Semantic Image Segmentation and Other Dense Per-Pixel Tasks: Practical Approaches

THE UNIVERSITY *of* ADELAIDE

Vladimir Nekrasov

School of Computer Science

The University of Adelaide

A thesis submitted for the degree of

Doctor of Philosophy

July 2020

Contents

List of Figures

List of Tables

Abstract

Computer vision-based and deep learning-driven applications and devices are now a part of our everyday life: from modern smartphones with an ever increasing number of cameras and other sensors to autonomous vehicles such as driverless cars and self-piloting drones. Even though a large portion of the algorithms behind those systems has been known for ages, the computational power together with the abundance of labelled data were lacking until recently. Now, following the Occam's razor principle, we should start re-thinking those algorithms and strive towards their further simplification, both to improve our own understanding and expand the realm of their practical applications.

With those goals in mind, in this work we will concentrate on a particular type of computer vision tasks that predict a certain quantity of interest for each pixel in the input image – these are so-called *dense per-pixel tasks*. This choice is not by chance: while there has been a huge amount of works concentrated on per-image tasks such as image classification with levels of performance reaching nearly 100%, dense per-pixel tasks bring a different set of challenges that traditionally require more computational resources and more complicated approaches. Throughout this thesis, our focus will be on reducing these computational requirements and instead presenting simple approaches to build *practical vision systems* that can be used in a variety of settings – *e.g.* indoors or outdoors, on low-resolution or high-resolution images, solving a single task or multiple tasks at once, running on modern GPU cards or on embedded devices such as Jetson TX.

In the first part of the manuscript we will adapt an existing powerful but slow semantic segmentation network into a faster and competitive one through a manual re-design and analysis of its building blocks. With this approach, we will achieve nearly 3× decrease in the number of parameters and in the runtime of the network with an equally high accuracy. In the second part we then will alter this compact network in order to solve multiple dense per-pixel tasks at once, still in real-time. We will also demonstrate the value of predicting multiple quantities at once, as an example creating a 3D semantic reconstruction of the scene.

In the third part, we will move away from the manual design and instead will rely on reinforcement learning to automatically traverse the search space of compact semantic segmentation architectures. While the majority of architecture search

methods are computationally extremely expensive even for image classification, we will present a solution that requires only 2 generic GPU cards. Finally, in the last part we will extend our automatic architecture search solution to discover tiny but still competitive networks with less than 300K parameters taking only 1.5MB of a disk space.

HDR Thesis Declaration

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

Signature V Date 14/July/2020

Acknowledgements

First of all, thanks to my supervisors – Prof. Ian Reid and Prof. Chunhua Shen, to whom I am grateful for their invaluable comments and suggestions, and insightful discussions. It was their advice to start my PhD journey by first re-implementing RefineNet and then showcasing it in a live demo. Those initial months taught me a lot about pitfalls involved in training a state-of-the-art segmentation model. Thanks also to Tong Shen who was of immense help to me during that period.

Thanks to my coauthors and collaborators, especially to Hao Chen, with whom we spent hours tweaking and adjusting our neural architecture search algorithm. It was fun to make it work from scratch.

Of course, nothing of this would have happened without a significant financial support; for that, I am grateful to the Australian Centre for Robotic Vision and the University of Adelaide. I am also thankful to Thuy Mai for arranging conference trips and handling numerous reimbursement forms.

Special thanks go to Prof. Jaesik Choi (currently at KAIST), who introduced me to deep learning and the task of semantic segmentation.

Last but not least, I would like to thank my mother Elena and my partner Jenica who were with me throughout the journey.

1 Introduction

Imagine yourself being tasked with associating elements from two disjoint sets, for example, a set of images of paintings and a set of names of artists, or a set of photos of birds and a set of sounds they produce. What would your strategy be?

Perhaps, for each pair of elements you would come up with some sort of number – a compatibility estimate. Or you would first try to reduce the burden of the task by grouping elements in each set based on some sort of similarity measure. Regardless of the strategy you choose, you will be hard-pressed to come up with a method that does not require you to seek patterns across the elements of both sets. Those patterns, in turn, emerge from the features that the elements possess. In our example, some features you could come up with would be the proportion of the red colour present in the painting, or the longevity of a bird's chirping. Obviously, some features would be good and some would be bad – for example, if all the sounds were of the same length, the sound longevity feature would give you no useful information at all.

*Evidently, machine learning algorithms tend to follow the same logic: namely, given some sort of data and some features, they would come up with some set of rules (*a solution*) describing patterns that tie the features with the corresponding data points.* **Importantly, where do the features come from?** *As per the examples above, the features can be (and often are) manually engineered, but this*

brings a disadvantage of not being able to know whether the given feature is good or bad. Can we do better – can we automatically extract useful features?

As it turns out, we can. The big shift in the paradigm of feature engineering came in *2010s* with the resurgence of *deep learning* (Krizhevsky et al., 2012; LeCun et al., 2015). Deep learning represents a set of algorithms built upon the concepts of artificial neurons and neural networks (Rosenblatt, 1958) loosely based on biological neurons. Given raw data sources, *e.g.* images, text or audio, and the desired objective function, deep neural networks *learn* the most appropriate set of features – often not readily interpretable to human experts.

In this manuscript we will focus on applications of deep learning to computer vision, in particular, on so-called *dense per-pixel tasks* where one is interested in predicting a certain quantity of interest for each pixel in the input image (or images) – be it a regression or classification vector (for example, a semantic labelling in the case of semantic segmentation, or distance to the object from the camera center – for the task of depth estimation; see Fig. 1.1).

This choice is not by chance: while there has been a huge amount of works concentrated on per-image tasks such as image classification (*i.e.* predicting a single label per image – *e.g.* a breed of a dog) with levels of performance reaching nearly 100%, dense per-pixel tasks bring a different set of challenges that need more computational resources and more complicated approaches. In particular, one set of challenges is simply due to the high-dimensionality of the dense output: this not only requires a larger memory for data processing but also larger networks for solving a more difficult problem. Another set of challenges is due to a significant class imbalance – oftentimes, the majority of the pixels in the image belong to an over-represented class, such as *background*, or *road*. This leads to longer training times and the necessity of more data. Hence, one of the goals of this manuscript is to build solutions that alleviate these problems and the need for an extreme processing power.

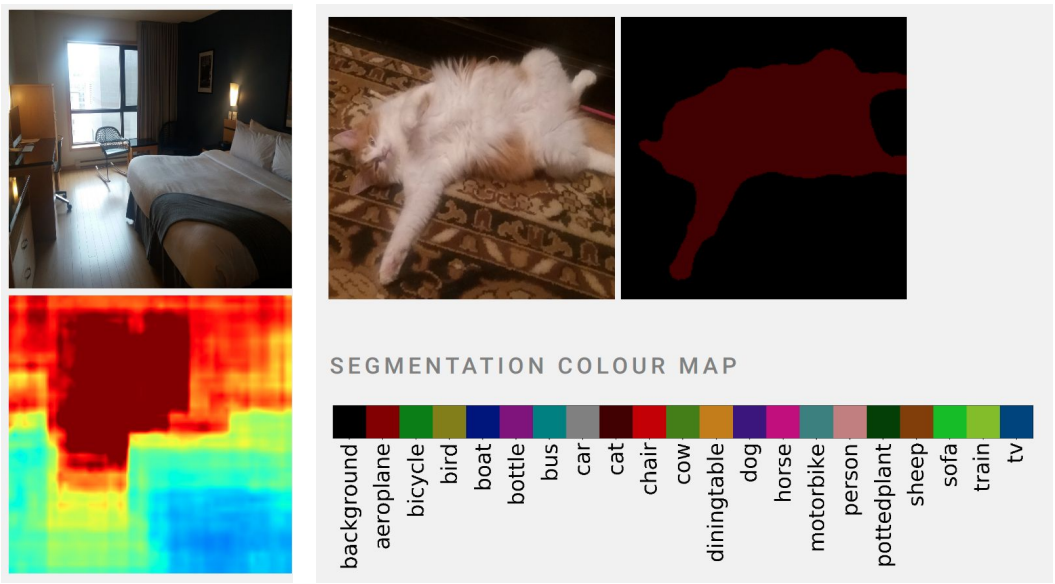


Figure 1.1: Examples of dense per-pixel tasks in action. *(a)* Depth estimation (bottom) of the input image (top). The depth colours are ranging from light blue indicating that the pixel is close to the camera to dark red – meaning that the pixel is further away from the camera. *(b)* Semantic segmentation (right) of the input image (left) with the given colour map (bottom). The results were produced by running a Light-Weight RefineNet network described in later chapters inside an internet browser at https://drsleep.github.io/demos – you can try it yourself! No post-processing was used.

On a related note, largely thanks to deep learning, computer vision-driven applications and devices are now a part of our everyday life: from modern smartphones with an ever increasing number of cameras to autonomous vehicles such as driverless cars and self-piloting drones. Dense per-pixel tasks, in particular, play an important role in perceiving the world around us through the camera lenses, as they process the given image holistically, pixel-by-pixel, opposed to a plain recognition of classes present in the image. These sorts of applications usually come with two major requirements: namely, performing computations in a reasonable amount of time (we will define this more rigorously below) and computing multiple quantities of interest at once. For example, in order for an autopilot of a vision-based driverless car to plan a safe trajectory, it must at least know what objects and obstacles surround it, as well as how far those are at any given time. In that regard, yet another goal of this manuscript is to lay down simple foundations for building and

diagnosing such practical dense vision systems. As the term *"practical"* is oftentimes domain-specific, let us first clarify its connotation here: **a practical vision system is the one that performs its task (or tasks) accurately enough in real-time**. For the sake of concrete numbers, we will assume that *"real-time"* implies the runtime of at least 30 frames per second (FPS) subject to an underlying device and input data[1]. The accuracy constraint is again domain-specific, and we will discuss actual numbers in later chapters.

While automatic feature extraction is one of attractive sides of deep learning, it turns out it can even be extended to *automatic architecture design* (also called *neural architecture search* (Zoph and Le, 2017)), where the structure of the neural network used to solve the task is learned together with its parameters. Even for small image-level problems this approach often requires an extensive amount of computational resources, hence it is non-trivial to scale it up for dense per-pixel tasks such as semantic segmentation. Thus, another goal of this manuscript is to come up with a more efficient way of searching for dense neural networks automatically.

1.1 Motivation

A curious and attentive reader might be wondering why the resurgence of deep learning only happened around 2010 when artificial neural networks had been known for more than 50 years? Two main reasons are large-scale datasets and greater computational resources. In particular, the ImageNet dataset (Deng et al., 2009) with more than 1M images and 1000 classes significantly advanced research in image classification and consequently led to the creation of datasets suitable for other tasks, for example, MS COCO (T. Lin et al., 2014) for object detection and semantic segmentation, and CityScapes (Cordts et al., 2016) for semantic segmentation and depth estimation. On the other hand, this coincided with hardware improvements of graphics processing units (GPUs), well-suited for quickly performing many algebraic computations in parallel on individual neurons or groups of neurons (also called *layers*) in neural networks.

[1] 30FPS, or, more precisely, 29.97FPS, has been one of the television standards for years.

If the trends are to continue, i.e. if data and hardware resources keep improving, why should we care about building compact and practical vision systems now when we can simply wait for those advances to take place?

First of all, our focus is on software and on solutions that are hardware-agnostic, making any hardware improvements complementary.

Secondly, even with hardware improvements in-place, devices with limited resources (such as mobile phones and embedded platforms) will always benefit from having more efficient models.

Thirdly, compact systems are much more amenable to analysis, debugging and prototyping than unnecessarily complicated ones.

Lastly, training of larger neural networks causes a significantly higher amount of CO_2 emissions (Strubell et al., 2019). While this manuscript is not explicitly aimed at targetting this particular problem, it is important to raise the awareness of the community and be mindful of various trade-offs involved in building deep learning-based solutions. For example, neural architecture search methods can often require several thousand GPU-days, thus making it important to develop more cost-effective solutions (presented in Chapters 5 and 6).

With those considerations in mind, dense per-pixel tasks in particular constitute a great and non-trivial case study as existing solutions tend to be over-complicated and consume a significant amount of resources.

1.2 Contributions

Aligned with the major goals of this manuscript mentioned above, we summarise its contributions in the following:

- in this manuscript we outline a way of building a practical, real-time semantic segmentation network by carefully rethinking an existing but slow segmentation network;
- in particular, we demonstrate the redundancy in certain operations which can be removed without a significant performance degradation;

- furthermore, we extend this approach to multi-task learning with a single network solving multiple dense per-pixel tasks at once in real-time;
- at the same time, this multi-task network is able to learn various domains simultaneously, such as indoor and outdoor scenes;
- additionally, we show that its predictions of depth and semantic segmentation made in 2D can be used to produce a 3D semantic map of the environment.

We also make significant contributions in the area of automated architecture search for dense per-pixel tasks, specifically:

- we propose an algorithm to find high-performing compact networks for semantic segmentation with a limited number of resources;
- concretely, our solution is able to reliably estimate the quality and the performance potential of any given architecture in less than two minutes;
- furthermore, we extend this approach to search for a larger number of neural network components;
- which, in turn, permits discovering high-performing tiny networks with less than 300K parameters.

1.3 Outline

In Chapter 2 of this thesis we will lay out necessary background information and will cover related work in the area of semantic segmentation. Each consecutive chapter will contain additional chapter-specific literature review.

In Chapter 3 we will start the discussion on real-time semantic image segmentation. The deep learning-based solutions for semantic segmentation has been constantly improving over the last years, largely due to the progress in other vision tasks. We will underline the challenges associated with building a real-time semantic segmentation network and lay out the motivation behind re-thinking an existing architecture such that redundant (non-needed) layers can be safely

omitted. In the context of next chapters, such an approach can be labelled as a *manual architecture search*.

In Chapter 4 we will consider a simple way of extending the single-task real-time network to support multiple dense per-pixel tasks at once. One of the main challenges in multi-task learning is the lack of joint annotations for all the tasks at once. To that end, we will cover a general solution that leads to stable training and superior results.

In Chapter 5 we will switch the focus to an automated architecture search of real-time networks suitable for semantic segmentation as well as other dense per-pixel tasks. Opposed to the manual approach guided by a human expert, in the automated search a secondary function, oftentimes another neural network, is tasked to predict a neural structure most suitable for the given task. While computationally expensive in the most general case, we will present the solution that achieve superior results given a limited number of GPU-machines.

Chapter 6 will continue where Chapter 5 left off: we will further overcome other limitations and restrictions of the automated neural architecture search and will discover extremely compact networks with less than 0.3M parameters.

2
Background

For clarity of our ideas and their presentation in later chapters, here we will provide necessary foundations and conventions used throughout the manuscript, as well as the overview of the related work.

2.1 Primer

First of all, all the algorithms discussed here operate on images, which we will represent as three-dimensional tensors with dimensions $C \times H \times W$, where C (stands for the *channel* dimension) equals 1 for grayscale and 3 for colour images, while H and W denote the height and width of the image, respectively. We will often stack several images together, in which case a new dimension, B (stands for *batch*), will be introduced: $B \times C \times H \times W$. A single operational element of an image X is called *pixel*; it will be a vector for colour images and a scalar for grayscale ones. We will usually denote the spatial coordinates of a pixel as i, j such that $X_{:,i,j}$ is the pixel value, where colon (:) denotes a subset operation through the channel dimension.

