other video segmentation approaches on the val set of CityScapes. Note that our method uses MobileNet-v2 as the encoder network. For tIoU, the trimap width is 3.

A few inference examples are visualised in Fig. 6.6. As can be seen, the dynamic cells enhance the per-frame baseline results and identify partially occluded vehicles more accurately (rows 1–2, 5), while also avoiding misclassification of traffic signs at pixels with similar texture patterns (rows 2–4).

6.4.3 Details of Discovered Architectures

We include characteristics of our networks together with numbers reported by others in Table 6.5. As evident, our dynamic segmentation approach is superior to others in terms of its latency and compactness. Concretely, all our architectures contain at most 3.4M parameters while having an average per-frame

²Anonymous test results: <https://bit.ly/2FrZ8jM>

³Anonymous test results: <https://bit.ly/2HyoVcb>

6.5 Discussion & Conclusions

It is still an open question of what is the optimal way of propagating and extracting information across video frames. While a straightforward solution involving the optical flow allows to achieve solid results, it possesses several disadvantages that stem from the limitations of the optical flow itself and ultimately limit the ability of the network to adapt to novel frames. Furthermore, computations involving the optical flow cause a significant overhead, prohibiting the final system from being deployed in real-time.

In this chapter, instead of manually enhancing static segmentation networks with dynamic components, we proposed an automatic approach based on neural architecture search methods. Such automation have multiple benefits as it explores a large pool of networks and finds best-performing ones on the given dataset. In a broader sense, starting from a static per-frame segmentation network, we showcased a way of generalising existing solutions without any reliance on the optical flow. In particular, we extended the static baseline with a dynamic cell, the design of which is automatically discovered with the help of reinforcement learning. The best discovered cells improve the baseline by more than 1% at the same time leading to significant memory and latency savings. Concretely, two discovered cells on CityScapes reach 75.6% mean IoU and require only 50ms on average to process a 2048×1024 frame. While the proposed methodology relies on the static baseline, we expect that omitting that requirement and searching for a video segmentation network end-to-end would be an interesting problem to consider in the future work.

Statement of Authorship

Title of Paper	NAS-FCOS: Fast Neural Architecture Search for Object Detection
Publication Status	<input checked="" type="checkbox"/> Published <input type="checkbox"/> Accepted for Publication <input type="checkbox"/> Submitted for Publication <input type="checkbox"/> Unpublished and Unsubmitted work written in manuscript style
Publication Details	CVPR 2020 11940-11948

Principal Author

Name of Principal Author (Candidate)	Hao Chen		
Contribution to the Paper	Experiment framework design and implementation; Writing method part of the paper.		
Overall percentage (%)	50%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	02/10/2020

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

- the candidate's stated contribution to the publication is accurate (as detailed above);
- permission is granted for the candidate to include the publication in the thesis; and
- the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Ning Wang		
Contribution to the Paper	Conducting the experiments, tuning design details, writing draft of the paper.		
Signature		Date	2/10/2020

Name of Co-Author	Yang Gao		
Contribution to the Paper	Conducting the experiments, tuning design details, writing draft of the paper.		
Signature		Date	2/10/2020

Name of Co-Author	Peng Wang		
Contribution to the Paper	Discussion and writing revision.		
Signature		Date	2/10/2020

Name of Co-Author	Zhi Tian		
Contribution to the Paper	Discussion and writing revision.		
Signature		Date	2/10/2020

Name of Co-Author	Chunhua Shen		
Contribution to the Paper	Discussion and writing revision.		
Signature		Date	2/10/2020

Name of Co-Author	Yanning Zhang		
Contribution to the Paper	Discussion and writing revision.		
Signature		Date	2/10/2020

Chapter 7

NAS-FCOS: Searching Decoder for Object Detection

7.1 Introduction

In this chapter, we further extend neural architecture search To date, on challenging vision tasks such as object detection, NAS, especially fast versions of NAS, is less studied. Here we propose to search for the decoder structure of object detectors with search efficiency being taken into consideration. To be more specific, we aim to efficiently search for the feature pyramid network (FPN) as well as the prediction head of a simple anchor-free object detector, namely FCOS [9], using a tailored reinforcement learning paradigm. With carefully designed search space, search algorithms and strategies for evaluating network quality, we are able to efficiently search a top-performing detection architecture within 4 days using 8 V100 GPUs. The discovered architecture surpasses state-of-the-art object detection models (such as Faster R-CNN, RetinaNet and FCOS) by 1.5 to 3.5 points in AP on the COCO dataset, with comparable computation complexity and memory footprint, demonstrating the efficacy of the proposed NAS for object detection.

7.2 Background

Object detection is one of the fundamental tasks in computer vision, and has been researched extensively. In the past few years, state-of-the-art methods for this task are based on deep convolutional neural networks (such as Faster R-CNN [8], RetinaNet [54]), due to their impressive performance. Typically, the designs of object detection networks are much more complex than those for image classification, because the former need to localize and classify multiple objects in an image simultaneously while the latter only need to output image-level labels. Due to its complex structure and numerous hyper-parameters,

designing effective object detection networks is more challenging and usually needs much manual effort.

To our knowledge, studies on efficient and accurate NAS approaches to object detection networks are rarely touched, despite its significant importance. One of the main problems prohibiting NAS from being used in more realistic applications is its search efficiency. The evaluation process is the most time consuming part because it involves a full training procedure of a neural network. To reduce the evaluation time, in practice a proxy task is often used as a lower cost substitution. In the proxy task, the input, network parameters and training iterations are often scaled down to speedup the evaluation. However, there is often a performance gap for samples between the proxy tasks and target tasks, which makes the evaluation process biased. How to design proxy tasks that are both accurate and efficient for specific problems is a challenging problem. Another solution to improve search efficiency is constructing a super-net that covers the complete search space and training candidate architectures with shared parameters [45], [118]. However, this solution leads to significantly increased memory consumption and restricts itself to small-to-moderate sized search spaces.

To this end, we present a fast and memory saving NAS method for object detection networks, which is capable of discovering top-performing architectures within significantly reduced search time. Our overall detection architecture is based on FCOS [9], a simple anchor-free one-stage object detection framework, in which the feature pyramid network and prediction head are searched using our proposed NAS method.

Our main contributions are summarized as follows.

- In this work, we propose a fast and memory-efficient NAS method for searching both FPN and head architectures, with carefully designed proxy tasks, search space and evaluation strategies, which is able to find top-performing architectures over 3,000 architectures using 28 GPU-days only. Specifically, this high efficiency is enabled with the following designs.
 - Developing a fast proxy task training scheme by skipping the backbone finetuning stage;
 - Adapting progressive search strategy to reduce time cost taken by the extended search space;
 - Using a more discriminative criterion for evaluation of searched architectures.

- Employing an efficient anchor-free one-stage detection framework with simple post processing;
- Using NAS, we explore the workload relationship between FPN and head, proving the importance of weight sharing in head.
- We show that the overall structure of NAS-FCOS is general and flexible in that it can be equipped with various backbones including MobileNetV2, ResNet-50, ResNet-101 and ResNeXt-101, and surpasses state-of-the-art object detection algorithms using comparable computation complexity and memory footprint. More specifically, our model can improve the AP by 1.5 ~ 3.5 points on all above models comparing to their FCOS counterparts.

To reduce the post processing overhead, we resort to a recently introduced anchor-free one-stage framework, namely, FCOS [9], which significantly improve the search efficiency by cancelling the processing time of anchor-box matching in RetinaNet.

Compared to its anchor-based counterpart, FCOS significantly reduces the training memory footprint while being able to improve the performance.

7.3 Methodology

In our work, we search for anchor-free fully convolutional detection models with fast decoder adaptation. Thus, NAS methods can be easily applied.

7.3.1 Problem Formulation

We base our search algorithm upon a one-stage framework FCOS due to its simplicity. Our training tuples $\{(\mathbf{x}, Y)\}$ consist of input image tensors \mathbf{x} of size $(3 \times H \times W)$ and FCOS output targets Y in a pyramid representation, which is a list of tensors \mathbf{y}_l each of size $((K + 4 + 1) \times H_l \times W_l)$ where $H_l \times W_l$ is feature map size on level p of the pyramid. $(K + 4 + 1)$ is the output channels of FCOS, the three terms are length- K one-hot classification labels, 4 bounding box regression targets and 1 centerness factor respectively.

The network $g : \mathbf{x} \rightarrow \hat{Y}$ in original FCOS consists of three parts, a backbone b , FPN f and multi-level subnets we call prediction heads h in this chapter. First backbone $b : \mathbf{x} \rightarrow C$ maps the input tensor to a set of intermediate-leveled features $C = \{\mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5\}$, with resolution $(H_i \times W_i) = (H/2^i \times W/2^i)$. Then FPN $f : C \rightarrow P$ maps the features to a feature pyramid $P = \{\mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7\}$.

Then the prediction head $h : \mathbf{p} \rightarrow \mathbf{y}$ is applied to each level of P and the result is collected to create the final prediction. To avoid overfitting, same h is often applied to all instances in P .

Since objects of different scales require different effective receptive fields, the mechanism to select and merge intermediate-level features C is particularly important in object detection network design. Thus, most researches [8], [29] are carried out on designing f and h while using widely-adopted backbone structures such as ResNet [77]. Following this principle, our search goal is to decide when to choose which features from C and how to merge them.

To improve the efficiency, we reuse the parameters in b pretrained on target dataset and search for the optimal structures after that. For the convenience of the following statement, we call the network components to search for, namely f and h , together the decoder structure for the objection detection network.

f and h take care of different parts of the detection job. f extracts features targeting different object scales in the pyramid representations P , while h is a unified mapping applied to each feature in P to avoid overfitting. In practice, people seldom discuss the possibility of using a more diversified f to extract features at different levels or how many layers in h need to be shared across the levels. In this work, we use NAS as an automatic method to test these possibilities.

7.3.2 Search Space

Considering the different functions of f and h , we apply two search space respectively. Given the particularity of FPN structure, a basic block with new overall connection and f 's output design is built for it. For simplicity, sequential space is applied for h part.

We replace the cell structure with atomic operations to provide even more flexibility. To construct one basic block, we first choose two layers $\mathbf{x}_1, \mathbf{x}_2$ from the sampling pool X at $\text{id1}, \text{id2}$, then two operations $\text{op1}, \text{op2}$ are applied to each of them and an aggregation operation agg merges the two output into one feature. To build a deep decoder structure, we apply multiple basic blocks with their outputs added to the sampling pool. Our basic block $bb_t : X_{t-1} \rightarrow X_t$ at time step t transforms the sampling pool X_{t-1} to $X_t = X_{t-1} \cup \{\mathbf{x}_t\}$, where \mathbf{x}_t is the output of bb_t .

The candidate operations are listed in Table 7.1. We include only separable/depth-wise convolutions so that the decoder can be efficient. In order to enable the

decoder to apply convolutional filters on irregular grids, here we have also included deformable 3×3 convolutions [66]. For the aggregation operations, we include element-wise sum and concatenation followed by a 1×1 convolution.

The decoder configuration can be represented by a sequence with three components, FPN configuration, head configuration and weight sharing stages. We provide detailed descriptions to each of them in the following sections. The complete diagram of our decoder structure is shown in Fig. 7.1.

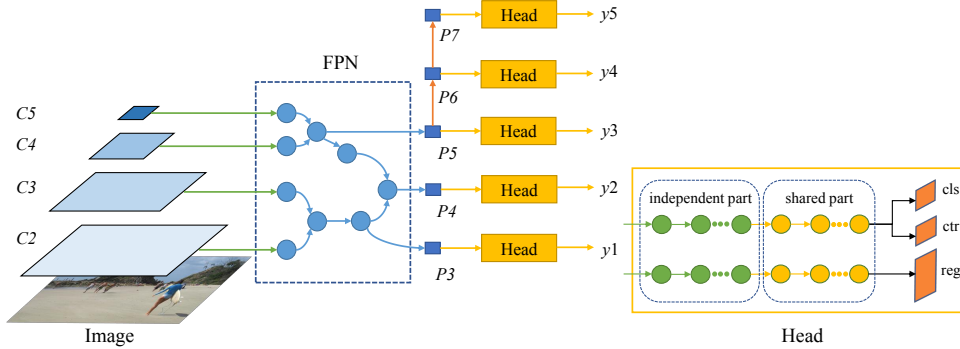


FIGURE 7.1. A conceptual example of our NAS-FCOS decoder. It consists of two sub networks, an FPN f and a set of prediction heads h which have shared structures. One notable difference with other FPN-based one-stage detectors is that our heads have partially shared weights. Only the last several layers of the predictions heads (marked as yellow) are tied by their weights. The number of layers to share is decided automatically by the search algorithm. Note that both FPN and head are in our actual search space; and have more layers than shown in this figure.

Here the figure is for illustration only.

FPN Search Space

As mentioned above, the FPN f maps the convolutional features C to P . First, we initialize the sampling pool as $X_0 = C$. Our FPN is defined by applying the basic block 7 times to the sampling pool, $f := bb_1^f \circ bb_2^f \circ \dots \circ bb_7^f$. To yield

ID	Description
0	separable conv 3×3
1	separable conv 3×3 with dilation rate 3
2	separable conv 5×5 with dilation rate 6
3	skip-connection
4	deformable 3×3 convolution

TABLE 7.1. Unary operations used in the search process.

pyramid features P , we collect the last three basic block outputs $\{\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7\}$ as $\{\mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5\}$.

To allow shared information across all layers, we use a simple rule to create global features. If there is some dangling layer \mathbf{x}_t which is not sampled by later blocks $\{bb_i^f | i > t\}$ nor belongs to the last three layers $t < 5$, we use element-wise add to merge it to all output features

$$\mathbf{p}_i^* = \mathbf{p}_i + \mathbf{x}_t, \quad i \in \{3, 4, 5\}. \quad (7.1)$$

Same as the aggregation operations, if the features have different resolution, the smaller one is upsampled with bilinear interpolation.

To be consistent with FCOS, \mathbf{p}_6 and \mathbf{p}_7 are obtained via a 3×3 stride-2 convolution on \mathbf{p}_5 and \mathbf{p}_6 respectively.

Prediction Head Search Space

Prediction head h maps each feature in the pyramid P to the output of corresponding \mathbf{y} , which in FCOS and RetinaNet, consists of four 3×3 convolutions. To explore the potential of the head, we therefore extend a sequential search space for its generation. Specifically, our head is defined as a sequence of six basic operations. Compared with candidate operations in the FPN structures, the head search space has two slight differences. First, we add standard convolution modules (including conv1x1 and conv3x3) to the head sampling pool for better comparison. Second, we follow the design of FCOS by replacing all the Batch Normalization (BN) layers to Group Normalization (GN) [119] in the operations sampling pool of head, considering that head needs to share weights between different levels, which causes BN invalid. The final output of head is the output of the last (sixth) layer.

Searching for Head Weight Sharing

To add even more flexibility and understand the effect of weight sharing in prediction heads, we further add an index i as the location where the prediction head starts to share weights. For every layer before stage i , the head h will create independent set of weights for each FPN output level, otherwise, it will use the global weights for sharing purpose.

Considering the independent part of the heads being extended FPN branch and the shared part as head with adaptive-length, we can further balance the workload for each individual FPN branch to extract level-specific features and the prediction head shared across all levels.

7.3.3 Search Strategy

RL based strategy is applied to the search process. We rely on an LSTM-based controller to predict the full configuration. We consider using a progressive search strategy rather than the joint search for both FPN structure and prediction head part, since the former requires less computing resources and time cost than the latter. The training dataset is randomly split into a meta-train D_t and meta-val D_v subset. To speed up the training, we fix the backbone network and cache the pre-computed backbone output C . This makes our single architecture training cost independent from the depth of backbone network. Taking this advantage, we can apply much more complex backbone structures and utilize high quality multilevel features as our decoder’s input. We find that the process of backbone finetuning can be skipped if the cached features are powerful enough. Speedup techniques such as Polyak weight averaging are also applied during the training.

The most widely used detection metric is average precision (AP). However, due to the difficulty of object detection task, at the early stages, AP is too low to tell the good architectures from the bad ones, which makes the controller take much more time to converge. To make the architecture evaluation process easier even at the early stages of the training, we therefore use negative loss sum as the reward instead of average precision:

$$R(a) = - \sum_{(x,Y) \in D_v} (L_{cls}(x, Y|a) + L_{reg}(x, Y|a) + L_{ctr}(x, Y|a)) \quad (7.2)$$

where L_{cls} , L_{reg} , L_{ctr} are the three loss terms in FCOS. Gradient of the controller is estimated via proximal policy optimization (PPO) [120].

7.4 Experiments

7.4.1 Implementation Details

Searching Phase

We design a fast proxy task for evaluating the decoder architectures sampled in the searching phase. PASCAL VOC is selected as the proxy dataset, which contains 5715 training images with object bounding box annotations of 20 classes. Transfer capacity of the structures can be illustrated since the search and full training phase use different datasets. The VOC training set is randomly split

into a meta-train set with 4,000 images and a meta-val set with 1715 images. For each sampled architecture, we train it on meta-train and compute the reward (7.2) on meta-val. Input images are resized to short size 384 and then randomly cropped to 384×384 . Target object sizes of interest are scaled correspondingly. We use Adam optimizer with learning rate $8e-4$ and batch size 200. Polyak averaging is applied with the decay rates of 0.9. The decoder is evaluated after 300 iterations. As we use fast decoder adaptation, the backbone features are fixed and cached during the search phase. To enhance the cached backbone features, we first initialize them with pre-trained weights provided by open-source implementation of FCOS and then finetune on VOC using the training strategies of FCOS. Note that the above finetuning process is only performed once at the beginning of the search phase.