Each image (or a batch of images) will be processed by a function f with *trainable* (or *learnable*) *parameters* (or *weights*) θ, resulting in the output tensor Y. A simple neural network can be represented as some operation g between inputs X and weights θ. For example, g can be a dot product such that $g(X; \theta) := \theta^T X$.

It is easy to see that this new representation can be itself regarded as an input to another neural network with weights $\hat{\theta}$: $g(\hat{X};\hat{\theta}) = \hat{\theta}^T\hat{X} = \hat{\theta}^T g(X;\theta) = \hat{\theta}^T(\theta^T X)$. Since the dot product is a linear operation, we can replace $\hat{\theta}^T\theta^T$ with a single set of weights $\tilde{\theta}^T := \hat{\theta}^T\theta^T$ taking us back to the simple dot product of $\tilde{\theta}^T X$. As a mapping between inputs and outputs is rarely governed by a trivial linear relationship, an element-wise non-linear operation (which we denote as $h(\cdot)$) is usually applied in-between – also called *non-linearity* or *activation function*. Some popular activation functions include *sigmoid* ($\frac{1}{1+\exp(-x)}$), *hyperbolic tangent* ($tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$) and *rectified linear unit*, or *ReLU* ($max(0, x)$) (Glorot et al., 2011). Returning back to the example with weights $\hat{\theta}$ we can now write $g(\hat{X};\hat{\theta}) = \hat{\theta}^T h(\hat{X}) = \hat{\theta}^T h(g(X;\theta)) = \hat{\theta}^T h(\theta^T X)$. While the addition of an activation function facilitates the learning of non-trivial representations, it also leads to a more difficult and most often non-convex optimisation problem[1].

In the most general case, a feedforward neural network f is represented as a composition of various operations (also called *layers*) $f = l_N \circ l_{N-1} \ldots l_0$, where each layer l_i takes as input the output from the previous layer l_{i-1} (except for the first layer l_0 which takes X as input) and performs some operation on it (for example, a dot product or a non-linearity), perhaps with some parameters θ_i. We will refer to the outputs of the layers as *feature maps*, or *hidden representations*.

Since the computation of the dot product between the whole image and a multi-dimensional weight tensor is expensive, a commonly used substitute is *convolution* (or, to be more precise, *correlation*) that computes a dot product between regions in the input image and a set of weights (also called *filters* or *kernels*) of the appropriate size in a sliding window manner (see Fig. 2.1). Usually, these filters are shared across various spatial locations, making them *spatially invariant* and allowing them to detect presence of the same feature at various locations. All the kernels of a convolutional layer are usually arranged in a four-dimensional tensor of shape $C_{in} \times C_{out} \times k_h \times k_w$ where C_{in} represents the channel dimension of the input feature maps and C_{out} is the channel dimension of the output feature maps, while k_h and

[1] Note that ReLU almost behaves as identity, which is easy to optimise and to compute, making it a very popular choice.

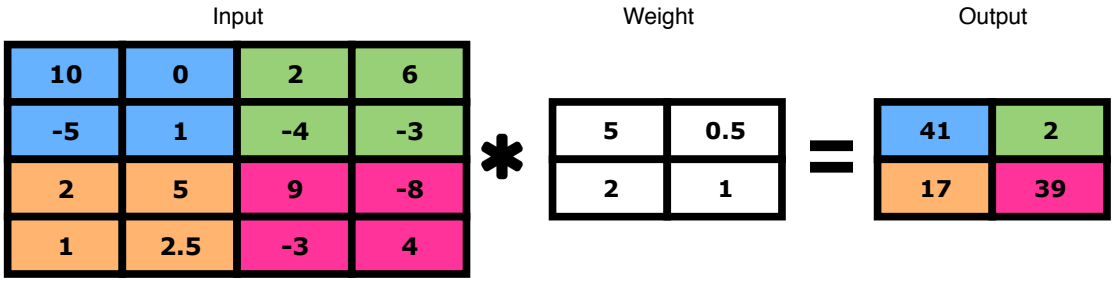


Figure 2.1: Convolution example on a 4×4 input using a single 2×2 kernel applied with the stride of 2. Colours in the input cells indicate that the dot product between the given region and the weight produces the output value in the corresponding cell of the same colour. In this example, the number 17 in the output is computed through the following sequence of multiplications and additions: $\mathbf{2} \cdot \mathbf{5} + \mathbf{5} \cdot \mathbf{0.5} + \mathbf{1} \cdot \mathbf{2} + \mathbf{2.5} \cdot \mathbf{1} = \mathbf{17}$.

k_w are height and width of the kernel, respectively. Often, we will omit channel dimensions and will simply denote a convolutional layer with its kernel dimensions $k_h \times k_w$ (*e.g.* 1×1 convolution would imply a convolution with $C_{in} \times C_{out}$ kernels of size 1×1). When spatially sliding the kernels, we can control the step size between each application of the kernel; this hyper-parameter is called *stride*. By increasing the stride of the convolution one can decrease the size of the spatial output. It is trivial to see that in the general case with an input of the spatial size of $H_{in} \times H_{in}$ and a $k \times k$ convolutional kernel, the output dimension is equal to $H_{out} = \frac{H_{in}-k}{s} + 1$.

Some other commonly used operations include *pooling* and *batch normalisation* (Ioffe and Szegedy, 2015). Pooling acts on image regions in the same way as convolution does, but instead of relying on a dot product uses a non-parametric operation – for example, by taking an average or a maximum of the region. Batch normalisation reparameterises feature maps such that they would follow a standard normal distribution, which tends to facilitate the optimisation process.

A neural network for which the output height and width depend on the spatial dimensions of the input, is called a *fully convolutional network* (or *FCN*) (Long et al., 2015). We will make an extensive use of a specific type of an FCN, called *encoder-decoder* (Noh et al., 2015) (see Fig. 2.2). This design implies a nominal division of the network into two parts – encoder and decoder. In the encoder the image undergoes a series of operations with some reducing the original spatial dimensions

by having strides larger than 1 (*i.e.* downsampling), while in the decoder the learned representations are being transformed up until the final prediction layer, often with some operations increasing the spatial dimensions (*i.e.* upsampling). The necessity of downsampling in the encoder is motivated by computational limits: if we were to remove it, we would end up with extremely large (both in spatial and channel dimensions) feature maps. In addition, that would further force us to increase either sizes of convolutional kernels or the number of layers so that each output pixel would still be a function of some other pixel at the same maximum distance (more formally, this distance is called the *receptive field size* of the network).[2] We will also rely on *skip-connections* (Long et al., 2015) that combine information from some encoder layers with some decoder layers of the same spatial size. It is believed (Long et al., 2015) that the decoder layers answer the question of what is present in the image, while the encoder layers answer the question of where it is present in the image, thus their combination answers both questions at the same time.

To find optimal weights θ for a given task, we will be minimising some loss function $\mathcal{L}$ on the training set, which is achieved by updating the network parameters using their gradient (*i.e.* a multi-dimensional derivative of the loss). We will rely on backpropagation (Rumelhart et al., 1986) and will carry out all the gradient computations with the help of automatic differentiation packages, such as PyTorch (Paszke et al., 2019). To define a loss function for semantic segmentation, we will follow the maximum likelihood principle,[3] which states that the optimal weights are the weights that maximise the product of probabilities of predicting the correct class at each pixel of each image from the training set, or, equivalently, that minimise the negative logarithm of the sum of these probabilities. More formally, for the training set of N images $\{X|X \in \mathbb{R}^{N \times C_{in} \times H \times W}\}$ and N corresponding annotations $\{Y|Y \in \overline{\{1, C_{out}\}}^{N \times H \times W}\}$ with C_{out} semantic classes, and a neural network f with weights θ, we can write:

[2] Larger receptive field size implies larger contextual coverage, which is important in cases when local information does not suffice to make a decision.

[3] Here we implicitly assume that all the training samples and their pixels are independent and identically distributed.

$$\theta_{optimal} = \arg\max_{\theta} \prod_{i,j,k} f(X;\theta)_{i,Y_{i,j,k},j,k} = -\arg\min_{\theta} \sum_{i,j,k} \log f(X;\theta)_{i,Y_{i,j,k},j,k} \quad (2.1)$$

Here we assume that the outputs of the network are probabilities; the common way to make it explicit is by applying an activation function called *softmax* on the outputs, which for the input vector $\vec{z} \in \mathbb{R}^C$ is formulated as follows:

$$softmax(\vec{z}) = (\frac{\exp(z_1)}{\sum_{i=1}^{C}\exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{C}\exp(z_i)}, \ldots, \frac{\exp(z_C)}{\sum_{i=1}^{C}\exp(z_i)})^T \quad (2.2)$$

It is easy to verify that all the values in the vector $softmax(\vec{z})$ lie between 0 and 1, while their sum is equal to 1, making it a valid probability distribution.

Since semantic segmentation is a (multi-label) classification task, conventional classification metrics can be applied to it, too. Formally, for semantic segmentation with C classes, we define *accuracy* of a single class i as the number of *true positives* (n_{ii} – *i.e.* the number of pixels belonging to class i and predicted as such) divided by the sum of true positives and *false positives* ($\sum_{j\neq i} n_{ij}$ – *i.e.* the number of pixels belonging to class i but predicted as class $j \neq i$); or, in other words, the number of true positives divided by the total number of pixels belonging to class i, computed as $\frac{n_{ii}}{n_{ii}+\sum_{j\neq i} n_{ij}}$. While this gives us a vector, it is common to take an average of its values and report a scalar called *mean per-class accuracy*. Since mean per-class accuracy does not penalise for having *false negatives* ($\sum_{j\neq i} n_{ji}$ – *i.e.* the number of pixels not belonging to class i but predicted as such), another metric called *intersection-over-union*, or *Jaccard Index* (Everingham et al., 2010), is usually measured: $\frac{n_{ii}}{n_{ii}+\sum_{j\neq i} n_{ij}+\sum_{j\neq i} n_{ji}}$. Once again, the average across the classes gives rise to a quantity named *mean intersection-over-union*, or *mean iou*, which we will rely upon extensively throughout the manuscript.

In addition to that, we will often report network-specific characteristics, such as the number of floating point operations (or *FLOPs*), or equivalently – the number of multiplications and additions (or *MAdds*). These numbers can serve as a rough estimate of the latency of the network – often implying that the network with fewer floating point operations is faster.

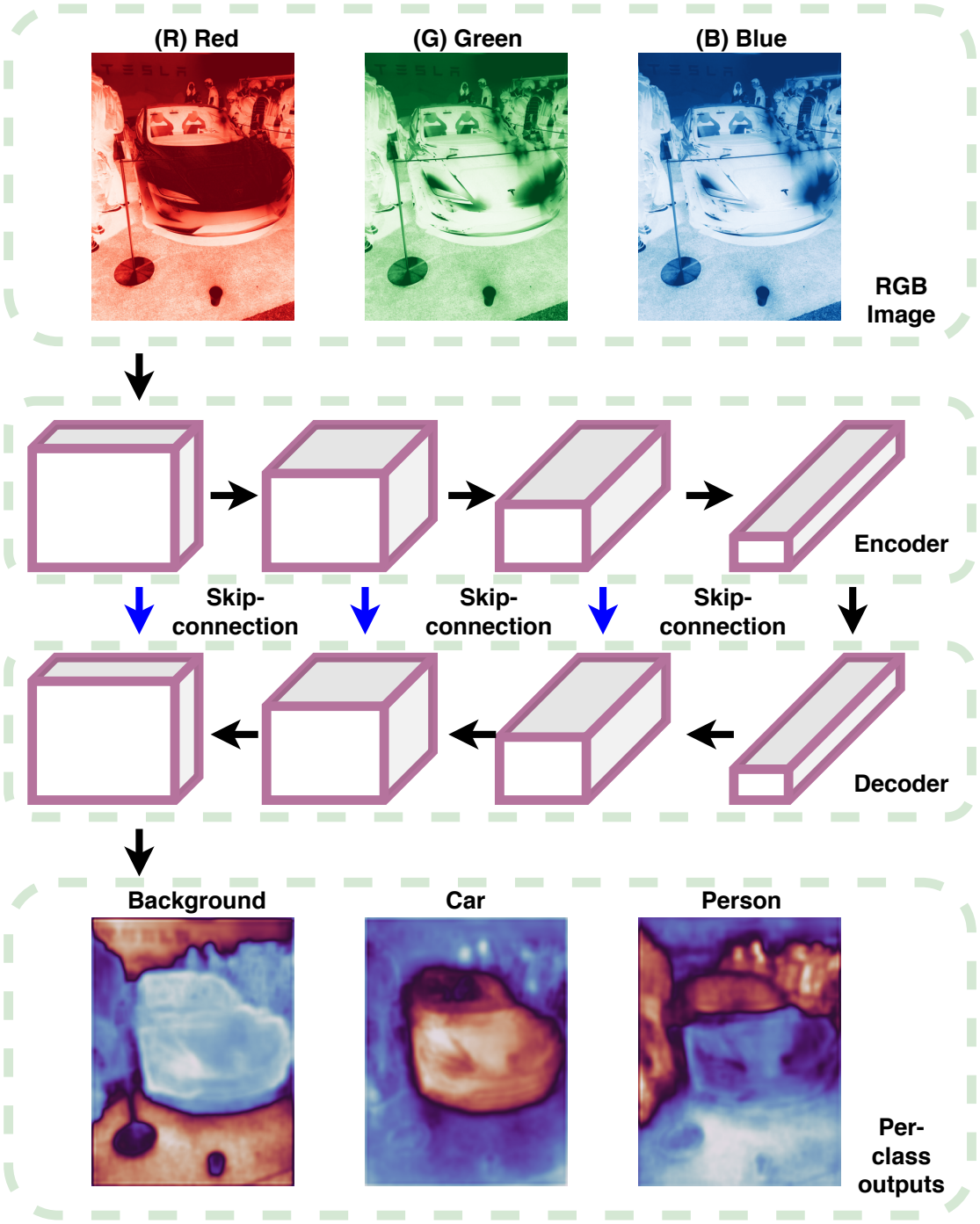


Figure 2.2: A structural depiction of the encoder-decoder segmentation network. Feature maps in convolutional neural networks are commonly presented as parallelepipeds, with height and width denoting the spatial resolution, and depth denoting the number of channels. The network takes as input a 3-channel RGB image which is first processed by multiple convolutional layers in the encoder part that reduce the spatial size of the feature maps, while increasing the number of channels. The encoder output is then fed into the decoder part, where the spatial size is being recovered. Skip-connections (*in blue*) connect intermediate feature maps between the encoder and the decoder parts. The final output is of the same spatial size as the input and the depth being equal to the number of predicted classes (here, three – *background, car* and *person*). Red colour in the outputs implies a higher probability of the given class at the corresponding location, whereas blue indicate a low probability.

2.2 Related Work

Whereas each consecutive chapter comes with its own specific survey on the relevant work, here we will provide a general overview of semantic segmentation and place some of the thesis contributions in the wider context.

2.2.1 Semantic Segmentation

Semantic image segmentation is a dense per-pixel labelling task, the goal of which is to assign a semantic class out of a pre-defined set to each pixel in the input image, *e.g.* body parts such as *head* / *torso*, or general categories – *cat* / *dog* / *person* / *car*. Early approaches relied on handcrafted features such as histogram of oriented gradients, or HOG (Dalal and Triggs, 2005) and scale-invariant feature transform, or SIFT, descriptors (Lowe, 2004), in combination with some popular classifiers (such as SVMs and decision trees) (Shotton et al., 2006; Csurka and Perronnin, 2008; Shotton et al., 2008; Fulkerson et al., 2009) and hierarchical graphical models (Ladicky et al., 2009; Plath et al., 2009; Krähenbühl and Koltun, 2011). This manual feature construction lasted until the resurgence of deep learning when a neural network-based approach led by Krizhevsky et al., 2012 famously won the ImageNet competition (Deng et al., 2009).