A progressive strategy is used for the search of f and h . We first search for the FPN part and retain the original head. All operations in the FPN structure have 64 output channels. The decoder inputs C are resized to fit output channel width of FPN via 1×1 convolutions. After this step, a searched FPN structure is fixed and the second stage searching for the head will be started based on it. Most parameters for searching head are identical to those for searching FPN structure, with the exception that the output channel width is adjusted from 64 to 128 to deliver more information.

For the FPN search part, the controller model nearly converged after searching over 2.8K architectures on the proxy task as shown in Fig. 7.2. Then, the top-20 best performing architectures on the proxy task are selected for the next full training phase. For the head search part, we choose the best searched FPN among the top-20 architectures and pre-fetch its features. It takes about 600 rounds for the controller to nearly converge, which is much faster than that for searching FPN architectures. After that, we select for full training the top-10 heads that achieve best performance on the proxy task. In total, the whole search phase can be finished within 4 days using 8 V100 GPUs.

Full Training Phase

In this phase, we fully train the searched models on the MS COCO training dataset, and select the best one by evaluating them on MS COCO validation images. Note that our training configurations are exactly the same as those in FCOS for fair comparison. Input images are resized to short size 800 and the maximum long side is set to be 1333. The models are trained using 4 V100 GPUs with batch size 16 for 90K iterations. The initial learning rate is 0.01

Decoder	Backbone	FLOPs (G)	Params (M)	AP
FPN-RetinaNet @256	MobileNetV2	133.4	11.3	30.8
FPN-FCOS @256	MobileNetV2	105.4	9.8	31.2
NAS-FCOS (ours) @128	MobileNetV2	39.3	5.9	32.0
NAS-FCOS (ours) @128-256	MobileNetV2	95.6	9.9	33.8
NAS-FCOS (ours) @256	MobileNetV2	121.8	16.1	34.7
FPN-RetinaNet @256	R-50	198.0	33.6	36.1
FPN-FCOS @256	R-50	169.9	32.0	37.4
NAS-FCOS (ours) @128	R-50	104.0	27.8	37.9
NAS-FCOS (ours) @128-256	R-50	160.4	31.8	39.1
NAS-FCOS (ours) @256	R-50	189.6	38.4	39.8
FPN-RetinaNet @256	R-101	262.4	52.5	37.8
FPN-FCOS @256	R-101	234.3	50.9	41.5
NAS-FCOS (ours) @256	R-101	254.0	57.3	43.0
FPN-FCOS @256	X-64x4d-101	371.2	89.6	43.2
NAS-FCOS (ours) @128-256	X-64x4d-101	361.6	89.4	44.5
FPN-FCOS @256 w/improvements	X-64x4d-101	371.2	89.6	44.7
NAS-FCOS (ours) @128-256 w/improvements	X-64x4d-101	361.6	89.4	46.1

TABLE 7.2. Results on test-dev set of MS COCO after full training. R-50 and R-101 represents ResNet backbones and X-64x4d-101 represents ResNeXt-101 ($64 \times 4d$). All networks share the same input image resolution. FLOPs and parameters are being measured on 1088×800 , which is the median of the input size on COCO. For RetinaNet and FCOS, we use official models provided by the authors. For our NAS-FCOS, @128 and @256 means that the decoder channel width is 128 and 256 respectively. @128-256 is the decoder with 128 FPN width and 256 head width. The same improving tricks used on the newest FCOS version are used in our model for fair comparison.

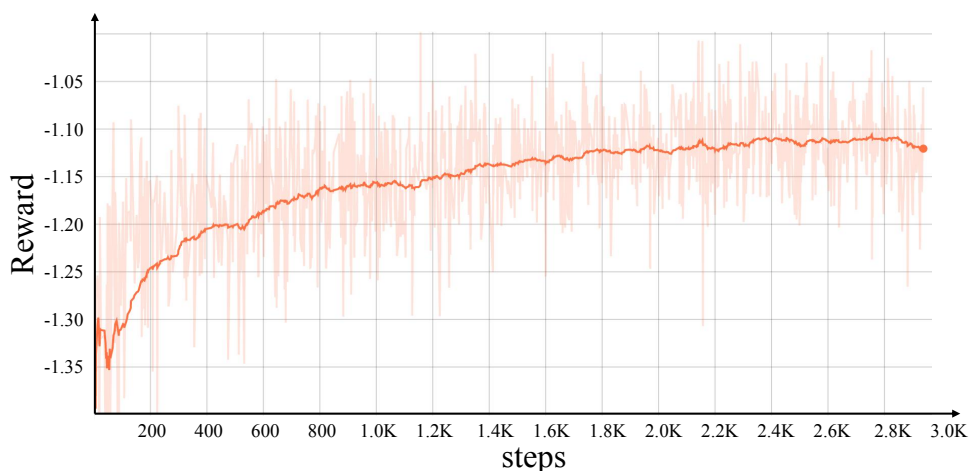


FIGURE 7.2. Performance of reward during the proxy task, which has been growing throughout the process, indicating that the model of reinforcement learning works.

and reduces to one tenth at the 60K-th and 80K-th iterations. The improving tricks are applied only on the final model (w/improv).

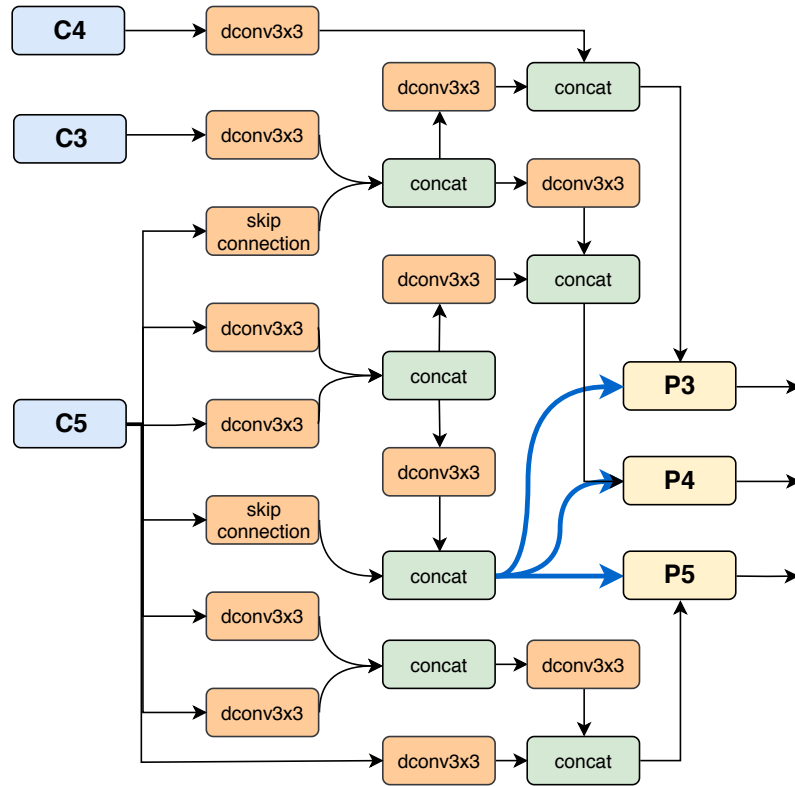


FIGURE 7.3. Our discovered FPN structure. C_2 is omitted from this figure since it is not chosen by this particular structure during the search process.

7.4.2 Search Results

The best FPN structure is illustrated in Fig. 7.3. The controller identifies that deformable convolution and concatenation are the best performing operations for unary and aggregation respectively. From Fig. 7.4, we can see that the controller chooses to use 4 operations (with two skip connections), rather than the maximum allowed 6 operations. Note that the discovered “dconv + 1x1 conv” structure achieves a good trade-off between accuracy and FLOPs. Compared with the original head, our searched head has fewer FLOPs/Params (FLOPs 79.24G vs. 89.16G, Params 3.41M vs. 4.92M) and significantly better performance (AP 38.7 vs. 37.4).

We use the searched decoder together with either light-weight backbones such as MobileNet-V2 [121] or more powerful backbones such as ResNet-101 [77]

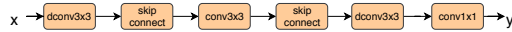


FIGURE 7.4. Our discovered Head structure.