Ciresan et al., 2012 first proposed to use a neural network to segment cell membranes in a sliding window manner by classifying the central pixel of each window, while Pinheiro and Collobert, 2014 increased the contextual coverage by refining the output of the network recurrently. At the same time Farabet et al., 2013 combined multi-scale convolutional features, superpixels (Felzenszwalb and Huttenlocher, 2004) and a conditional random field (CRF), while still employing a patch-based training. Long et al., 2015 were the first to propose a complete fully-convolutional approach for the task of semantic segmentation. This was done by changing fully-connected (or linear) layers of existing classification networks into convolutional ones with 1×1 filters. Besides the general architecture that became known as *FCN* (standing for fully-convolutional network) Long et al., 2015 also proposed *skip-connections* between feature maps from earlier to later layers. As

earlier layers in convolutional networks tend to act as edge detectors with later layers learning more class-specific details (Zeiler and Fergus, 2014), such a fusion of information between low-level and high-level feature maps led to a more accurate delineation of contours of segmentation objects.

The simplicity of that approach spurred more research in the field; in particular, L. Chen et al., 2015 were responsible for two important modifications – dilated convolutions and post-processing with CRFs using convolutional features (opposed to Farabet et al., 2013 who built the CRF with superpixels). The first modification was proposed to reduce the effects of multiple downsampling layers present in image classification networks; concretely, L. Chen et al., 2015 removed the last two max-pooling layers in the fully-convolutional VGG network (Simonyan and Zisserman, 2015). In order to compensate for the shrinkage in the contextual coverage, they used so-called atrous or dilated convolutions (see Fig. 2.3) (Holschneider et al., 1990) that kept the field of view of the network intact. The second modification was a direct adaptation of the fully-connected CRF model proposed few years earlier by Krähenbühl and Koltun, 2011, where the outputs of the convolutional networks were used in-place of unary potentials. This work later became known as *DeepLab-v1*. As the removal of downsampling operations leads to an increase in the spatial size of feature maps, the memory footprint and the latency of networks with dilated convolutions tend to be significantly higher than those with the standard setup.

Noh et al., 2015 paired a fully-convolutional network with a structural copy giving a name to the *encoder-decoder* approach. They also exploited un-pooling layers in the decoder part that negated the pooling in the first part (encoder) by propagating pooling indices. At the same time, due to the success of post-processing with CRF in DeepLab, several authors built other combinations of probabilistic graphical models with neural networks. In particular, G. Lin et al., 2016 relied on the piecewise training (C. A. Sutton and McCallum, 2005) of the CRF model, while Z. Liu et al., 2015 turned to the mean-field approximation (Opper, Winther, et al., 2001). Zheng et al., 2015 proposed to recast CRF updates as a recurrent neural network (RNN), allowing to train the full system end-to-end in a simple way.

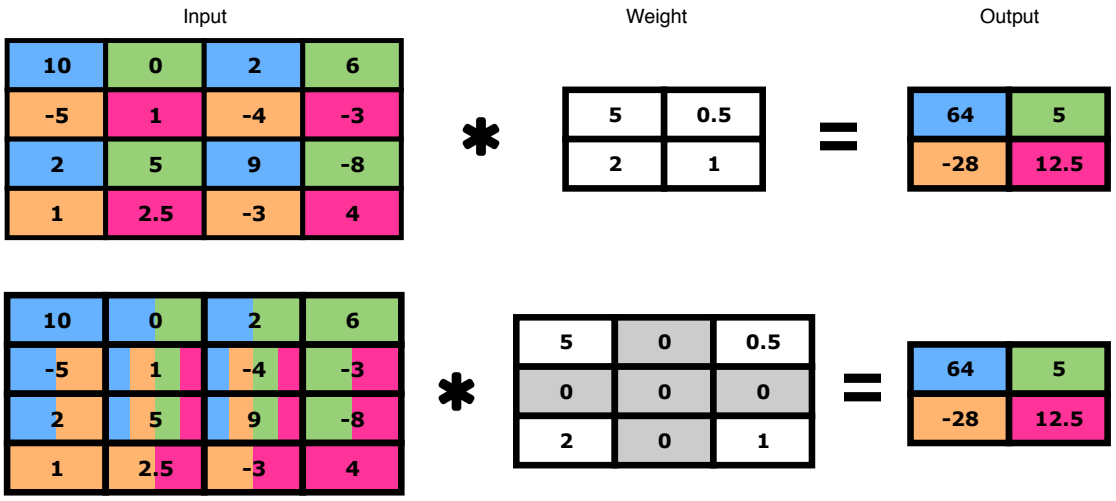


Figure 2.3: (*top*) An example of dilated convolution with the dilation rate of 2 on a 4×4 input using a single 2×2 kernel applied with the stride of 1. (*bottom*) The dilated convolution is equivalent to the standard convolution with zeros inserted between the values of the kernel (*in gray colour*), making it of size 3×3. Colours in the input cells indicate that the dot product between the given region and the weight produces output value in the corresponding cell of the same colour. Several colours in the same cell are used to denote that the value is being used multiple times to compute different outputs. In the bottom example, the number **12.5** in the output is computed through the following sequence of multiplications and additions: $(\mathbf{1} \cdot \mathbf{5} + \mathbf{-4} \cdot \mathbf{0} + \mathbf{-3} \cdot \mathbf{0.5}) + (\mathbf{5} \cdot \mathbf{0} + \mathbf{9} \cdot \mathbf{0} + \mathbf{-8} \cdot \mathbf{0}) + (\mathbf{2.5} \cdot \mathbf{2} + \mathbf{-3} \cdot \mathbf{0} + \mathbf{4} \cdot \mathbf{1}) = \mathbf{12.5}$.

The success of residual networks (He et al., 2016) also led to significant improvements on semantic segmentation benchmarks with the most popular being PASCAL VOC (Everingham et al., 2010). With additional annotations from Hariharan et al., 2011 it contained roughly 10000 training images with the maximum resolution of 500×500. Later, Cordts et al., 2016 collected a novel dataset with nearly 3000 high-resolution (2048×1024) training images of European urban street scenes using a camera mounted on a car. Largely due to those advances, a significant research went into exploring ways of capturing contextual information without any reliance on computationally expensive graphical models. With that regard, DeepLab-v2 by L. Chen et al., 2018b used an atrous spatial pyramid pooling layer (*ASPP*) that contained several dilated convolutions arranged in parallel in order to better capture various features at different scales. Instead of relying on dilated convolutions in the contextual block, Zhao et al., 2017 built a variation of pyramid pooling (He et al., 2014) with several spatial resolutions, before aggregating information from all the

scales using concatenation. In yet another line of work, which we will cover in detail in next chapters, G. Lin et al., 2017 enhanced the encoder-decoder network with multiple residual blocks in the decoder part, including a pooling-based contextual structure called *chained residual pooling* (CRP).

More recently, L. Chen et al., 2017 (DeepLab-v3) further improved their approach by experimenting with multiple design choices related to the usage of atrous convolution: as the result, the authors added a global pooling layer inside ASPP in order to better capture global context. In another follow-up, DeepLab-v3+ (L. Chen et al., 2018c) was proposed with a skip-connection from a lower layer and the inclusion of depthwise separable convolutions to reduce computational costs (Sifre and Mallat, 2014; Chollet, 2017).

The latest advances in semantic segmentation has been achieved by neural architecture search: in particular, Auto-DeepLab (C. Liu et al., 2019) employed a stochastic differentiation method (H. Liu et al., 2019), while L. Chen et al., 2018a searched for a single set of operations (dense predictive cell, or DPC) using 400 GPUs for 7 days. In Chapter 5 we will discuss a way of efficiently searching for real-time semantic segmentation networks, which was developed in parallel to DPC and requires only 8 GPU-days.

2.2.2 Real-Time Semantic Segmentation

Advances in general semantic segmentation also spurred a significant progress in building practical semantic segmentation networks.

SegNet proposed by Badrinarayanan et al., 2015 comprised two copies of the VGG-16 net (Simonyan and Zisserman, 2015) for the encoder and the decoder, and explicitly preserved pooling indices from the encoder for their further reuse in the decoder (analogously to the work by Noh et al., 2015). They also abandoned computationally expensive layers of VGG and added batch normalisation layers (Ioffe and Szegedy, 2015) to stabilise the training. While this led to a faster runtime, the structure of both the encoder and the decoder were identical and the authors did not explore ways of designing each part separately.

ENet (Paszke et al., 2016) instead used custom encoder and decoder with factorised residual blocks, where 5×5 2D convolution was split into 2 separate 1D convolutions (1×5 and 5×1, respectively). Neither of those methods were close to state-of-the-art results achieved by larger models as they primarily prioritised the runtime.

Hence other works were directly adapting existing larger networks: e.g., ICNet (Zhao et al., 2018), where the authors relied on PSPNet (Zhao et al., 2017) to deal with multiple image scales progressively. Specifically, they used 3 scales of the input image, and applied the smallest number of layers to the input with the highest resolution. It must be noted that such a division into scales is only effective for large inputs, limiting the range of applications of ICNet.

Li et al., 2017 proposed a cascading approach where in each progressive level of the cascade only a certain portion of pixels was processed. The choice of which pixels to propagate was done explicitly by setting a hard threshold on intermediate classifier outputs.

Notably, their most accurate approach (80.3% mean iou on PASCAL VOC (Everingham et al., 2010)) was also the slowest with the runtime of only 1 FPS. In contrast, in Chapter 3 we will talk about Light-Weight RefineNet (Nekrasov et al., 2018b) that was proposed at the same time with those works, and was able to reach nearly 83% mean iou on PASCAL VOC running at more than 30 FPS, as well as being close to state-of-the-art results on several other benchmarks.

Akin to ICNet, some recent works on real-time semantic segmentation largely concentrated on high-resolution images from CityScapes (Cordts et al., 2016) and ways of making models more compact. In particular, BiSeNet (C. Yu et al., 2018) contained two paths – one that kept the spatial information intact, and one contextual with an aggressive downsampling strategy. Two paths were merged by a specifically designed fusion module with per-channel attention. In contrast, in ESPNet (Mehta et al., 2018) the authors replaced a convolutional layer with a hierarchical pyramid of point-wise and dilated convolutions. Later, Mehta et al., 2019 made ESPNet even more efficient by exploiting separable convolutions (Sifre and Mallat, 2014; Chollet, 2017). Importantly, as we will see in Chapter 6, neural

architecture search can discover competitive architectures on that benchmark with less than 0.5M parameters underlining that the number of parameters needed for a decent performance on this dataset can a priori be quite low.

3

Real-Time Semantic Segmentation

In this chapter we will consider an important task of effective and efficient semantic image segmentation. While significant strides have been made towards improving the accuracy of semantic segmentation models, these advances has come at the expense of latency. Mitigating this trade-off is non-trivial but essential for creating practical vision systems as introduced in Chapter 1.

To this end, here we will propose a manual way of adapting a powerful yet slow segmentation approach of RefineNet (G. Lin et al., 2017) into the more compact and faster one. In particular, we will identify computationally expensive blocks in the original setup and come up with two modifications aimed to decrease the number of parameters and floating point operations without any significant performance degradation. We will demonstrate that it is possible to build a faster version of RefineNet through a careful manual re-design of the underlying architecture. This simpler version will also allow us to gain a better understanding of the importance of various components used in the encoder-decoder semantic segmentation models.

The results of this chapter first appeared in the proceedings of the 29*th British Machine Vision Conference, (BMVC* 2018*), under the title of 'Light-Weight RefineNet for Real-Time Semantic Segmentation' (Nekrasov et al., 2018b).*[1]

[1]The trained models and the training code have been released and are available here: https://github.com/DrSleep/light-weight-refinenet

3.1 Related Work

Perhaps, the most straightforward path towards a real-time network is by manual design from scratch: for example, such compact classification networks as MobileNet (Howard et al., 2017; Sandler et al., 2018), ShuffleNet (X. Zhang et al., 2018; Ma et al., 2018), SqueezeNet (Iandola et al., 2016) were carefully crafted out of groupwise and 1×1 convolutions (which, when applied sequentially, leads to a separable convolution (Sifre and Mallat, 2014; Chollet, 2017)). Recent works in semantic segmentation has followed the same principles (Mehta et al., 2018; Mehta et al., 2019; Romera et al., 2018), but, opposed to the classification counterparts, are lagging far behind state-of-the-art results established by larger models – for example, by up to 20% in mean iou on CityScapes (Cordts et al., 2016).

The next logical approach is to manually adapt the design of an already existing, but larger network. This is the direction we consider in this chapter. Prior to our contribution, Zhao et al., 2018 proposed a real-time variation of PSPNet (Zhao et al., 2017) varying the number of layers needed for different image resolutions. With the latency being reduced, this also led to a drop in performance by more than 10% in mean iou on CityScapes in comparison to PSPNet. In contrast, our methodology achieves results on par with those of the original RefineNet (G. Lin et al., 2017) by carefully selecting unnecessary layers.

More recently, due to the advances in the neural architecture search (Zoph and Le, 2017), compact and real-time designs can be found automatically: in particular, in Chapter 5 we will do so by restricting the search space. On smaller domains, such as image classification, Tan et al., 2019; B. Wu et al., 2019 demonstrated that an addition of a latency-specific objective (*e.g.* as measured on mobile phones) helps in discovering tiny yet accurate networks.

While so far we have only discussed ways of altering the design of a network before the training even starts, several solutions exist for compressing an existing network – for example, in pruning (Han et al., 2015b; LeCun et al., 1989; Hassibi and Stork, 1992; Han et al., 2015a; Molchanov et al., 2019) some connections (or even whole layers) are being removed within the trained network, while in

quantisation (Gong et al., 2014; S. Zhou et al., 2016; A. Zhou et al., 2017; Jacob et al., 2018; K. Wang et al., 2019) either the network weights or activations (or even both) are being converted into a lower-precision format that reduces the memory footprint and can further lead to a faster inference runtime on certain hardware setups. Yet another approach involves factorisation of existing layers into lower-dimensional ones (Denton et al., 2014; Jaderberg et al., 2014). Here it is important to emphasise that all these methods are often-times task-agnostic and orthogonal to existing design solutions, which makes them applicable to a variety of cases.

Finally, we must mention knowledge distillation (Ba and Caruana, 2014; Hinton et al., 2014; Bucila et al., 2006; Romero et al., 2015) as a sensible way of acquiring a compact, high-performing model. Its main idea is to force a smaller network (oftentimes, called *student*) to mimic the outputs of a larger existing network (or *teacher*). We will see the application of knowledge distillation in the next chapter.

3.2 RefineNet Primer

The RefineNet architecture (G. Lin et al., 2017) belongs to the family of the encoder-decoder approaches (Noh et al., 2015). As the encoder backbone, it can re-use most popular classification networks; in particular, the original contribution was showcased using residual networks (He et al., 2016). The approach does not amend an underlying classification network in any way except for omitting last classification layers.