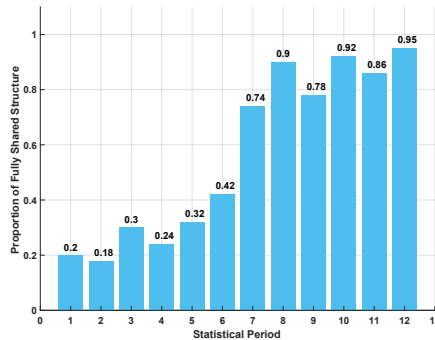


FIGURE 7.5. Trend of head weight sharing during search. The coordinates in the horizontal axis represent the number of the statistical period. A period consists of 50 head structures. The vertical axis represents the proportion of heads that fully share weights in 50 structures.

and ResNeXt-101 [122]. To balance the performance and efficiency, we implement three decoders with different computation budgets: one with feature dimension of 128 (@128), one with 256 (@256) and another with FPN channel width 128 and prediction head 256 (@128-256). The results on the COCO test-dev with short side being 800 is shown in Table 7.2. The searched decoder with feature dimension of 256 (@256) surpasses its FCOS counterpart by 1.5 to 3.5 points in AP under different backbones. The one with 128 channels (@128) has significantly reduced parameters and calculation, making it more suitable for resource-constrained environments. In particular, our searched model with 128 channels and MobileNetV2 backbone surpasses the original FCOS with the same backbone by 0.8 AP points with only 1/3 FLOPS. The third type of decoder (@128-256) achieves a good balance between accuracy and parameters. Note that our searched model outperforms the strongest FCOS variant by 1.4 AP points (46.1 vs. 44.7) with slightly smaller FLOPs and Params. The comparison of FLOPs and number of parameters with other models are illustrated in Fig. 7.7 and Fig. 7.8 respectively.

In order to understand the importance of weight sharing in head, we add the number of layers shared by weights as an object of the search. Fig. 7.5 shows a trend graph of head weight sharing during search. We set 50 structures as a statistical cycle. As the search deepens, the proportion of fully shared structures increases, indicating that on the multi-scale detection model, head weight sharing is a necessity.

We also demonstrate the comparison with other NAS methods for object

Arch	FLOPs (G)	Search Cost (GPU-day)	Searched Archs	AP
NAS-FPN @256 R-50	>325.0	333×#TPUs	17000	<38.0
NAS-FPN 7@256 R-50	1125.5	333×#TPUs	17000	44.8
DetNAS-FPN-Faster	-	44	2200	40.2
DetNAS-RetinaNet	-	44	2200	33.3
NAS-FCOS (ours) @256 R-50	189.6	28	3000	39.8
NAS-FCOS (ours) @128-256 X-64x4d-101	361.6	28	3000	46.1

TABLE 7.3. Comparison with other NAS methods. For NAS-FPN, the input size is 1280×1280 and the search cost should be timed by their number of TPUs used to train each architecture. Note that the FLOPs and AP of NAS-FPN @256 here are from Figure 11 in NAS-FPN [55], and NAS-FPN 7@256 stacks the searched FPN structure 7 times. The input images are resized such that their shorter size is 800 pixels in DetNASNet [49] and our models.

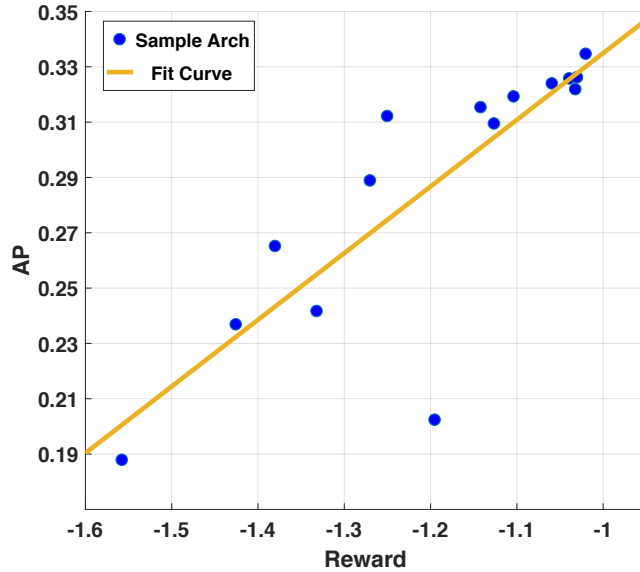


FIGURE 7.6. Correlation between the search reward obtained on the VOC meta-val dataset and the AP evaluated on COCO-val.

detection in Table 7.3. Our method is able to search for twice more architectures than DetNAS [49] per GPU-day. Note that the AP of NAS-FPN [55] is achieved by stacking the searched FPN 7 times, while we do not stack our searched FPN. Our model with ResNeXt-101 (64x4d) as backbone outperforms NAS-FPN by 1.3 AP points while using only 1/3 FLOPs and less calculation cost.

We further measure the correlation between rewards obtained during the search process with the proxy dataset and APs attained by same architectures trained on COCO. Specifically, we randomly sample 15 architectures from all the searched structures trained on COCO with batch size 16. Since full training on COCO is time-consuming, we reduce the iterations to 60K. The model is then evaluated on the COCO 2017 validation set. As visible in Fig. 7.6, there is a strong correlation between search rewards and APs obtained from COCO. Poor- and well-performing architectures can be distinguished by the rewards on the proxy task very well.

7.4.3 Ablation Study

Design of Reinforcement Learning Reward

As we discussed above, it is common to use widely accepted indicators as rewards for specific tasks in the search, such as mIOU for segmentation and AP for object detection. However, we found that using AP as reward did not show a clear upward trend in short-term search rounds (blue curve in Fig. 7.9). We further analyze the possible reason to be that the controller tries to learn a

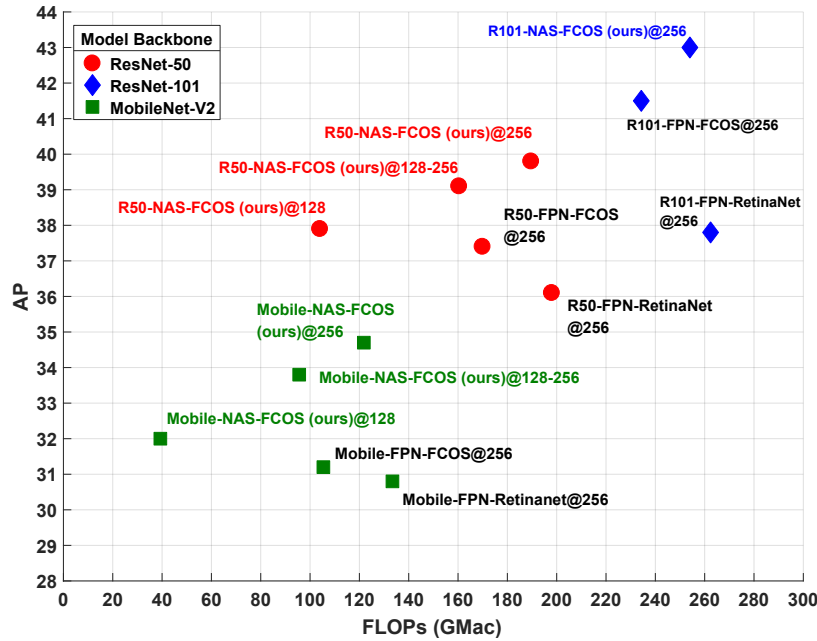


FIGURE 7.7. Diagram of the relationship between FLOPs and AP with different backbones. Points of different shapes represent different backbones. NAS-FCOS@128 has a slight increase in precision which also gains the advantage of computation quantity. One with 256 channels obtains the highest precision with more computation complexity. Using FPN channel width 128 and prediction head 256 (@128-256) offers a trade-off.

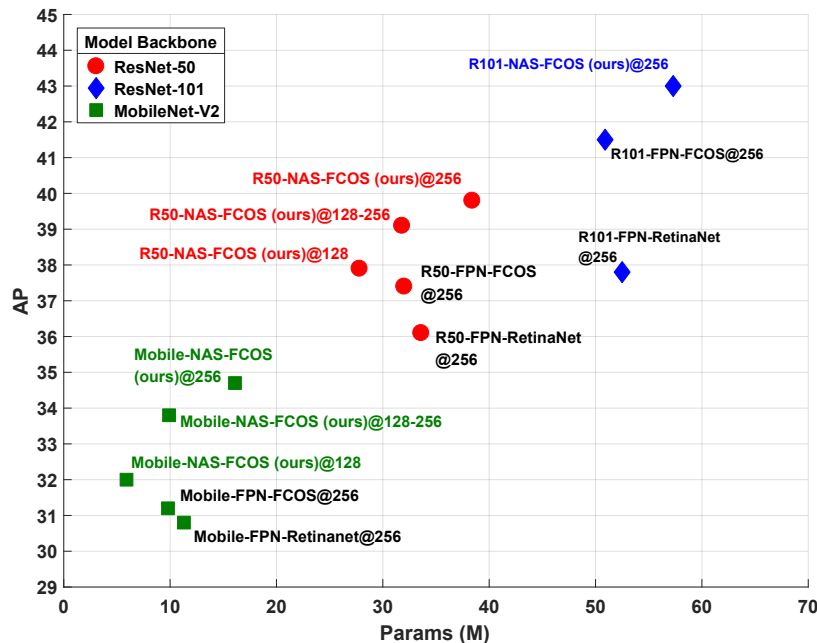


FIGURE 7.8. Diagram of the relationship between parameters and AP with different backbones. Adjusting the number of channels in the FPN structure and head helps to achieve a balance between accuracy and parameters.