We have chosen this particular structure as it i.) shows the best performance among other encoder-decoder networks, and ii.) does not operate over large feature maps in the last layers as methods exploiting atrous convolution do (L. Chen et al., 2015; L. Chen et al., 2018b; L. Chen et al., 2017; Zhao et al., 2017; F. Yu et al., 2017). The processing of large feature maps significantly hinders real-time performance of such architectures as DeepLab (L. Chen et al., 2015; L. Chen et al., 2018b; L. Chen et al., 2017) and PSPNet (Zhao et al., 2017) due to the increase in the number of floating point operations. As a weak point, the *encoder-decoder* approaches tend to have a larger number of parameters due to the decoder part

that recovers the high resolution output. In this chapter, we tackle the real-time performance problem by specifically concentrating on the decoder and will show empirically that both the number of parameters and floating point operations can be drastically reduced without a significant drop in accuracy.

In the decoder part, RefineNet relies on two types of additional abstractions: residual convolutional unit (RCU) (Fig. 3.1(b)) and chained residual pooling (CRP) (Fig. 3.1(c)). The first one is a simplification of the original residual block (He et al., 2016) without batch normalisation (Ioffe and Szegedy, 2015), while the second one is a sequence of multiple convolutional and pooling layers, also arranged in a residual manner; as noted by G. Lin et al. (2017), the abundance of residual connections facilitates the optimisation of RefineNet. All of these blocks use 3×3 convolutions and 5×5 pooling with an appropriate padding so that the spatial size would stay intact.

The decoding process starts with the propagation of the last output from the classifier (with the lowest resolution) through two RCU blocks followed by four pooling blocks of CRP and another three RCU blocks before an application of a fusion block (Fig. 3.1(d)) that merges these features with the features of the second to last output. Inside the fusion block, each path is convolved with 3×3 convolution and upsampled to the largest resolution among the paths. Two paths are then summed up, and are analogously propagated further through several RCU, CRP and fusion blocks until the desired resolution is reached. The final convolutional layer produces the score map.

The authors motivate the usage of the CRP block as a way of increasing the contextual coverage, and they demonstrate its value through a set of ablation experiments. The role of the RCU block, on the other hand, is left unclear. In contrast, we not only amend the design choices for both blocks (Sect. 3.3.1), but also measure the value of the RCU block (Sect. 3.3.2) and provide an in-depth analysis of the importance of these two components from the perspective of contextual coverage (Sect. 3.5.1) and feature learning (Sect. 3.5.2).

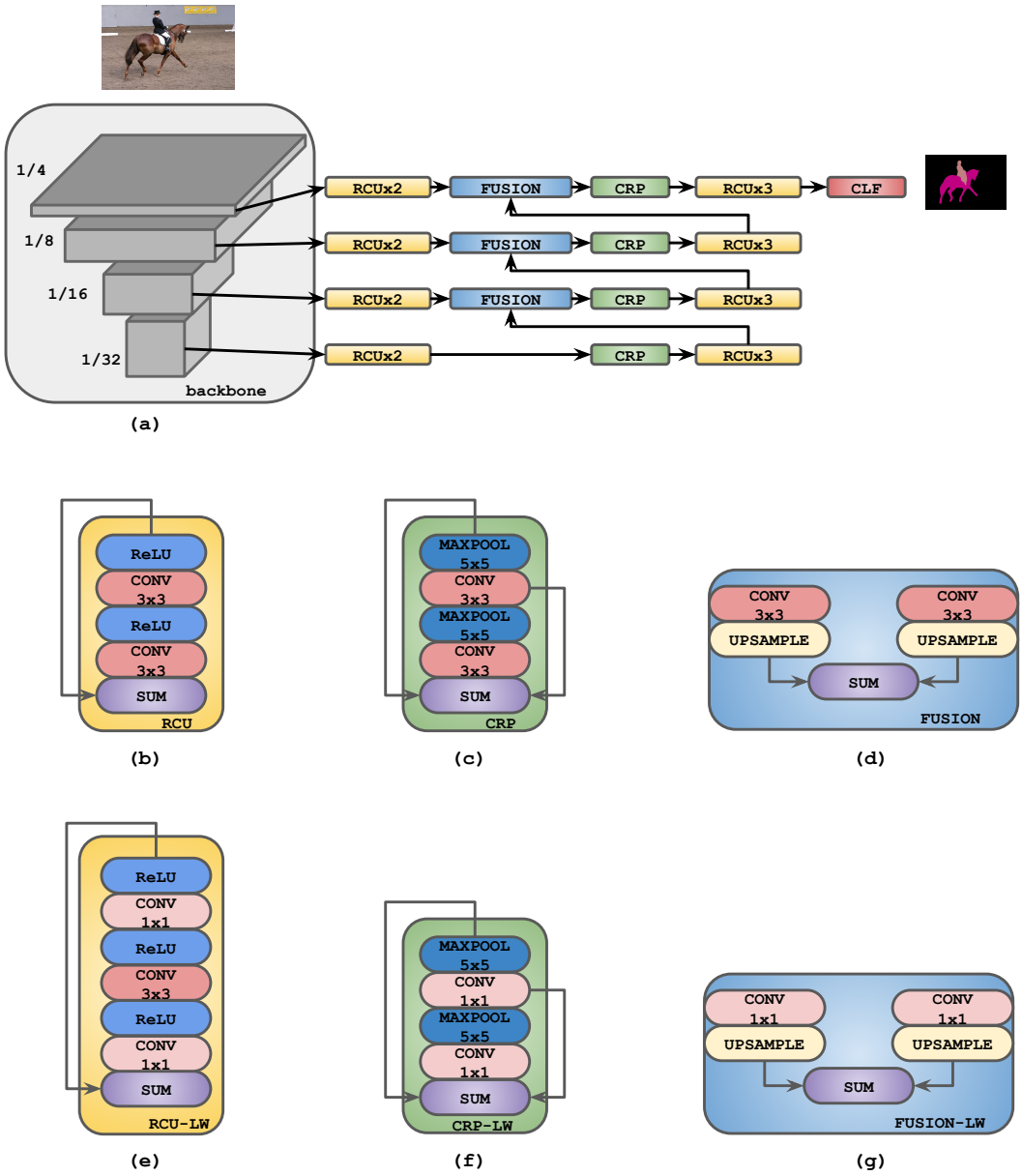


Figure 3.1: RefineNet structure. (a) General network architecture with RefineNet for semantic segmentation, where *CLF* stands for a single 3x3 convolutional layer with the number of channels being equal to the number of output classes; (b)-(d) general outline of original RCU, CRP and fusion blocks; (e)-(g) light-weight RCU, CRP and fusion blocks. In the interests of brevity, we only visualise 2 convolutional layers for the CRP blocks (instead of 4 used in the original architecture). Note that we do not use any RCU blocks in our final architecture as discussed in Sec. 3.3.2.

Model	Parameters,M	FLOPs,B
RefineNet-101 (G. Lin et al., 2017)	118	263
RefineNet-101-LW-with-RCU	54	76
RefineNet-101-LW	**46**	**52**

Table 3.1: Comparison of the number of parameters and floating point operations on 512×512 inputs between original RefineNet, Light-Weight RefineNet with RCU (**LW-with-RCU**), and Light-Weight RefineNet without RCU (**LW**). All the networks exploit ResNet-101 as backbone.

3.3 Light-Weight RefineNet

3.3.1 Replacing 3x3 convolutions

We start by highlighting that 3×3 convolutions are the most expensive parts in the original RefineNet in terms of both the number of parameters and the number of floating point operations. Thus, we focus on replacing them with simpler counterparts without a performance drop.

Intuitively, convolutions with larger kernel sizes aim to increase the receptive field size (and the global context coverage), while 1×1 convolutions are merely transforming per-pixel features from one space to another locally. We argue that in the case of RefineNet we do not require expensive 3×3 convolutions in the decoder part at all, and we are able to show empirically that replacing them with 1×1 convolutions does not hurt performance. To further establish our claims, we evaluate the empirical receptive field size (B. Zhou et al., 2015) in the original and modified RefineNet, and do not find any significant difference (Section 3.5.1). In particular, we replace 3×3 convolutions within CRP and fusion blocks with 1×1 counterpart (Fig. 3.1(f-g)), and we amend RCU into the one with the bottleneck design (He et al., 2016) (Fig. 3.1(e)). This way, we reduce the number of parameters by more than 2× times, and the number of FLOPs by more than 3× times (Table 3.1).

3.3.2 Omitting RCU blocks

We proceeded with initial experiments and trained the light-weight architecture outlined in the previous section using PASCAL VOC (Everingham et al., 2010). We were able to achieve close to the original network performance, and, furthermore, observed that removing RCU blocks did not lead to any accuracy deterioration – in fact, the weights in RCU blocks almost completely saturated.

To confirm that this only occurs in the light-weight case, we omitted RCU blocks in the original RefineNet architecture – this led to more than 5% performance drop on the same dataset. We argue that this happens due to the RCU blocks being redundant in the 1×1 convolution regime, as the only important goal of increasing the contextual coverage is essentially performed by pooling layers inside CRP. To back up this claim, we conduct a series of ablation experiments which are covered later in Section 3.5.2.

Our final architecture does not contain any RCU blocks and only relies on CRP blocks with 1×1 convolutions and 5×5 max-pooling inside, which makes our method extremely fast and light-weight.

3.3.3 Adaptation to different backbones

A significant practical benefit of the RefineNet architecture is that it can be mixed with any backbone network as long as the backbone has several subsampling operations inside (which is the case for most SOTA classifiers). Thus, there are no specific changes needed to be made in order to apply the RefineNet architecture using any other model. In particular, in the experiments section, we showcase its adaptation using efficient NASNet-Mobile (Zoph et al., 2018) and MobileNet-v2 (Sandler et al., 2018) networks; we still achieve solid performance with the limited number of parameters and floating point operations.

3.4 Experiments

In the experimental part, our aims are to prove empirically that we are able i.) to achieve similar performance levels with the original RefineNet while ii.)

significantly reducing the number of parameters and iii.) drastically increasing the speed of a forward pass; and iv.) to highlight the possibility of applying our method using other architectures.

To this end, we consider five segmentation datasets: namely, NYUDv2 (Silberman et al., 2012; Gupta et al., 2013), PASCAL VOC (Everingham et al., 2010), PASCAL Person-Part (X. Chen et al., 2014; L. Chen et al., 2016), PASCAL Context (Mottaghi et al., 2014) and CityScapes (Cordts et al., 2016), and five classification networks: i.e., ResNet-50, ResNet-101, ResNet-152 (He et al., 2016), NASNet-Mobile (Zoph et al., 2018) and MobileNet-v2 (Sandler et al., 2018) (only for Pascal VOC), all of which have been pre-trained on ImageNet (Deng et al., 2009). As a general practice, we report mean intersection over union (Everingham et al., 2010) on each benchmark.

We perform all our experiments in PyTorch (Paszke et al., 2019), and train using stochastic gradient descent with momentum. For all residual networks we start with the initial learning rate of 5e-4, for NASNet-mobile and MobileNet-v2 we start with the learning rate of 1e-3. We keep batch norm statistics frozen during the training.

For benchmarking, we use a workstation with 8GB RAM, Intel i5-7600 processor, and one GT1080Ti GPU card. For a fair comparison of runtime, we re-implement the original RefineNet in PyTorch.[2] We compute 100 forward passes with random inputs, and average the results; when reporting, we provide both the mean and standard deviation values.

We provide qualitative results for PASCAL VOC in the main text and for the rest of datasets in Appendix A.

3.4.1 NYUDv2

We first conduct a series of initial experiments on the NYUDv2 dataset (Silberman et al., 2012; Gupta et al., 2013). This dataset comprises 1449 RGB-D images with 40 segmentation classes, of which 795 are used for training and 654 for testing, respectively. We do not make use of depth information in any way. We reduce

[2]The re-implementation of RefineNet is available here: https://github.com/DrSleep/refinenet-pytorch

Model	NYUD mIoU,%	Person mIoU,%	Params,M	Runtime,ms
FCN16-s RGB-HHA (Long et al., 2015)	34.0	-	-	-
Context (G. Lin et al., 2016)	40.6	-	-	-
DeepLab-v2-CRF (L. Chen et al., 2018b)	-	64.9 (*msc*)	**44**	-
RefineNet-50 (G. Lin et al., 2017)	42.5	65.7	99	54.18 ± 0.46
RefineNet-101 (G. Lin et al., 2017)	43.6	67.6	118	60.25 ± 0.53
RefineNet-152 (G. Lin et al., 2017)	**46.5** (*msc*)	**68.8** (*msc*)	134	69.37 ± 0.78B
RefineNet-LW-50 (ours)	41.7	64.9	**27**	**19.56** ± 0.29
RefineNet-LW-101 (ours)	43.6	66.7	**46**	**27.16** ± 0.19
RefineNet-LW-152 (ours)	44.4	67.6	**62**	**35.82** ± 0.23

Table 3.2: Quantitative results on the test sets of NYUDv2 and PASCAL Person-Part. Mean iou, the number of parameters and the runtime (mean±std) of one forward pass on 625×468 inputs are reported, where possible. Multi-scale evaluation is defined as *msc*.

the learning rate by half after 100 and 200 epochs, and keep training until 300 epochs, or until an earlier convergence.

Our quantitative results are provided in Table 3.2 along with the results from the original RefineNet, and other competitive methods on this dataset. We note that for all ResNet networks we are able to closely match the performance of the original RefineNet, while having only a slight portion of the original parameters – in particular, the smallest RefineNet, *i.e.* RefineNet-50, achieves 42.5% mean iou with 99M parameters, while the largest Light-Weight RefineNet, *i.e.* Light-Weight RefineNet-152, shows 44.4% mean iou with 62M parameters. In fact, the largest Light-Weight RefineNet model is 1.5× faster than the smallest RefineNet model – with nearly 30FPS for RefineNet-LW-152 against nearly 20FPS for RefineNet-50 on 625×468 inputs.

3.4.2 Person-Part

PASCAL Person-Part dataset (X. Chen et al., 2014; L. Chen et al., 2016) consists of 1716 training and 1817 validation images with 6 semantic classes (head, torso, and upper/lower legs/arms) and a single background class. We follow the same training strategy as for NYUDv2, and provide our results in Table 3.2. Again, we achieve analogous to the original models results with the Light-Weight RefineNet models showing mean iou values of 64.9%, 66.7%, 67.6%, respectively, against 65.7%, 67.6% and 68.8% of those by the original RefineNet networks.

3.4.3 PASCAL VOC

We continue our experiments with a standard benchmark dataset for semantic segmentation, PASCAL VOC (Everingham et al., 2010). This dataset consists of 4369 images with 20 semantic classes plus background, of which 1464 constitute the training set, 1449 – validation, and 1456 – test, respectively. As commonly done, we augment it with additionally annotated VOC images from BSD (Hariharan et al., 2011), as well as images from MS COCO (T. Lin et al., 2014).

We train all the networks using the same strategy, but with different learning rates as outlined above: in particular, we reduce the learning rate by half after 20 epochs of COCO training and after 50 epochs of the BSD training, and keep training until 200 total epochs, or until the accuracy on the validation set stops improving.

Our quantitative results on validation and test sets are given in Table 3.3 along with the results from the original RefineNet. We again notice that for all ResNet networks, we achieve performance on-par with the original RefineNet, while having significantly fewer FLOPs – *e.g.* 71B for Light-Weight RefineNet-152 against 263B for RefineNet-101; for NASNet-Mobile and MobileNet-v2, we outperform both MobileNet-v1+DeepLab-v3 and MobileNet-v2+DeepLab-v3 approaches (Sandler et al., 2018), which are closely related to our method. Qualitative results are provided in Figure 3.2 – note how all Light-Weight RefineNet with the ResNet-backbone models are able to segment a small bottle in the bottom row and nearly correctly segment dogs in rows 2 and 3; also note how both networks with the light-weight backbones (NAS and MOB) seem to slightly struggle with thin bicycle structures in the first row.

3.4.4 PASCAL-Context

This dataset comprises 60 semantic labels (including background), and consists of 4998 training images, and 5105 validation images. During training, we divide the learning rate by half twice after 50 epochs and after 100 epochs, respectively, and keep training until 200 epochs, or an earlier convergence. We do not pre-train on PASCAL VOC or COCO.

Model	val mIoU,%	test mIoU,%	FLOPS,B
DeepLab-v2-ResNet-101-CRF (L. Chen et al., 2018b)	77.7	79.7	-
RefineNet-101 (G. Lin et al., 2017)	-	82.4	263
RefineNet-152 (G. Lin et al., 2017)	-	**83.4**	283
RefineNet-LW-50 (ours)	78.5	81.1	**33**
RefineNet-LW-101 (ours)	80.3	82.0	52
RefineNet-LW-152 (ours)	82.1	82.7	71
MobileNet-v1-DeepLab-v3 (Sandler et al., 2018)	75.3	-	14.2
MobileNet-v2-DeepLab-v3 (Sandler et al., 2018)	75.7	-	**5.8**
RefineNet-LW-MobileNet-v2 (ours)	**76.2**	**79.2**	**9.3**
RefineNet-LW-NASNet-Mobile (ours)	**77.4**	**79.3**	11.4

Table 3.3: Quantitative results on PASCAL VOC. Mean iou and the number of FLOPs on 512 × 512 inputs are reported, where possible.

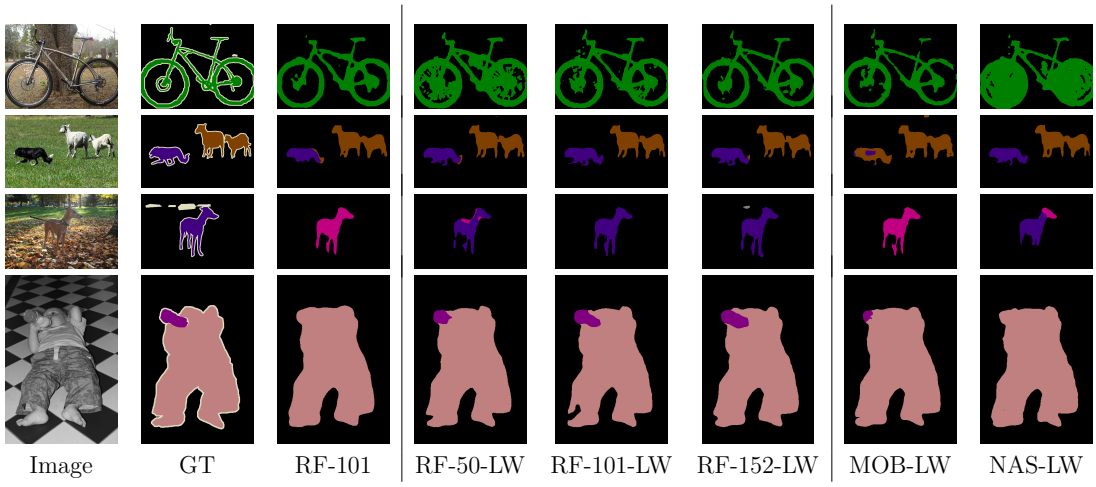


Figure 3.2: Visual results on validation set of PASCAL VOC with residual models (RF), MobileNet-v2 (MOB) and NASNet-Mobile (NAS). The original RefineNet-101 (RF-101) is our re-implementation. 'GT' stands for 'ground truth'.

Our quantitative results are provided in Table 3.4 – single-scale evaluation of Light-Weight RefineNet-101 and RefineNet-152 achieves 45.1% and 45.8% mean iou against 47.1% and 47.3% of RefineNet-101 and RefineNet-152 evaluated with multiple scales.

3.4.5 CityScapes

Finally, we turn our attention to the CityScapes dataset (Cordts et al., 2016), that contains 5000 high-resolution (1024 × 2048) images with 19 semantic classes, of which 2975 images are used for training, 500 for validation, and 1525 for testing, respectively. We use the same learning strategy as for PASCAL-Context. Our

single-scale model with ResNet-101 as backbone is able to achieve 72.1% mean iou on the test set, which is close to the original RefineNet result of 73.6% with multi-scale evaluation (G. Lin et al., 2017).

Model	mIoU,%
DeepLab-v2-CRF (L. Chen et al., 2018b)	45.7 (*msc*)
RefineNet-101 (G. Lin et al., 2017)	47.1 (*msc*)
RefineNet-152 (G. Lin et al., 2017)	47.3 (*msc*)
RefineNet-LW-101 (ours)	45.1
RefineNet-LW-152 (ours)	45.8

Table 3.4: Quantitative results on the test set of PASCAL Context. Multi-scale evaluation is defined as *msc*.

3.5 Discussion

As noted above, we are able to achieve similar results by omitting more than half of the original parameters. This raises the question on how and why this does happen. We conduct a series of experiments aimed to provide some insights into this matter.

3.5.1 Receptive field size

First, we consider the issue of the receptive field size. Intuitively, dropping 3×3 convolutions should significantly harm the receptive field size of the original architecture. Nevertheless, we note that we do not experience this due to i.) the skip-design structure of RefineNet, where low-level features are being summed up with the high-level ones, and ii.) keeping CRP blocks that are responsible for gathering contextual information.

In particular, we explore the empirical receptive field (ERF) size (B. Zhou et al., 2015) in Light-Weight RefineNet-101 pre-trained on PASCAL VOC (Fig. 3.3). Before CRP ERF is concentrated around small object parts, and barely covers the objects. The CRP block tends to enlarge ERF, while the summation with the lower layer features further produces significantly larger activation contours.

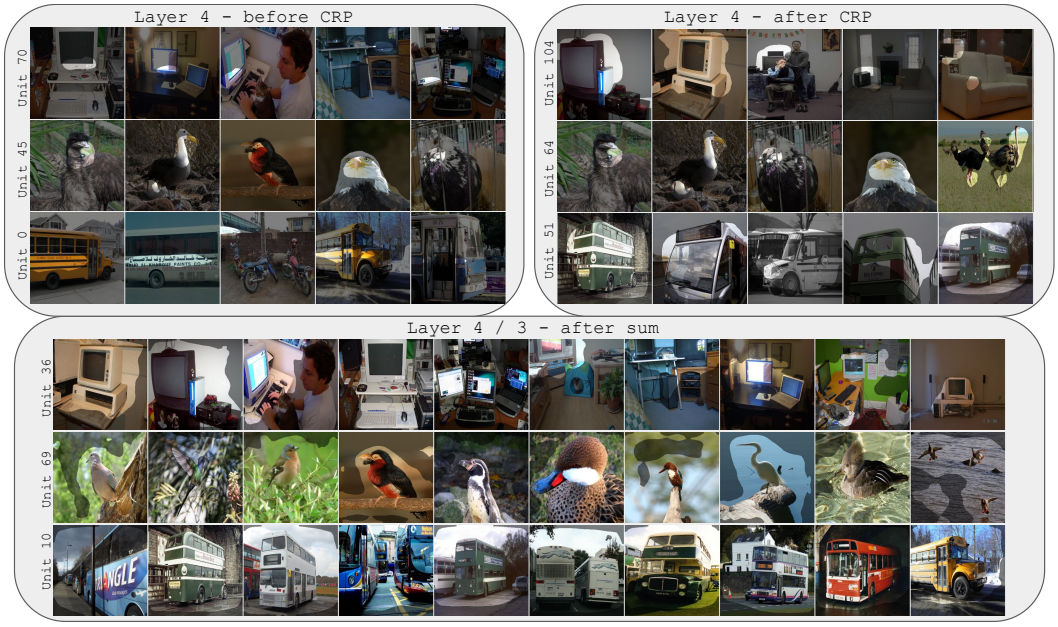


Figure 3.3: Comparison of empirical receptive field before CRP (top left), after CRP (top right), and after the summation of levels 4 and 3 (bottom) in RefineNet-LW-101 pre-trained on PASCAL VOC. Top activated images along with top activated regions are shown.

We also compare ERF of the last (classification) layer between original RefineNet-101 and RefineNet-LW-101 both pre-trained on PASCAL VOC. From Figure 3.4, it can be noticed that both networks exhibit semantically similar activation contours, although the original architecture tends to produce less jagged boundaries.

3.5.2 Representational power

Even though the RCU block seems not to influence the receptive field size of the network, it might still be responsible for producing important features for both segmentation and classification.

To evaluate whether it is the case indeed, we conduct ablation experiments by adding multi-label classification and segmentation heads in the lowest resolution block of RefineNet-101 pre-trained on PASCAL VOC 1.) before RCU, 2.) after RCU, and 3.) after CRP. We fix the rest of the trained network and fine-tune only these heads separately using 1464 training images of the VOC dataset. We evaluate all the results on the validation subset of VOC.

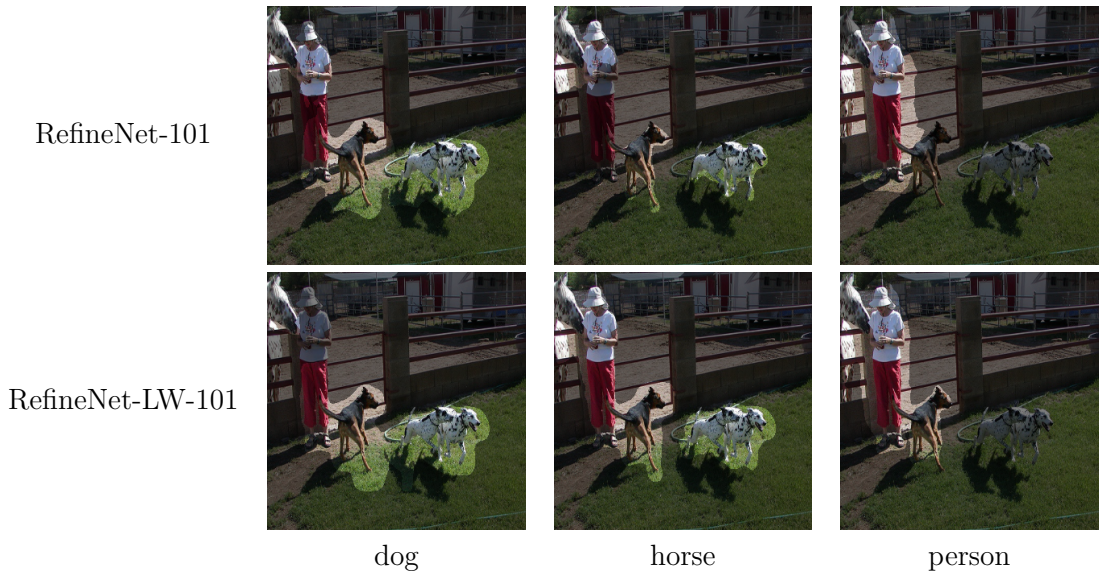


Figure 3.4: Comparison of empirical receptive field in the last (classification) layer between RefineNet-101 (top) and RefineNet-LW-101 (bottom). Top activated regions for each unit are highlighted.

Model	mIoU,%	acc,%
Before RCU	54.63	95.39
After RCU	55.16	95.42
After CRP	**57.83**	**97.71**

Table 3.5: Ablation experiments comparing CRP and RCU. Multi-label accuracy (without background) and mean IoU are reported as measured on the validation set of PASCAL VOC.

As seen from Table 3.5, CRP is the main driving force behind accurate segmentation and classification as the results after CRP are improved by more than 2% in both mean iou and accuracy, while the RCU block improves the results just marginally – by only 0.5% in mean iou and 0.03% in accuracy.

3.6 Conclusion

In this chapter, we tackled the problem of rethinking an existing semantic segmentation architecture into the one suitable for real-time performance, while keeping the performance levels mostly intact. We achieved that by proposing simple modifications to the existing network and highlighting which building blocks

were redundant for the final result. Our method can be applied along with any classification network for any dataset and can further benefit from using light-weight backbone networks. Quantitatively, we were able to closely match the performance of the original network while significantly surpassing its runtime and even acquiring 55 FPS on 512×512 inputs (from initial 20 FPS). Besides that, we demonstrated that having convolutions with large kernel sizes may be unnecessary in the decoder part of segmentation networks.

This chapter serves as an important milestone on our way towards building practical vision systems as we demonstrate the value of manually re-designing and analysing various components of a deep neural network used for arguably the most popular dense per-pixel task – semantic segmentation. As we show in the next chapter, the benefits of Light-Weight RefineNet are not limited to semantic segmentation, and are applicable to other dense per-pixel tasks, such as monocular depth prediction and surface normals estimation. More crucially, we will extend our approach to solve all these tasks at once in real-time using a single network.

From another perspective, in Chapter 5 we will build an algorithm that automatically finds compact and accurate architectures, thus reducing the number of manual decisions required from a human expert.

4

Real-Time Multi-Task Learning of Dense Per-Pixel Tasks

In the last chapter, we derived a compact and accurate semantic segmentation architecture through a careful re-design and analysis of a larger network. In this chapter, we will go beyond semantic segmentation and apply the same architecture to other dense per-pixel tasks, such as depth prediction and surface normals estimation. In particular, our focus will be on the real-time multi-task learning scenario where a single network solves several tasks at once.

In theory, multi-task learning leads to considerable savings of resources, and may often result in an overall performance improvement. In practice, a few prominent hurdles must be addressed, namely, i.) how to adapt a single model to perform multiple tasks at once, while ii.) doing it in real-time, and iii.) using asymmetric datasets with an unequal number of annotations per each modality. These hurdles are innate for various vision and non-vision tasks but are especially challenging for dense per-pixel tasks with extreme annotation costs, where it is rare to have ground truth labels for all the modalities at the same time.

To overcome the above issues, we will rely on Light-Weight RefineNet introduced in the previous chapter, and we will make several changes to it to further reduce the number of floating-point operations. To compensate for the asymmetry in

annotations, we will leverage hard knowledge distillation, where outputs of a large and slow 'teacher' network serve as noisy ground truth annotations for missing modalities. In addition to that, we will conduct experiments with a single model performing depth estimation and segmentation both indoors and outdoors at once, while also showcasing that the raw predictions of our network can be used to create a dense 3D semantic reconstruction of the indoor scene.

The results of this chapter first appeared in the proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2019*) under the title of 'Real-Time Joint Semantic Segmentation and Depth Estimation Using Asymmetric Annotations' (Nekrasov et al., 2019b).*[1]

4.1 Related Work

With semantic segmentation previously covered, we first turn our attention to two other inter-related dense per-pixel tasks – *depth prediction* and *surface normals estimation*. The goal of the first one is to determine the distance to each pixel in the image from the centre of the camera with which the image was taken, while the goal of the second one is to predict a 3-dimensional vector orthogonal to the surface at each pixel in the image. In contrast to the costly manual collection of semantic segmentation annotations required for supervised learning, several sensors exist that readily provide depth measurements – *e.g.* Kinect (Z. Zhang, 2012) and LiDAR (Wikipedia contributors, 2020). Given depth, it is further possible to acquire an estimate of surface normals using least squares (Silberman et al., 2012).[2] As in case of semantic segmentation, deep learning networks can be used to map a given image to depth or surface normals – *e.g.* Eigen et al. (2014), F. Liu et al. (2015), and Laina et al. (2016) did so in a fully-supervised manner, while Garg et al. (2016), Godard et al. (2017), Kuznietsov et al. (2017), and Godard et al. (2019) relied

[1] The trained models have been released and are available here: https://github.com/DrSleep/multi-task-refinenet with the training code example provided here: https://github.com/DrSleep/DenseTorch/tree/master/examples/multitask

[2] The estimate of the reverse (*i.e.* of depth given surface normals) can also be found, *e.g.* see (X. Qi et al., 2018).

on unsupervised, or semi-supervised learning, exploiting stereo pairs of images to impose various consistency losses based on the predicted depth.