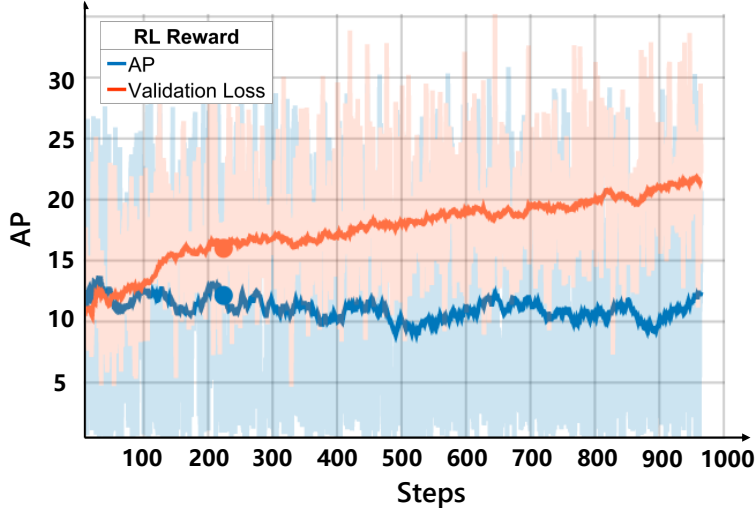


FIGURE 7.9. Comparison of two different RL reward designs. The vertical axis represents AP obtained from the proxy task on the validation dataset.

Decoder	Search Space	AP
FPN-FCOS @256	-	37.4
NAS-FCOS @256	h only	38.7
NAS-FCOS @256	f only	38.9
NAS-FCOS @256	$f + h$	39.8

TABLE 7.4. Comparisons between APs obtained under different search space with ResNet-50 backbone.

mapping from the decoder to the reward while the calculation of AP itself is complicated, which makes it difficult to learn this mapping within a limited number of iterations. In comparison, we clearly see the increase of AP with the validation loss as RL rewards (red curve in Fig. 7.9).

Effectiveness of Search Space

To further discuss the impact of the search spaces f and h , we design three experiments for verification. One is to search f with the original head being fixed, one is to search h with the original FPN being fixed and another is to search the entire decoder ($f+h$). As shown in Table 7.4, it turns out that searching f brings slightly more benefits than searching h only. And our progressive search which combines both f and h achieves a better result.

Impact of Deformable Convolution

As aforementioned, deformable convolutions are included in the set of candidate operations for both f and h , which are able to adapt to the geometric variations of objects. For fair comparison, we also replace the whole standard 3×3 convolutions with deformable 3×3 convolutions in FPN structure of the original FCOS and repeat them twice, making the FLOPs and parameters nearly equal to our searched model. The new model is therefore called DeformFPN-FCOS. It turns out that our NAS-FCOS model still achieves better performance (AP = 38.9 with FPN search only, and AP = 39.8 with both FPN and Head searched) than the DeformFPN-FCOS model (AP = 38.4) under this circumstance.

7.5 Conclusion

In this chapter, we have proposed to use Neural Architecture Search to further optimize the process of designing object detection networks. It is shown in this work that top-performing detectors can be efficiently searched using carefully designed proxy tasks, search strategies and model evaluation metrics. The experiments on COCO demonstrates the efficiency of our discovered model NAS-FCOS and its flexibility to be used with various backbone architectures.

Chapter 8

Conclusion

We have come to an age of AI applications getting involved into many aspects of everyday life. The efficiency and compactness of deep neural networks has becomes a major bottleneck for wider spread applications. One key question in designing these networks for perception tasks is how to efficiently reuse the features for multiple tasks. To achieve these goal, it is important for us to find a unified representation for different tasks.

This dissertation focuses on unifying two main categories of perception tasks, dense prediction tasks such as semantic segmentation and instance-wise prediction tasks such as object detection and instance segmentation. Previous methods use different paradigms for these tasks, densely prediction models follow a fully convolutional structure and instance-wise models follow a two-stage detect-then-segmentation structure. We reduce the gap between these two tasks and propose efficient network architecture optimization methods.

Fully convolutional instance segmentation. To reduce the gap between two-stage and fully-convolutional approached, we design SmalMask [3], a fully-convolutional instance segmentation network that has equivalent expressiveness as two-stage frameworks. Overcome the disadvantages of two-stage methods, i.e. low prediction resolution and inefficient second-stage computation, we design BlendMask [10], which is the first fully-convolutional framework that is faster and more accurate than two-stage approaches.

Fully convolutional multi-task networks. Because of the simple structure of BlendMask, we can easily share the intermediate features with other tasks to create very efficient multitask networks. We exploit this benefit to unify instance segmentation and semantic segmentation networks. Specifically, we design a more efficient instance segmentation network, DR1Mask and extend it for panoptic segmentation by adding merely one convolution layer [11]. The resulting network is two times faster than previous SOTA.

NAS for fully convolutional perception models. We then introduce our approach for efficient perception task oriented neural architecture search

algorithms. By caching backbone features and incorporating several speedup strategies, we are able to find efficient and high-performing task-specific sub-networks for video segmentation [5] and object detection [6] under the fully-convolutional regime.

Future work. BlendMask and DR1Mask offer useful insights into designing simple and efficient fully convolutional networks for complex perception tasks. The key idea is to merge contextual information and low-level details for more efficient instance representation. Blending module and DR1Module in Chapter 4 and 5 are in initial exploration for dynamic feature fusion for perception tasks. It remains future work to search for more efficient structures for panoptic segmentation under this framework, especially designing better dynamic modules. We hope the searching experiments in Chapter 6 and 7 could serve as examples for using NAS techniques for dynamic module analysis and complex perception proxy task design.

Next generation vision algorithms should target broader scenarios. With more challenging tasks such as 3D vision and more computation constrained platforms in the future, efficient and powerful perception models are urgently needed. There will be exponentially many multimodal input combinations for perception algorithms to deal with. Thus, finding a unified structure for multiple input and output tasks are crucial for the model scalability. It remains an unanswered question whether there is a general and powerful representation for different tasks, or if it can be manually designed. However, we are certain that automatic model design algorithms is very important and can shorten the time for us to find such models.

Appendix A

Details for Semantic Video Segmentation

A.1 Training Details of Static Baseline

The static baseline that we consider in the main text is *arch2* from [53], which we pre-train on CamVid [111] and CityScapes [112].

We utilise the ‘poly’ learning schedule [16] with the initial learning rates of $5e-2$ and $1e-2$ for the encoder and the decoder, respectively. As in [53], we set the weight for auxiliary losses to 0.3.

On CityScapes, we train for 1000 epochs with mini-batches of 28 examples each randomly scaled with the scale factor in range of $[0.5, 2.0]$ and randomly cropped to 800×800 with each side zero-padded accordingly. On CamVid, we train for 2000 epochs with mini-batches of 32 examples each randomly scaled with the scale factor in range of $[0.5, 2.0]$ and randomly cropped to 600×600 with each side zero-padded accordingly.

A.2 Search Space Aggregation Operations

In addition to the definitions of all operations in the main text, we provide the code for each aggregation operation written in PyTorch [123] in Listings 1, 2 and 3.

```

import torch
import torch.nn as nn
import torch.nn.functional as F

def resize(x1, x2):
    """Spatially resize two tensors to the largest size among them"""
    if x1.size()[2:] > x2.size()[2:]:
        x2 = nn.Upsample(size=x1.size()[2:], mode='bilinear')(x2)
    elif x1.size()[2:] < x2.size()[2:]:
        x1 = nn.Upsample(size=x2.size()[2:], mode='bilinear')(x1)
    return x1, x2

def conv(C_in, C_out, k, groups=1, stride=1, bias=False):
    return nn.Conv2d(C_in, C_out, k, stride, padding=k // 2, bias=bias, groups=groups)

class ParamSum(nn.Module):
    """ID 0: Summation with per-channel learnable weights per each input.

    Args:
        C (int) : number of input channels.

    """
    def __init__(self, C):
        super(ParamSum, self).__init__()
        self.a = nn.Parameter(torch.ones(C))
        self.b = nn.Parameter(torch.ones(C))

    def forward(self, x, y):
        x, y = resize(x, y)
        return (self.a.expand(x.size(0), -1)[: , :, None, None] * x +
                self.b.expand(y.size(0), -1)[: , :, None, None] * y)

class ConcatReduce(nn.Module):
    """ID 1: Channel-wise concatenation followed by grouped 1x1 convolution.

    Args:
        C (int) : number of input channels (also the number of groups).

    """
    def __init__(self, C):
        super(ConcatReduce, self).__init__()
        self.conv1x1 = nn.Sequential(
            nn.BatchNorm2d(2 * C),
            nn.ReLU(),
            conv(2 * C, C, 1, groups=C))

    def forward(self, x, y):
        x, y = resize(x, y)
        z = torch.cat([x, y], 1)
        return self.conv1x1(z)