The main focus of this chapter is on *multi-task learning*; according to Caruana (1993) and Baxter (2000), learning a single model that simultaneously performs several related tasks can improve generalisation via imposing an inductive bias on the learned representations – *i.e.* in effect regularising the set of all possible weights configurations. For example, the information learned from the semantic labels can help in depth prediction as closely located pixels from the same semantic category tend to have similar depth values; in reverse, nearby pixels with equal depth values are likely to belong to the same object and, hence, the same semantic class. Such dependencies will implicitly arise during the training of a multi-task network through the gradients of loss functions with respect to shared and task-specific parameters.

Multi-task computer vision works tend to follow this strategy: in particular, Eigen and Fergus (2015) trained a single architecture to predict depth and surface normals with a shared computation in the early layers (the authors also used the same architecture but with different weights for semantic segmentation). Kokkinos (2017) proposed a universal network to tackle 7 different vision tasks at once with shared features – the tasks included boundary detection, surface normals estimation, saliency estimation, semantic segmentation, human body part segmentation, semantic boundary detection and object detection. Dvornik et al. (2017) instead concentrated on the real-time component of joint semantic segmentation and object detection by appending only few layers with task-specific parameters at the end of the network, while Kendall et al. (2018) learned optimal task-specific loss weights with the help of Bayesian modelling to perform instance segmentation, semantic segmentation and depth estimation. Dharmasiri et al. (2017) showcased performance benefits of jointly predicting depth, surface normals and surface curvature, while X. Qi et al. (2018) directly encoded the geometric interrelation between depth and surface normals as a part of the network architecture. None of the aforementioned methods makes any use of already existing models for each separate task, and none of them, except BlitzNet (Dvornik et al., 2017), achieves real-time performance. In

contrast, we will show how to exploit large pre-trained models to stabilise the training of a smaller model and acquire better results, and how to do inference in real-time.

In the ideal scenario with an equal number of annotations per each given task, the multi-task training procedure barely differs from that of the single-task. The most interesting (and difficult) case arises when various tasks have various numbers of labels, making the learned features biased towards the tasks with the highest presence. To overcome this problem, Kokkinos (2017) resorted to asynchronous gradient updates, where the gradients computed from task-specific losses are being summed up and stored in-memory until a certain number of examples belonging to a particular task has been processed. After that, the standard gradient rule is applied (Sect. 2.1), the accumulated gradients are reset, and the process is repeated until convergence. In contrast to this approach, Dvornik et al. (2017) simply resorted to keeping the task branch with no ground truth available intact until at least one example of that modality is seen.

Several works also explored alternative ways of learning task-specific and shared features: Misra et al. (2016) proposed learnable cross-stitch units, which are linear combinations of representations from several single-task networks; Mallya et al. (2018) and Mallya and Lazebnik (2018) optimised for task-specific binary masks applied on weights inside a single network. Since for N tasks this requires N forward passes (with a selected mask turned on), S. Liu et al. (2019) instead used N task-specific blocks that learned to attend to a shared representation, thus reducing the number of forward passes at the expense of the number of parameters. Zamir et al. (2018) and Standley et al. (2019) further conducted an extensive set of experiments to computationally measure task relationships highlighting how various tasks must be learned together. While the above methods might be beneficial for a specific situation, they require either a non-trivial amount of computations (Zamir et al., 2018; Mallya and Lazebnik, 2018; Standley et al., 2019) or complicated design implementations (Misra et al., 2016; Mallya et al., 2018; S. Liu et al., 2019). Aligned with the goal of creating practical vision systems, we will instead choose a simple path of branching out into task-specific layers at the end of the network

requiring us to make only few changes to a single-task architecture presented in Chapter 3. In the next sections, we will demonstrate that this choice leads to highly competitive performance and can easily be extended to handle different situations, such as learning of multiple tasks from multiple domains at once.

4.2 Methodology

While we primarily discuss the case with only two tasks present, the same machinery applies for more tasks, as demonstrated in Sect. 4.4.1.

4.2.1 Backbone Network

As a backbone network we consider Light-Weight Refinet built on top of MobileNet-v2 (Sect. 3.3.3). Even though this network already achieves real-time performance and has a small number of parameters for a single task, for the joint estimation of depth and semantic segmentation (of 40 classes) it requires more than 14 GFLOPs on inputs of size 640×480, which may hinder it from the direct deployment on mobile platforms with few resources available. In particular, the last chained residual pooling (CRP) block is responsible for more than half of the FLOPs as it deals with the high-resolution feature maps (1/4 from the original resolution). This cost is due to 1×1 convolutions that are equivalent to a matrix product between $C_{in}\times H\times W$ input tensor with C_{in} input channels and $C_{in}\times C_{out}$ weight matrix with C_{out} output channels. Since for convolutions used in CRP we have $C_{in} = C_{out} = C$, instead of a scalar weight for each pair of input and output channels (hence the dimension $C_{in}\times C_{out}$ of the weight matrix), we can resort to a single weight vector of size C multiplied $H\times W$ times with each input feature (see Fig. 4.1). This reduces the number of operations by more than half, down to just 6.5 GFLOPs. Such convolution is often called *depthwise convolution* (Sifre and Mallat, 2014; Chollet, 2017).

4.2.2 Joint Semantic Segmentation and Depth Estimation

In the general case, it is non-trivial to decide where to branch out the backbone network into separate task-specific paths in order to achieve the optimal performance

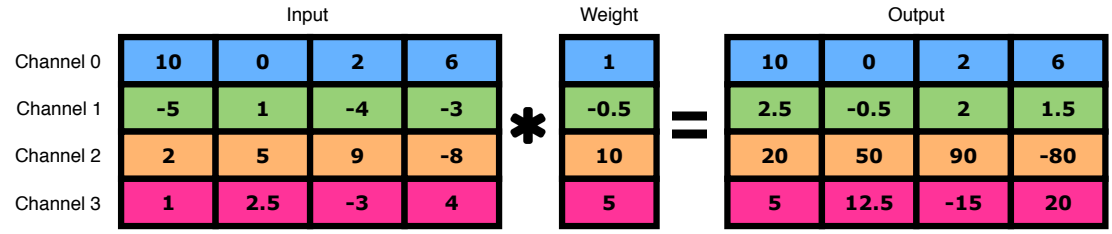


Figure 4.1: An example of depthwise 1×1 convolution on 1×4 input with the stride of 1 and 4 input-output channels. It is equivalent to a scalar multiplication of each pixel with the weight value of the same colour. In this example, the numbers (**10**, **0**, **2**, **6**) in the output of Channel 0 are equal to the numbers in the input of Channel 0 multiplied by the corresponding weight value of **1**.

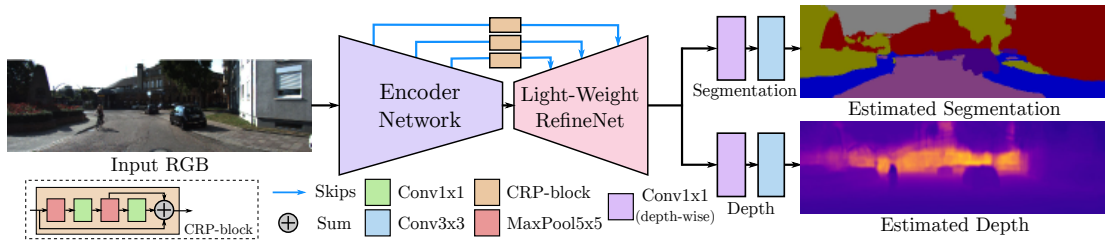


Figure 4.2: General network structure for joint semantic segmentation and depth estimation. Each task has only 2 specific parametric layers, while everything else is shared. In the depth image, closer pixels have **dark violet** colours, while pixels further away are coloured **orange**.

on all of the tasks simultaneously. For simplicity, we branch out right after the last CRP block, and append two additional convolutional layers (one depthwise 1×1 and one plain 3×3) for each task (Fig. 4.2).

If we denote the output of the network before the branching as $\tilde{y} = f_{\theta_b}(X)$, where f_{θ_b} is the backbone network with a set of parameters θ_b, and X is the input RGB-image, then the depth and segmentation predictions can be denoted as $\tilde{y}_s = g_{\theta_s}(\tilde{y})$ and $\tilde{y}_d = g_{\theta_d}(\tilde{y})$, where g_{θ_s} and g_{θ_d} are segmentation and depth estimation branches with the sets of parameters θ_s and θ_d, respectively. We use the standard softmax cross-entropy loss for segmentation (Eqn. 2.1) and the reverse Huber loss for depth estimation (Laina et al., 2016). Our total loss (Eqn. 4.1) contains an additional scaling parameter, λ, which, for simplicity, we set to 0.5:

$$\mathcal{L}_{total}(X, Y_s, Y_d; \theta_b, \theta_s, \theta_d) = \lambda \cdot \mathcal{L}_{segm}(X, Y_s; \theta_b, \theta_s) \ + (1 - \lambda) \cdot \mathcal{L}_{depth}(X, Y_d; \theta_b, \theta_d),$$
$$\mathcal{L}_{depth}(X, Y) = \begin{cases} |\tilde{y}_d - Y|, \ \text{if } |\tilde{y}_d - Y| \leq c \\ ((\tilde{y}_d - Y)^2 + c^2)/(2c), \ \text{otherwise,} \end{cases}$$
$$c \stackrel{\text{def}}{=} 0.2 \cdot \max |\tilde{y}_d - Y|, \tag{4.1}$$

where Y_s and Y_d denote ground truth segmentation mask and depth map, correspondingly.

4.2.3 Expert Labeling for Asymmetric Annotations

As one would expect, it is impossible to have all the ground truth sensory information available for every single image. Quite naturally, this poses a question of how to deal with a set of images $S = \{X\}$ among which some have an annotation of one modality, but not another. Assuming that one modality is always present for each image, this then divides the set S into two disjoint sets $S_1 = S_{T_1}$ and $S_2 = S_{T_1,T_2}$ such that $S = S_1 \cup S_2$, where T_1 and T_2 denote two tasks, respectively, and the set S_1 consists of images for which there are no annotations of the second task available, while S_2 comprises images having both sets of annotations.

Plainly speaking, there is nothing that prohibits us from still exploiting equation (4.1), in which case only the weights of the branch with available labels will be updated. As we will show in our ablation experiments (Sect. 4.3.2), this leads to biased gradients and, consequently, sub-optimal solutions. Instead, emphasising the need for updating both branches simultaneously, we rely on an expert model to provide us with noisy estimates in place of missing annotations. By the expert model here we understand a larger neural network able to solve the given task with a certain accuracy. This concept is in line with knowledge distillation (Ba and Caruana, 2014; Hinton et al., 2014; Bucila et al., 2006; Romero et al., 2015), where the expert model is called *teacher* (see Sect. 3.1 for a brief discussion on knowledge distillation).

More formally, if we denote the expert model on the second task as E_{T_2}, then its predictions $\tilde{S}_1 = E_{T_2}(S_1)$ on the set S_1 serve as proxy ground truth data, which

we will use to pre-train our joint model before the final fine-tuning on the original set S_2 with readily available ground truth data for both tasks.

Note also that our framework is directly transferable to the case where the set S comprises several datasets. In Sect. 4.4.2 we showcase a way of exploiting all of them in the same time using a single copy of the model.

4.3 Experimental Results

In our experiments, we consider two datasets, NYUDv2-40 (Silberman et al., 2012; Gupta et al., 2013) and KITTI (Geiger et al., 2013; Ros et al., 2015), representing indoor and outdoor settings, respectively, and both being used extensively in the robotics community.

All the training experiments follow the same protocol. In particular, we initialise the classifier part using the weights pre-trained on ImageNet (Deng et al., 2009), and train using mini-batch SGD with momentum with the initial learning rate of $1e{-}3$ and the momentum value of 0.9. Following the setup of Light-Weight RefineNet from Chapter 3, we keep batch norm statistics frozen. We divide the learning rate by 10 after pre-training on a large set with proxy annotations. We train with a random square crop of 350×350 augmented with random mirror (*i.e.* random horizontal flip).

4.3.1 Evaluation metrics

To report the quality of depth estimation for a given set D with examples $\{i\}$, ground truth depth maps $\{Y_i\}$ and predicted depth $\{\tilde{y}_i\}$ we will compute the metrics from Table 4.1. For surface normals we will compute the angle between the predicted and the ground truth vectors and report the mean and median of this distribution.

4.3.2 NYUDv2

NYUDv2 is an indoor dataset with 40 semantic labels containing 1449 RGB images with both segmentation and depth annotations, of which 795 comprise the training set and 654 – validation. The raw dataset has more than 300K training images with only depth annotations. During training, we use less than 10% (25K images)

Name	Abbr.	Equation
Linear Root Mean Square Error	RMSE lin	$\sqrt{\frac{1}{\lvert D\rvert}\sum_{i=1}^{i=\lvert D\rvert}\lvert\lvert Y_i-\tilde{y}_i\rvert\rvert^2}$
Log Root Mean Square Error	RMSE log	$\sqrt{\frac{1}{\lvert D\rvert}\sum_{i=1}^{i=\lvert D\rvert}\lvert\lvert \log(Y_i)-\log(\tilde{y}_i)\rvert\rvert^2}$
Absolute Relative Difference	abs rel	$\frac{1}{\lvert D\rvert}\sum_{i=1}^{i=\lvert D\rvert}\frac{\lvert Y_i-\tilde{y}_i\rvert}{Y_i}$
Squared Relative Difference	sqr rel	$\frac{1}{\lvert D\rvert}\sum_{i=1}^{i=\lvert D\rvert}\frac{\lvert\lvert Y_i-\tilde{y}_i\rvert\rvert^2}{Y_i}$
Threshold	$\delta < threshold$	$\frac{1}{\lvert D\rvert}\sum_{i=1}^{i=\lvert D\rvert}(\max(\frac{Y_i}{\tilde{y}_i},\frac{\tilde{y}_i}{Y_i}) < threshold)$

Table 4.1: Commonly used metrics for depth estimation (Eigen et al., 2014). For the reported RMSE, abs rel and sqr rel the lower the better, whereas for accuracies (δ) the higher the better.

of this data following Dharmasiri et al. (2018). As discussed in Sect. 4.2.3, we annotate these images for semantic segmentation using a teacher network – here, we take Light-Weight RefineNet-152 that achieves 44.4% mean iou on the validation set (Sect. 3.4.1). After acquiring the proxy annotations, we pre-train the network on the large set and then fine-tune it on the original small set of 795 images.

Quantitatively, we are able to achieve 42.02% mean iou and 0.565m RMSE (lin) on the validation set (Table 4.2), outperforming several large models, while performing both tasks in real-time simultaneously. In depth estimation we achieve the lowest *sqr rel* and joint highest accuracy for the threshold of 1.25^3 (Table 4.3). Qualitatively (Fig. 4.3) our model predicts smooth depth maps and able to correctly segment the majority of objects present in the environment.