```

LISTING 1. Aggregation Operations 0-1.


```

class PredOP(nn.Module):
    """ID 2: (weight) predictive operation, where
    the first input becomes a set of spatial convolutional
    filters (weights) applied on the second one.

    Args:
        C (int) : number of input channels.
        ksize (int, default=3) : kernel size of the resultant convolution.

    """
    def __init__(self, C, ksize=3):
        super(PredOP, self).__init__()
        self.ksize = ksize
        self.conv = nn.Sequential(
            nn.ReLU(), conv(C, C, 3, groups=C),
            nn.ReLU(), conv(C, C, 3, groups=C),
            nn.ReLU(), conv(C, ksize * ksize, 1), nn.Softmax(dim=1))

    def forward(self, x, y):
        x, y = resize(x, y)
        b, c, h, w = y.size()
        x = (self.conv(x)
             .permute(0, 2, 3, 1)
             .contiguous().view(b, h*w, self.ksize**2, 1))
        p = self.ksize // 2
        cols = F.unfold(
            y, kernel_size=self.ksize, dilation=p, padding=p, stride=1) # im2col
        out = torch.matmul(
            cols.permute(0, 2, 1).contiguous().view(b, -1, c, self.ksize**2), x)
        out = out.permute(0, 2, 1, 3).contiguous().view(b, c, h, w)
        return out

class BilSampling(nn.Module):
    """ID 3: Bilinear sampling of the first input with the affine grid
    predicted based on the values of the second input.

    Args:
        C (int) : number of input channels.

    """
    def __init__(self, C):
        super(BilSampling, self).__init__()
        self.conv_loc = nn.Sequential(conv(C, 3 * 2, 1), nn.ReLU())
        self.fc_loc = nn.Linear(3 * 2, 3 * 2)

    def forward(self, x, y):
        x, y = resize(x, y)
        yconv = self.conv_loc(y).mean(2).mean(2)
        theta = self.fc_loc(yconv).view(-1, 2, 3)
        grid = F.affine_grid(theta, x.size())
        x = F.grid_sample(x, grid)
        return x + y

```

```

class Conv3d(nn.Module):
    """ID 4: 3D-convolution, where
    two inputs are stacked together forming a new dimension
    with 2x3x3 grouped convolution applied on top.

    Args:
        C (int) : number of input channels (also the number of groups).
        ksize (int, default=3) : kernel size in (2, ksize, ksize) convolution.

    """
    def __init__(self, C, ksize=3):
        super(Conv3d, self).__init__()
        p = int(ksize // 2)
        self.conv = torch.nn.Conv3d(
            C, C, kernel_size=(2, ksize, ksize), padding=(0, p, p),
            groups=C, bias=False)

    def forward(self, x, y):
        x, y = resize(x, y)
        return self.conv(torch.stack([x,y], 2)).squeeze(2)

class DenseAttention(nn.Module):
    """ID 5: Element-wise multiplication between the first input and
    the sigmoid-activated second one.

    """
    def __init__(self):
        super(DenseAttention, self).__init__()

    def forward(self, x, y):
        x, y = resize(x, y)
        return x * F.sigmoid(y)

```

LISTING 3. Aggregation Operations 4-5.

Bibliography

- [1] A. Kirillov, R. B. Girshick, K. He, and P. Dollár, “Panoptic feature pyramid networks”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 6399–6408. [Online]. Available: http://openaccess.thecvf.com/content/_CVPR/_2019/html/Kirillov/_Panoptic/_Feature/_Pyramid/_Networks/_CVPR/_2019/_paper.html.
- [2] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017, pp. 2961–2969.
- [3] H. Chen, Z. Tian, and C. Shen, “Smooth and aligned features for real-time instance segmentation”, in *Work in progress*, 2020.
- [4] V. Nekrasov, H. Chen, C. Shen, and I. Reid, “Fast neural architecture search of compact semantic segmentation models via auxiliary cells”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.
- [5] V. Nekrasov, H. Chen, C. Shen, and I. D. Reid, “Architecture search of dynamic cells for semantic video segmentation”, in *IEEE Winter Conference on Applications of Computer Vision, WACV 2020, Snowmass Village, CO, USA, March 1-5, 2020*, IEEE, 2020, pp. 1959–1968. DOI: [10.1109/WACV45572.2020.9093531](https://doi.org/10.1109/WACV45572.2020.9093531). [Online]. Available: <https://doi.org/10.1109/WACV45572.2020.9093531>.
- [6] N. Wang, Y. Gao, H. Chen, P. Wang, Z. Tian, C. Shen, and Y. Zhang, “NAS-FCOS: fast neural architecture search for object detection”, in *CVPR*, IEEE, 2020, pp. 11 940–11 948.
- [7] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation”, in *Proc. Eur. Conf. Comp. Vis.*, 2018, pp. 833–851. DOI: [10.1007/978-3-030-01234-2_49](https://doi.org/10.1007/978-3-030-01234-2_49). [Online]. Available: https://doi.org/10.1007/978-3-030-01234-2_49.
- [8] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks”, in *Proc. Advances in Neural Inf. Process. Syst.*, 2015, pp. 91–99.

- [9] Z. Tian, C. Shen, H. Chen, and T. He, “FCOS: Fully convolutional one-stage object detection”, in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.
- [10] H. Chen, K. Sun, Z. Tian, C. Shen, Y. Huang, and Y. Yan, “Blend-mask: Top-down meets bottom-up for instance segmentation”, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, IEEE, 2020, pp. 8570–8578. DOI: [10.1109/CVPR42600.2020.00860](https://doi.org/10.1109/CVPR42600.2020.00860). [Online]. Available: <https://doi.org/10.1109/CVPR42600.2020.00860>.
- [11] H. Chen, Z. Tian, and C. Shen, “Segmenting background for free with rank-1 dynamic convolution”, in *Work in progress*, 2020.
- [12] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2015.
- [13] G. Lin, A. Milan, C. Shen, and I. D. Reid, “RefineNet: Multi-path refinement networks for high-resolution semantic segmentation”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.
- [14] F. Yu, V. Koltun, and T. A. Funkhouser, “Dilated residual networks”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.
- [15] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.
- [16] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”, *IEEE Trans. Pattern Anal. Mach. Intell.*, 2018.
- [17] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation”, in *Proc. Eur. Conf. Comp. Vis.*, 2018.
- [18] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, “Icnet for real-time semantic segmentation on high-resolution images”, in *Proc. Eur. Conf. Comp. Vis.*, 2018.
- [19] V. Nekrasov, C. Shen, and I. D. Reid, “Light-weight refinenet for real-time semantic segmentation”, in *Proc. British Machine Vis. Conf.*, 2018.
- [20] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, “Deep feature flow for video recognition”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.

-
- [21] R. Gadde, V. Jampani, and P. V. Gehler, “Semantic video cnns through representation warping”, in *Proc. IEEE Int. Conf. Comp. Vis.*, 2017.
 - [22] S. Jain, X. Wang, and J. Gonzalez, “Accel: A corrective fusion network for efficient semantic segmentation on video”, *arXiv: Comp. Res. Repository*, vol. abs/1807.06667, 2018.
 - [23] Y. Xu, T. Fu, H. Yang, and C. Lee, “Dynamic video segmentation network”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.
 - [24] D. Nilsson and C. Sminchisescu, “Semantic video segmentation by gated recurrent flow propagation”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.
 - [25] Y. Li, J. Shi, and D. Lin, “Low-latency video semantic segmentation”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.
 - [26] S. Chandra, C. Couprie, and I. Kokkinos, “Deep spatio-temporal random fields for efficient video segmentation”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.
 - [27] S. Chandra and I. Kokkinos, “Fast, exact and multi-scale inference for semantic image segmentation with deep gaussian crfs”, in *Proc. Eur. Conf. Comp. Vis.*, 2016.
 - [28] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017, pp. 2980–2988.
 - [29] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. Berg, “SSD: Single shot multibox detector”, in *Proc. Eur. Conf. Comp. Vis.*, 2016, pp. 21–37.
 - [30] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement”, in *arXiv preprint arXiv:1804.02767*, 2018.
 - [31] T. Kong, F. Sun, H. Liu, Y. Jiang, and J. Shi, “Foveabox: Beyond anchor-based object detector”, in *arXiv preprint arXiv:1904.03797*, 2019.
 - [32] H. Law and J. Deng, “Cornersnet: Detecting objects as paired keypoints”, in *Proc. Eur. Conf. Comp. Vis.*, 2018, pp. 734–750.
 - [33] X. Zhou, D. Wang, and P. Krähenbühl, “Objects as points”, in *arXiv preprint arXiv:1904.07850*, 2019.
 - [34] C. Zhu, Y. He, and M. Savvides, “Feature selective anchor-free module for single-shot object detection”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