Ablation Studies. To evaluate the importance of pre-training using the proxy annotations and the benefits of performing two tasks jointly, we conduct a series of ablation experiments. In particular, we compare three baseline models trained on the small set of 795 images and three other approaches that make use of additional data – ours with noisy estimates from a larger model, and two methods, one by Kokkinos (2017), where the gradients are being accumulated until a certain number of examples is seen, and one by Dvornik et al. (2017), where the task branch is updated every time at least one example with the corresponding task annotation is seen.

		Segm.	Depth	General		
Model	Regime	mIoU,%	RMSE (lin),m	Param.,M	GFLOPs	Runtime,ms
†Multi-Task RefineNet-LW	Segm,Depth	42.02	0.565	**3.07**	**6.49**	**12.8±0.1**
RefineNet-101 (G. Lin et al., 2017)	Segm	**43.6**	–	118	–	60.3 ± 0.5
RefineNet-LW-50	Segm	41.7	–	27	33	19.6 ± 0.3
Context (G. Lin et al., 2016)	Segm	40.6	–	–	–	–
†Sem-CRF+ (Mousavian et al., 2016)	Segm,Depth	39.2	0.816	–	–	–
‡Kendall and Gal (2017)	Segm,Depth	37.3	**0.506**	–	–	150
Zuo and Drummond (2017)	Segm	34.3	–	–	–	48.4
‡Eigen and Fergus (2015)	Segm,Depth	34.1	0.641	–	–	–
Laina et al. (2016)	Depth	–	0.573	63.6	–	55
†X. Qi et al. (2018)	Depth,Normals	–	0.569	-	–	870

Table 4.2: Results on the test set of NYUDv2. The speed of a single forward pass and the number of FLOPs are measured on 640×480 inputs. For the reported mIoU the higher the better, whereas for the reported RMSE the lower the better. (†) means that both tasks are performed simultaneously using a single model, while (‡) denotes that two tasks employ the same architecture but use different copies of weights per task.

	Ours	Laina et al. (2016)	Kendall and Gal (2017)	X. Qi et al. (2018)
RMSE (lin)	0.565	0.573	**0.506**	0.569
RMSE (log)	0.205	**0.195**	–	–
abs rel	0.149	0.127	**0.11**	–
sqr rel	**0.105**	–	–	0.128
$\delta < 1.25$	0.790	0.811	0.817	**0.834**
$\delta < 1.25^2$	0.955	0.953	0.959	**0.960**
$\delta < 1.25^3$	**0.990**	0.988	0.989	**0.990**

Table 4.3: Detailed results on the test set of NYUDv2 for the depth estimation task. For the reported RMSE, abs rel and sqr rel the lower the better, whereas for accuracies (δ) the higher the better.

The results of our experiments are given in Table 4.4. The first observation that we make is that performing two tasks jointly on the small set does not provide any significant benefits for each separate task, and even substantially harms semantic segmentation – 32.5% mean iou and 0.633m RMSE of the joint model against 34.4% mean iou and 0.638m RMSE of the single-task models. In contrast, having a large set of depth annotations results in improvements both in depth estimation and even semantic segmentation, when it is coupled with a clever strategy of accumulating gradients – 34.8% mean iou and 0.582m RMSE for BlitzNet (Dvornik et al., 2017) and 35.9% mean iou and 0.573m RMSE for UberNet (Kokkinos, 2017). Nevertheless, none of the methods can achieve competitive results on semantic segmentation, whereas our proposed approach reaches 42.0% mean iou and 0.565m RMSE without

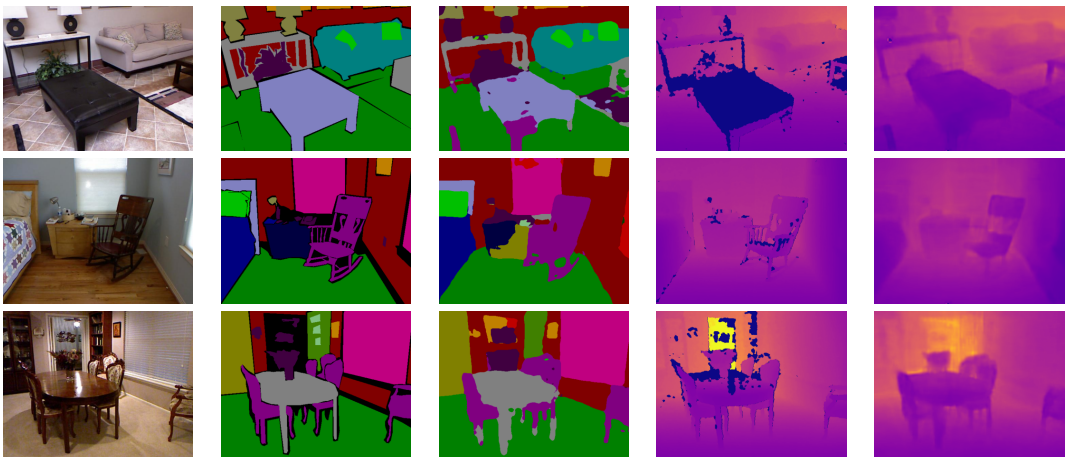


Figure 4.3: Qualitative results on the test set of NYUD-v2. The black and dark-blue pixels in 'GT-Segm' and 'GT-Depth', respectively, indicate pixels without an annotation or label.

any changes to the underlying optimisation algorithm.

	Annotations		Update Frequency		Segm.	Depth
Method	Pre-Training	Fine-Tuning	Task	Base	mIoU,%	RMSE (lin),m
Baseline (SD)	795SD	–	1	1	32.48	0.6328
Baseline (S)	795S	–	1	1	34.44	–
Baseline (D)	795D	–	1	1	–	0.6380
BlitzNet (Dvornik et al., 2017)	25405D + 795SD	795SD	1	1	34.82	0.5823
UberNet (Kokkinos, 2017)	25405D + 795SD	795SD	10	30	35.88	0.5728
Multi-Task RefineNet-LW	25405SD	795SD	1	1	**42.02**	**0.5648**

Table 4.4: Results of ablation experiments on the test set of NYUDv2. *SD* means how many images have a joint pair of annotations – both segmentation (S) and depth (D); *task update frequency* denotes the number of examples of each task to be seen before performing a gradient step on task-specific parameters; *base update frequency* is the number of examples to be seen (regardless of the task) before performing a gradient step on shared parameters.

4.3.3 KITTI

KITTI is an outdoor dataset that contains 100 images semantically annotated for training (with 11 semantic classes) and 46 images for testing (Ros et al., 2015) without ground truth depth maps. Following the previous work by S. Wang et al. (2015) we only keep 6 well-represented classes.

Besides segmentation, we follow Eigen et al. (2014) and employ 20K images

		Segm.	Depth		General			
Model	Regime	mIoU,%	RMSE (lin),m	RMSE (log)	Param.,M	Input Size	GFLOPs	Runtime,ms
Multi-Task RefineNet-LW	Segm,Depth	**87.02**	**3.453**	0.182	**2.99**	1200x350	**6.45**	**16.9±0.1**
Zuo and Drummond (2017)	Segm	84.9	–	–	–	1200x350	–	106.35
S. Wang et al. (2015)	Segm	74.8	–	–	–	–	–	–
Garg et al. (2016)	Depth	–	5.104	0.273	–	–	–	–
Godard et al. (2017)	Depth	–	4.471	0.232	31	512x256	–	35.0
Kuznietsov et al. (2017)	Depth	–	3.518	**0.179**	–	621x187	–	48.0

Table 4.5: Results on the test set of KITTI-6 for segmentation and KITTI for depth estimation.

with depth annotations available for training (Geiger et al., 2013), and 697 images for testing. Due to the similarities with the CityScapes dataset (Cordts et al., 2016), we consider ResNet-38 (Z. Wu et al., 2019) trained on CityScapes as our teacher network and use it to annotate the training images that have depth but not semantic segmentation. In turn, to annotate missing depth annotations on 100 images with semantic labels from KITTI-6, we first train a separate copy of our network on the depth task only and then use it as a teacher. Note that we abandon this copy of the network and do not make any further use of it.

After pre-training on the large set, we fine-tune the model on the small set of 100 examples. Among other methods (Table 4.5) we achieve the highest mean iou of 87% (against the second-best of 84.9% by Zuo and Drummond (2017)) and the lowest linear RMSE of 3.45m. In the semantic segmentation task we outperform other methods in 4 out of 6 classes (Table 4.6). Besides being superior across a large set of metrics, our approach is also light-weight with low latency. We further visualise a few of the network predictions in Fig. 4.4.

Model	sky	building	road	sidewalk	vegetation	car	Total
Multi-Task RefineNet-LW	85.1	**87.7**	**92.8**	**82.7**	86.1	**87.6**	**87.0**
Zuo and Drummond (2017)	84.5	85.9	92.3	78.8	**87.8**	80.3	84.9
S. Wang et al. (2015)	**88.6**	80.1	80.9	43.6	81.6	63.5	74.8

Table 4.6: Detailed segmentation results on the test set of KITTI-6.

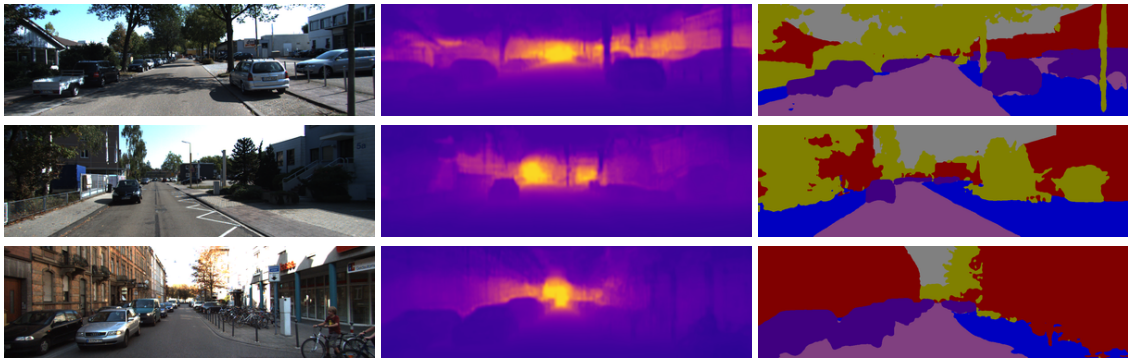


Figure 4.4: Qualitative results on the test set of KITTI (for which only GT depth maps are available). We do not visualise GT depth maps due to their sparsity.

4.4 Extensions

The goal of this section is to demonstrate the ease with which our approach can be directly applied in other practical scenarios, such as, for example, the deployment of a single model performing three tasks at once, and the deployment of a single model performing two tasks at once under two different scenarios – indoor and outdoor. As the third task, here we consider surface normals estimation, and as two scenarios, we consider training a single model on both NYUD and KITTI simultaneously without the necessity of having a separate copy of the same architecture for each dataset.

In this section, we strive for simplicity and do not aim to achieve high-performance numbers, thus we directly apply the same training scheme as outlined in the previous section.

4.4.1 Single Model – Three Tasks

Analogously to the depth and segmentation branches, we append the same structure with two convolutional layers for surface normals. We employ the negative dot product (after l_2-normalisation) as the training loss for surface normals, and we multiply the learning rate for the normals parameters by 10, as done by Eigen and Fergus (2015).

We exploit the full raw training set of NYUDv2, having (noisy) depth maps from the Kinect sensor and with surface normals computed using the toolbox provided

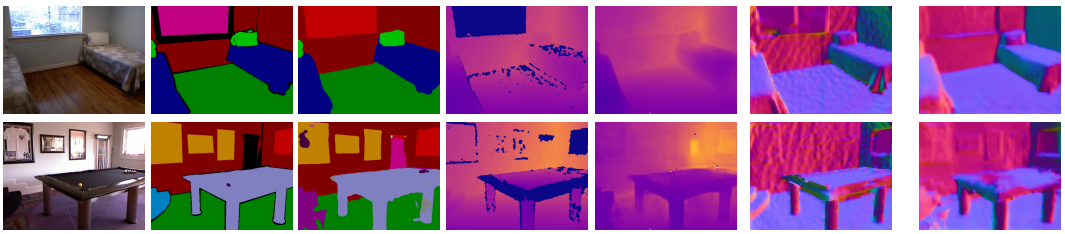


Figure 4.5: Qualitative results on the test set of NYUD-v2 for three tasks. The black pixels in the 'GT-Segm' images indicate those without a semantic label, whereas the dark blue pixels in the 'GT-Depth' images indicate missing depth values.

by the authors (Silberman et al., 2012). To acquire missing segmentation labels, we repeat the same procedure outlined in the main experiments – in particular, we use the Light-Weight RefineNet-152 network to get noisy labels. After pre-training on this large dataset, we divide the learning rate by 10 and fine-tune the model on the small dataset of 795 images having annotations for each modality.

Our straightforward approach achieves practically the same numbers on depth estimation (0.565m RMSE of the two-task model against 0.566m RMSE of the three-task model) but suffers a significant performance drop of nearly 3.5% mean iou on semantic segmentation (Table 4.7). This might be directly caused by the excessive number of imperfect and noisy labels, on which the semantic segmentation part is being pre-trained. Nevertheless, the results on all three tasks remain competitive, and we are able to perform all three of them in real-time simultaneously, on average spending less than 13.5ms per a single image. We provide a few examples of our approach in Figure 4.5.

Segm.	**Depth**		**Surface Normals**		**General**
mIoU,%	RMSE (lin),m	RMSE (log)	Mean Angle	Median Angle	Runtime,ms (mean/std)
38.66	0.566	0.209	23.95	17.74	13.4±0.1
42.02	**0.565**	**0.205**	–	–	**12.8±0.1**

Table 4.7: Results on the test set of NYUDv2 of our single network predicting three modalities at once with surface normals annotations from Silberman et al. (2012). The speed of a single forward pass is measured on 640 × 480 inputs. Baseline results (with a single network performing only segmentation and depth) are in **bold**.

4.4.2 Single Model – Two Datasets, Two Tasks

Next, we consider the case when it is undesirable to have a separate copy of the same model architecture for each dataset. Concretely, our goal is to train a single model that can perform semantic segmentation and depth estimation on both NYUD and KITTI at once. To this end, we simply concatenate both datasets and amend the segmentation branch to predict 46 labels (40 from NYUD and 6 from KITTI-6).

We follow the same training strategy, and after pre-training on the union of large sets we fine-tune the model on the union of small training sets. Our network exhibits no difficulties in differentiating between two regimes (Table 4.8) and achieves results on the same level with two separate networks. In particular, the single network shows 38.8% mean iou on NYUD (nearly 3.5% decrease against the NYUD-only model) and 86.1% mean iou on KITTI (less than 1% decrease against the KITTI-only model), while for depth estimation only linear RMSE drops significantly – by 0.25m and 0.2m on NYUD and KITTI, respectively.

NYUDv2			**KITTI**		
Segm.	**Depth**		**Segm.**	**Depth**	
mIoU,%	RMSE (lin),m	RMSE (log)	mIoU,%	RMSE (lin),m	RMSE (log)
38.76	0.59	0.213	86.1	3.659	0.190
42.02	**0.565**	**0.205**	**87.0**	**3.453**	**0.182**

Table 4.8: Results on the test set of NYUDv2, KITTI (for depth) and KITTI-6 (for segmentation) of our single network predicting two modalities on both datasets together. Baseline results (with separate networks per dataset) are in **bold**.