- [35] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, “Fully convolutional instance-aware semantic segmentation”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017, pp. 4438–4446. DOI: [10.1109/CVPR.2017.472](https://doi.org/10.1109/CVPR.2017.472). [Online]. Available: <https://doi.org/10.1109/CVPR.2017.472>.
- [36] J. Dai, Y. Li, K. He, and J. Sun, “R-FCN: object detection via region-based fully convolutional networks”, in *Proc. Advances in Neural Inf. Process. Syst.*, 2016, pp. 379–387. [Online]. Available: <http://papers.nips.cc/paper/6465-r-fcn-object-detection-via-region-based-fully-convolutional-networks>.
- [37] L. Chen, A. Hermans, G. Papandreou, F. Schroff, P. Wang, and H. Adam, “Masklab: Instance segmentation by refining object detection with semantic and direction features”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018, pp. 4013–4022. DOI: [10.1109/CVPR.2018.00422](https://doi.org/10.1109/CVPR.2018.00422). [Online]. Available: http://openaccess.thecvf.com/content/_cvpr/_2018/html/Chen_MaskLab_Instance_Segmentation_CVPR_2018_paper.html.
- [38] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, “YOLACT: real-time instance segmentation”, *Proc. IEEE Int. Conf. Comp. Vis.*, vol. abs/1904.02689, 2019. arXiv: [1904.02689](https://arxiv.org/abs/1904.02689). [Online]. Available: <http://arxiv.org/abs/1904.02689>.
- [39] B. D. Brabandere, D. Neven, and L. V. Gool, “Semantic instance segmentation with a discriminative loss function”, *arXiv: Comp. Res. Repository*, vol. abs/1708.02551, 2017. arXiv: [1708.02551](https://arxiv.org/abs/1708.02551). [Online]. Available: <http://arxiv.org/abs/1708.02551>.
- [40] Y. Liu, S. Yang, B. Li, W. Zhou, J. Xu, H. Li, and Y. Lu, “Affinity derivation and graph merge for instance segmentation”, in *Proc. Eur. Conf. Comp. Vis.*, 2018, pp. 708–724. DOI: [10.1007/978-3-030-01219-9_42](https://doi.org/10.1007/978-3-030-01219-9_42). [Online]. Available: https://doi.org/10.1007/978-3-030-01219-9_42.
- [41] J. Dai, K. He, Y. Li, S. Ren, and J. Sun, “Instance-sensitive fully convolutional networks”, in *Proc. Eur. Conf. Comp. Vis.*, 2016, pp. 534–549. DOI: [10.1007/978-3-319-46466-4_32](https://doi.org/10.1007/978-3-319-46466-4_32). [Online]. Available: https://doi.org/10.1007/978-3-319-46466-4_32.
- [42] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning”, in *Proc. Int. Conf. Learn. Representations*, 2017.

-
- [43] H. Zhou, M. Yang, J. Wang, and W. Pan, “BayesNAS: A bayesian approach for neural architecture search”, in *Proc. Int. Conf. Mach. Learn.*, 2019.
- [44] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei, “Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.
- [45] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture search”, in *Proc. Int. Conf. Learn. Representations*, 2019.
- [46] H. Cai, L. Zhu, and S. Han, “Proxylessnas: Direct neural architecture search on target task and hardware”, in *Proc. Int. Conf. Learn. Representations*, 2019.
- [47] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, “Single path one-shot neural architecture search with uniform sampling”, in *arXiv preprint arXiv:1904.00420*, 2019.
- [48] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, “Single-path NAS: Designing hardware-efficient convnets in less than 4 hours”, in *arXiv preprint arXiv:1904.02877*, 2019.
- [49] Y. Chen, T. Yang, X. Zhang, G. Meng, C. Pan, and J. Sun, “DetNAS: Neural architecture search on object detection”, in *Proc. Advances in Neural Inf. Process. Syst.*, 2019.
- [50] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [51] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning”, *Proc. Int. Conf. Learn. Representations*, 2017.
- [52] L. Chen, M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens, “Searching for efficient multi-scale architectures for dense image prediction”, *Proc. Advances in Neural Inf. Process. Syst.*, 2018.
- [53] V. Nekrasov, H. Chen, C. Shen, and I. D. Reid, “Fast neural architecture search of compact semantic segmentation models via auxiliary cells”, *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.
- [54] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017, pp. 2117–2125.

- [55] G. Ghiasi, T.-Y. Lin, R. Pang, and Q. V. Le, “NAS-FPN: Learning scalable feature pyramid architecture for object detection”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.
- [56] H. Liu, C. Peng, C. Yu, J. Wang, X. Liu, G. Yu, and W. Jiang, “An end-to-end network for panoptic segmentation”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.
- [57] T. Zhao and X. Wu, “Pyramid feature attention network for saliency detection”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.
- [58] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016, pp. 779–788. DOI: [10.1109/CVPR.2016.91](https://doi.org/10.1109/CVPR.2016.91). [Online]. Available: <https://doi.org/10.1109/CVPR.2016.91>.
- [59] P. H. O. Pinheiro, R. Collobert, and P. Dollár, “Learning to segment object candidates”, in *Proc. Advances in Neural Inf. Process. Syst.*, 2015, pp. 1990–1998. [Online]. Available: <http://papers.nips.cc/paper/5852-learning-to-segment-object-candidates>.
- [60] X. Chen, R. B. Girshick, K. He, and P. Dollár, “Tensormask: A foundation for dense object segmentation”, *Proc. IEEE Int. Conf. Comp. Vis.*, vol. abs/1903.12174, 2019. arXiv: [1903.12174](https://arxiv.org/abs/1903.12174). [Online]. Available: <http://arxiv.org/abs/1903.12174>.
- [61] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, “Single-shot refinement neural network for object detection”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018, pp. 4203–4212. DOI: [10.1109/CVPR.2018.00442](https://doi.org/10.1109/CVPR.2018.00442). [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_Single-Shot_Refinement_Neural_CVPR_2018_paper.html.
- [62] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, “Deformable convolutional networks”, in *Proc. IEEE Int. Conf. Comp. Vis.*, 2017, pp. 764–773. DOI: [10.1109/ICCV.2017.89](https://doi.org/10.1109/ICCV.2017.89). [Online]. Available: <https://doi.org/10.1109/ICCV.2017.89>.
- [63] Y. Chen, C. Han, N. Wang, and Z. Zhang, “Revisiting feature alignment for one-stage object detection”, *arXiv: Comp. Res. Repository*, vol. abs/1908.01570, 2019. arXiv: [1908.01570](https://arxiv.org/abs/1908.01570). [Online]. Available: <http://arxiv.org/abs/1908.01570>.

- [64] R. Zhang, “Making convolutional networks shift-invariant again”, in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7324–7334. [Online]. Available: <http://proceedings.mlr.press/v97/zhang19a.html>.
- [65] J. Wang, K. Chen, S. Yang, C. C. Loy, and D. Lin, “Region proposal by guided anchoring”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 2965–2974.
- [66] X. Zhu, H. Hu, S. Lin, and J. Dai, “Deformable convnets V2: more deformable, better results”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 9308–9316. [Online]. Available: http://openaccess.thecvf.com/content/_CVPR/_2019/html/Zhu_Deformable_ConvNets_V2_More_Deformable_Better_Results_CVPR_2019_paper.html.
- [67] W. Leler, “Human vision, anti-aliasing, and the cheap 4000 line display”, in *Proc. Conf. Computer Graphics Interactive Techniques, Sig-Graph*, 1980, pp. 308–313. DOI: [10.1145/800250.807509](https://doi.org/10.1145/800250.807509). [Online]. Available: <https://doi.org/10.1145/800250.807509>.
- [68] Y. Li, Y. Chen, N. Wang, and Z. Zhang, “Scale-aware trident networks for object detection”, *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.
- [69] B. Singh, M. Najibi, and L. S. Davis, “SNIPER: efficient multi-scale training”, in *Proc. Advances in Neural Inf. Process. Syst.*, 2018, pp. 9333–9343. [Online]. Available: <http://papers.nips.cc/paper/8143-sniper-efficient-multi-scale-training>.
- [70] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: common objects in context”, in *Proc. Eur. Conf. Comp. Vis.*, 2014, pp. 740–755. DOI: [10.1007/978-3-319-10602-1_48](https://doi.org/10.1007/978-3-319-10602-1_48). [Online]. Available: https://doi.org/10.1007/978-3-319-10602-1_48.
- [71] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016, pp. 770–778. DOI: [10.1109/CVPR.2016.90](https://doi.org/10.1109/CVPR.2016.90). [Online]. Available: <https://doi.org/10.1109/CVPR.2016.90>.
- [72] A. Arnab and P. H. S. Torr, “Bottom-up instance segmentation using deep higher-order CRFs”, in *Proc. British Machine Vis. Conf.*, 2016. [Online]. Available: <http://www.bmva.org/bmvc/2016/papers/paper019/index.html>.

- [73] D. Neven, B. D. Brabandere, M. Proesmans, and L. V. Gool, “Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 8837–8845. [Online]. Available: http://openaccess.thecvf.com/content/_CVPR/_2019/html/Neven_Instance_Segmentation_by_Jointly_Optimizing_Spatial_Embeddings_and_Clustering_Bandwidth_CVPR_2019_paper.html.
- [74] C. Fu, T. L. Berg, and A. C. Berg, “IMP: instance mask projection for high accuracy semantic segmentation of things”, *arXiv: Comp. Res. Repository*, vol. abs/1906.06597, 2019. arXiv: 1906.06597. [Online]. Available: <http://arxiv.org/abs/1906.06597>.
- [75] E. Xie, P. Sun, X. Song, W. Wang, X. Liu, D. Liang, C. Shen, and P. Luo, “PolarMask: Single shot instance segmentation with polar representation”, *arXiv: Comp. Res. Repository*, 2019, arxiv.org/abs/1909.13226.
- [76] X. Wang, T. Kong, C. Shen, Y. Jiang, and L. Li, “Solo: Segmenting objects by locations”, *arXiv preprint arXiv:1912.04488*, 2019.
- [77] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks”, in *Proc. Eur. Conf. Comp. Vis.*, 2016, pp. 630–645.
- [78] R. B. Girshick, “Fast R-CNN”, in *Proc. IEEE Int. Conf. Comp. Vis.*, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169. [Online]. Available: <https://doi.org/10.1109/ICCV.2015.169>.
- [79] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, “YOLACT++: Better real-time instance segmentation”, *arXiv preprint arXiv:1912.06218*, 2019.
- [80] A. Gupta, P. Dollar, and R. Girshick, “LVIS: A dataset for large vocabulary instance segmentation”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
- [81] A. Kirillov, Y. Wu, K. He, and R. Girshick, “Pointrend: Image segmentation as rendering”, *arXiv preprint arXiv:1912.08193*, 2019.
- [82] Z. Tian, C. Shen, and H. Chen, “Conditional convolutions for instance segmentation”, *CoRR*, vol. abs/2003.05664, 2020.
- [83] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2018.
- [84] X. Wang, R. B. Girshick, A. Gupta, and K. He, “Non-local neural networks”, in *CVPR*, IEEE Computer Society, 2018, pp. 7794–7803.

- [85] X. Li, Y. Yang, Q. Zhao, T. Shen, Z. Lin, and H. Liu, “Spatial pyramid based graph reasoning for semantic segmentation”, in *CVPR*, IEEE, 2020, pp. 8947–8956.
- [86] B. Cheng, M. D. Collins, Y. Zhu, T. Liu, T. S. Huang, H. Adam, and L. Chen, “Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation”, in *CVPR*, IEEE, 2020, pp. 12 472–12 482.
- [87] Y. Xiong, R. Liao, H. Zhao, R. Hu, M. Bai, E. Yumer, and R. Urtasun, “Upsnet: A unified panoptic segmentation network”, in *CVPR*, Computer Vision Foundation / IEEE, 2019, pp. 8818–8826.
- [88] Y. Yang, H. Li, X. Li, Q. Zhao, J. Wu, and Z. Lin, “Sognet: Scene overlap graph network for panoptic segmentation”, in *AAAI*, AAAI Press, 2020, pp. 12 637–12 644.
- [89] G. Neubig, C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastopoulos, M. Ballesteros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette, Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson, N. Saphra, S. Swayamdipta, and P. Yin, “DyNet: The dynamic neural network toolkit”, *arXiv preprint arXiv:1701.03980*, 2017.
- [90] L. Hou, L. Shang, X. Jiang, and Q. Liu, “Dynabert: Dynamic BERT with adaptive width and depth”, *CoRR*, vol. abs/2004.04037, 2020.
- [91] Y. Li, L. Song, Y. Chen, Z. Li, X. Zhang, X. Wang, and J. Sun, “Learning dynamic routing for semantic segmentation”, in *CVPR*, IEEE, 2020, pp. 8550–8559.
- [92] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi, “Supermasks in superposition”, *CoRR*, vol. abs/2006.14769, 2020.
- [93] V. Dumoulin, E. Perez, N. Schucher, F. Strub, H. d. Vries, A. Courville, and Y. Bengio, “Feature-wise transformations”, *Distill*, 2018, <https://distill.pub/2018/feature-wise-transformations>. DOI: [10.23915/distill.00011](https://doi.org/10.23915/distill.00011).
- [94] Y. Li, N. Wang, J. Shi, X. Hou, and J. Liu, “Adaptive batch normalization for practical domain adaptation”, *Pattern Recognit.*, vol. 80, pp. 109–117, 2018.
- [95] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville, “Film: Visual reasoning with a general conditioning layer”, in *AAAI*, AAAI Press, 2018, pp. 3942–3951.

- [96] H. Zhang, K. J. Dana, J. Shi, Z. Zhang, X. Wang, A. Tyagi, and A. Agrawal, “Context encoding for semantic segmentation”, in *CVPR*, IEEE Computer Society, 2018, pp. 7151–7160.
- [97] H. Zhao, Y. Zhang, S. Liu, J. Shi, C. C. Loy, D. Lin, and J. Jia, “Psanet: Point-wise spatial attention network for scene parsing”, in *ECCV (9)*, ser. Lecture Notes in Computer Science, vol. 11213, Springer, 2018, pp. 270–286.
- [98] Y. Wen, D. Tran, and J. Ba, “Batchensemble: An alternative approach to efficient ensemble and lifelong learning”, in *ICLR*, OpenReview.net, 2020.
- [99] M. W. Dusenberry, G. Jerfel, Y. Wen, Y. Ma, J. Snoek, K. A. Heller, B. Lakshminarayanan, and D. Tran, “Efficient and scalable bayesian neural nets with rank-1 factors”, *CoRR*, vol. abs/2005.07186, 2020.
- [100] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, “Linformer: Self-attention with linear complexity”, *arXiv: Comp. Res. Repository*, vol. abs/2006.04768, 2020.
- [101] K. Choromanski, V. Likhoshesterov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller, “Rethinking attention with performers”, *arXiv: Comp. Res. Repository*, vol. abs/2009.14794, 2020.
- [102] H. Wang, Y. Zhu, B. Green, H. Adam, A. L. Yuille, and L. Chen, “Axial-deeplab: Stand-alone axial-attention for panoptic segmentation”, in *Proc. Eur. Conf. Comp. Vis.*, 2020, pp. 108–126.
- [103] E. Shelhamer, K. Rakelly, J. Hoffman, and T. Darrell, “Clockwork convnets for video semantic segmentation”, in *Proc. Eur. Conf. Comp. Vis.*, 2016.
- [104] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey”, in *J. Mach. Learn. Res.*, 2019.
- [105] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition”, *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.
- [106] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy, “Progressive neural architecture search”, in *Proc. Eur. Conf. Comp. Vis.*, 2018.

-
- [107] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, “Efficient neural architecture search via parameter sharing”, in *Proc. Int. Conf. Mach. Learn.*, 2018.
- [108] S. Xie, H. Zheng, C. Liu, and L. Lin, “SNAS: stochastic neural architecture search”, *arXiv: Comp. Res. Repository*, vol. abs/1812.09926, 2018.
- [109] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation”, *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.
- [110] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial transformer networks”, in *Proc. Advances in Neural Inf. Process. Syst.*, 2015.
- [111] G. J. Brostow, J. Fauqueur, and R. Cipolla, “Semantic object classes in video: A high-definition ground truth database”, *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.
- [112] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The cityscapes dataset for semantic urban scene understanding”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016.
- [113] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation”, 12, vol. 39, 2017, pp. 2481–2495.
- [114] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms”, vol. abs/1707.06347, 2017.
- [115] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization”, vol. abs/1412.6980, 2014.
- [116] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016.
- [117] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions”, *Proc. Int. Conf. Learn. Representations*, 2016.
- [118] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, “Efficient neural architecture search via parameter sharing”, in *Proc. Int. Conf. Mach. Learn.*, 2018.
- [119] Y. Wu and K. He, “Group normalization”, in *Proc. Eur. Conf. Comp. Vis.*, 2018, pp. 3–19.

-
- [120] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms”, in *arXiv preprint arXiv:1707.06347*, 2017.
 - [121] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018, pp. 4510–4520.
 - [122] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks”, in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.
 - [123] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch”, in *NeurIPS-W*, 2017.