4.4.3 Dense Semantic SLAM

Finally, we demonstrate that quantities predicted by our joint network performing depth estimation and semantic segmentation indoors can be directly incorporated into an existing SLAM framework.

In particular, we consider SemanticFusion (McCormac et al., 2017), where the SLAM reconstruction is carried out by ElasticFusion (Whelan et al., 2016), which relies on RGB-D (RGB-image and depth map) inputs in order to find dense

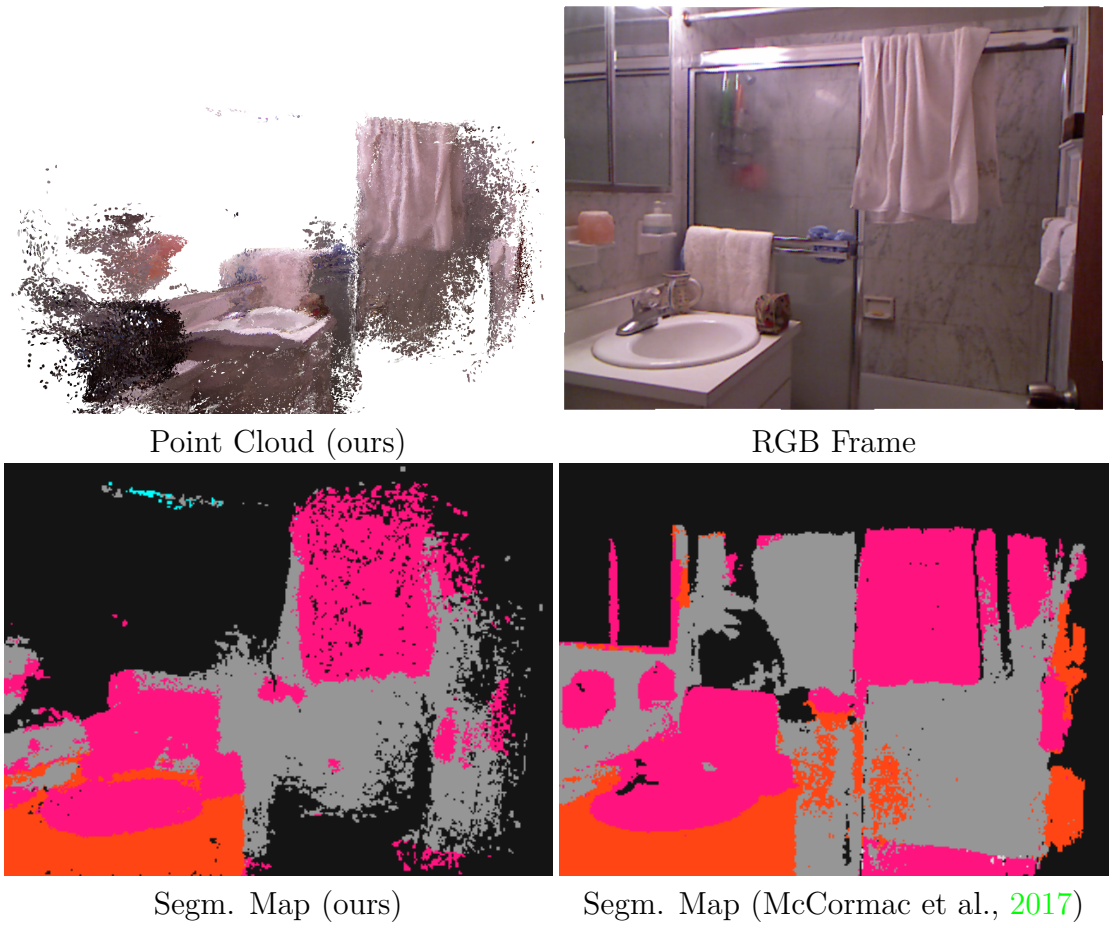


Figure 4.6: 3D reconstruction output using our per-frame depths and segmentation inside SemanticFusion (McCormac et al., 2017). The segmentation map by McCormac et al. (2017) was computed using Kinect depth measurements and a standalone semantic segmentation network; in contrast, our results are based on a single network performing both depth estimation and semantic segmentation at the same time.

correspondences between frames. A separate CNN, also operating on RGB-D inputs, was used by McCormac et al. (2017) to acquire 2D semantic segmentation map of the current frame. A dense 3D semantic map of the scene is obtained with the help of tracked poses predicted by the SLAM system.

We consider one sequence of the NYUD validation set provided by the authors[3], and directly replace ground truth depth measurements with the outputs of our network performing depth and segmentation jointly (Sect. 4.3.2). Likewise, we do not make use of the authors' segmentation CNN and instead exploit segmentation predictions from our network with the results directly re-mapped into the 13-classes

[3] https://bitbucket.org/dysonroboticslab/semanticfusion/overview

domain (Couprie et al., 2013). We visualise dense surfel-based reconstruction along with the dense segmentation and current frame in Fig. 4.6. Please refer to the video material for the full sequence results.[4]

4.5 Conclusion

Based on the foundations laid in the previous chapter, here we extended the architecture of Light-Weight RefineNet and presented a simple way of achieving real-time performance for the joint task of depth estimation and semantic segmentation. We showcased that it is possible (and indeed beneficial) to re-use large existing models in order to generate proxy labels important for the pre-training stage of a compact model. Moreover, we demonstrated how our method can be easily extended to handle more tasks and more datasets simultaneously, while raw depth and segmentation predictions of our network can be seamlessly used within available dense SLAM systems. Our multi-task network solves two tasks of semantic segmentation and depth estimation simultaneously in 12.8ms on average on 640×480 inputs, and solves three tasks (with the addition of surface normals) in 13.4ms on average.

It is important to re-emphasise that efficient and effective exploitation of visual information using deep learning models is crucial for further development and deployment of robots and autonomous vehicles, and the creation of practical vision systems. While so far we have achieved these goals by manually designing necessary components, in the next two chapters we will turn our attention to an automated search of compact and accurate networks. In the next chapter in particular we will once again showcase how semantic segmentation architectures can be transferred to the task of depth estimation.

[4] https://youtu.be/qwShIBhaq8Y

5

Neural Architecture Search of Efficient Dense Per-Pixel Networks

In the last two chapters, we were designing efficient single-task and multi-task fully convolutional networks manually through the removal of redundant layers and replacement of expensive operations with cheaper alternatives in an otherwise untouched encoder-decoder structure. In this chapter we will formally define a search space of compact architectures, increasing the range of possible structural changes that can be applied to the baseline network and considering a larger number of operations. To discover high-performing networks we will move from manual analysis to the usage of an automated algorithm.

In doing so we will be relying on ***reinforcement learning*** *(R. S. Sutton and Barto, 1998), abbreviated as RL. In the traditional RL setting an agent (*e.g. *a robot) interacts with its environment through a sequence of actions and receives a reward signal based on its decisions. The agent's goal is to improve its strategy of selecting actions (also called "policy") to maximise the expected reward. In our case, in place of the agent we will have another neural network, called "controller", with the search space of compact segmentation architectures serving as the environment. Each sequence of actions made by the controller will define a single design from*

this search space, with the controller receiving the reward based on the performance of the corresponding architecture.

Our main focus here will be on semantic segmentation, although we will also demonstrate the transferability of discovered architectures to another dense per-pixel task of depth estimation. To reduce the computational burden, we will only search for decoder structures given a pre-trained encoder part. Later in Chapter 6 we will introduce a series of modifications to the search space that would allow us to lessen the dependency on the fully-defined encoder.

We will further show that our methodology automatically discovers architectures that perform better and faster than manually designed ones with a comparable number of training parameters.

The results outlined in this chapter first appeared in the proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019*) under the title of 'Fast Neural Architecture Search of Compact Semantic Segmentation Models via Auxiliary Cells' (Nekrasov et al., 2019a).*[1]

5.1 Related Work

Traditionally, architecture search methods have been relying upon evolutionary strategies (Angeline et al., 1994; Stanley and Miikkulainen, 2002; Stanley et al., 2009), where a population of networks (oftentimes together with their weights) is being continuously mutated with less promising networks being discarded. Since these methods often require an extensive number of iterations, modern neuro-evolutionary approaches (Real et al., 2017; H. Liu et al., 2018) greatly benefit from available computational resources that facilitate search in much larger spaces and permit mutations of whole layers.

Bayesian optimisation (*BO*) methods, on the other hand, have long been used for hyper-parameter search (Snoek et al., 2012; Bergstra et al., 2013); they work by estimating the probability density of an objective function (or its surrogate)

[1]The algorithm implementation and discovered models are available here: https://github.com/DrSleep/nas-segm-pytorch

and selecting the most promising next candidate for the evaluation. While BO can also find hyper-parameters of each individual layer in the given neural network, reasoning about whole network structures and comparing different architectures in BO are not trivial; these directions are at the centre of an ongoing work with few kernel-based solutions already showing solid results in discovering high-performing designs (Swersky et al., 2014; Kandasamy et al., 2018).

Most recently, ***neural architecture search*** (NAS) strategies based on reinforcement learning (Fig. 5.1) have become popular across various domains (Baker et al., 2017; Zoph and Le, 2017; Zoph et al., 2018). Relying on enormous computational resources, these algorithms comprise a separate neural network, *controller*, that emits an architecture design and receives a scalar reward after the architecture is trained on the task of interest. The controller is then updated to maximise the reward of architectures it produces. Notably, thousands of iterations and GPU-days are usually needed for convergence.

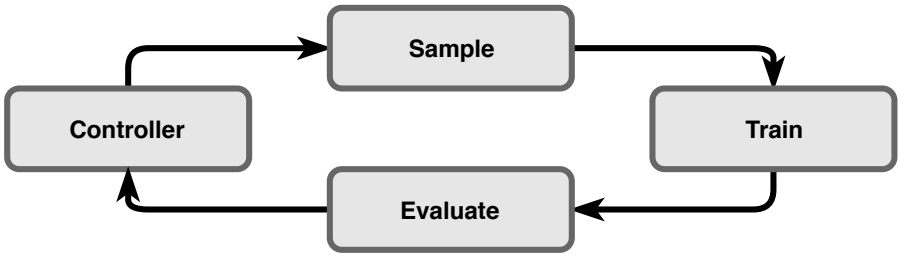


Figure 5.1: High-level overview of NAS with RL: an architecture is first sampled from a recurrent neural network, controller, and then trained on the meta-train set. Finally, the validation score on the meta-val set is used as the reward signal to train the controller.

The first NAS work in image classification (Zoph and Le, 2017) emitted a full description of the layers to use – including the number of input/output channels, kernel sizes, strides, etc. To limit the search space Zoph et al. (2018) later proposed to search only for two sets of operations – so-called, *reduction* and *normal* cells that are stacked and arranged in multiple blocks to form the final network. In a similar vein, H. Liu et al. (2018) defined so-called *motifs* that comprised several operations in a unified block – those were used inside an evolutionary search algorithm and underwent mutations depending on their fitness scores.

Several other works have focused on making NAS methods more efficient. In particular, Pham et al. (2018) unrolled the computational graph of all possible architectures and allowed sharing the weights among different architectures. Since this alleviated the need to re-train each sampled architecture from scratch, the number of resources required to run the search process was dramatically reduced to a single GPU. C. Liu et al. (2018) instead exploited a progressive strategy where the network complexity was gradually increased, and the ranking network was trained in parallel to predict the performance of a new architecture. A few other methods have been built around continuous relaxation of the search problem. Particularly, Luo et al. (2018) used an encoder to embed the architecture description into a latent space, and an estimator to predict the performance of an architecture given its embedding. Similar to Pham et al. (2018), H. Liu et al. (2019) initialised a set of all possible architectures, but instead of relying on RL, applied continuous relaxation to directly optimise a weighted average of all possible networks at once. Although these methods succeed in reducing computational requirements, they achieve so by significantly limiting the expressiveness of the search space, and hence may result in a sub-optimal solution.

NAS approaches for semantic segmentation have followed in the footsteps of NAS in image classification – in particular, L. Chen et al. (2018a) used random search to find a single cell on top of a fully convolutional classifier, while C. Liu et al. (2019) directly adapted the continuous learning relaxation method of H. Liu et al. (2019) to find a whole network structure. Importantly, L. Chen et al. (2018a) needed *nearly* 400 *GPUs over the range of* 7 *days*. In contrast, our presented method is able to find *compact* segmentation models only in a fraction of that time – *in* 4 *days on* 2 *GPUs*, which is more efficient by more than two orders of magnitude. We further differ from C. Liu et al. (2019) in that we rely on RL, do not perform weight sharing across architectures and, as in the previous chapters, focus exclusively on compact networks for practical applications.

5.2 Methodology

5.2.1 Overview

The focus of this chapter is on an automatic discovery of compact high-performing fully convolutional architectures, able to run in real-time on a low-computational budget, for example, on the Jetson platform. To this end, we are explicitly looking for structures that not only improve the performance on the held-out set but also facilitate the optimisation during the training stage.

As in the previous chapters we consider the encoder-decoder type of a fully-convolutional network (Long et al., 2015). While the encoder is still a pre-trained image classifier, the decoder structure here is emitted by the controller network; we only search for the decoder part due to computational limitations and the fact that the majority of gains in semantic segmentation came from better decoder designs – *e.g.* RefineNet (G. Lin et al., 2017), PSPNet (Zhao et al., 2017), DeepLab (L. Chen et al., 2018b).

The controller generates the connectivity structure between the encoder and the decoder, as well as the sequence of operations (that form the so-called *cell*) to be applied on each connected path. The same cell structure is used to form an auxiliary classifier, the goal of which is to provide intermediate supervision and to implicitly over-parameterise the model. Over-parameterisation is believed to be the primary reason behind the successes of deep learning models, and a few theoretical works have already addressed it in simplified cases (Soltanolkotabi et al., 2018; Du and Lee, 2018). Along with empirical results, this is the primary motivation behind the described approach. We discuss the details of the auxiliary cell later in Sect. 5.2.4.

With the above in mind, we devise a search strategy that permits discovering high-performing architectures within a small number of days using only few GPUs. Concretely, we pursue two goals here:

i.) To prevent *'bad'* architectures from being trained for long; and

ii.) To achieve a solid performance estimate as soon as possible.

To tackle the first goal, we divide the training process during the search into two stages. During the first stage, we fix the encoder's weights and pre-compute its outputs, while only training the decoder part. For the second stage, we train the whole model end-to-end. We validate the performance after the first stage and terminate the training of unpromising architectures. For the second goal, we employ Polyak averaging (Polyak and Juditsky, 1992) and knowledge distillation (Hinton et al., 2014) to speed-up the convergence.

To summarise, our main contribution in this chapter is an efficient neural architecture search strategy for semantic segmentation that (i.) allows to sample compact high-performing architectures, which, in turn, (ii.) can be used in real-time on low-computing platforms, such as JetsonTX2. In particular, the above points are made possible by:

- Devising a progressive strategy that eliminates poor candidates early in the training;
- Developing a training schedule for semantic segmentation that quickly provides reliable results via the means of knowledge distillation and Polyak averaging;
- Searching for an over-parameterised auxiliary cell that facilitates the training process and is obsolete during inference.

5.2.2 Problem Formulation

We consider a dense prediction task T, for which we have multiple training tuples $\{(X_i, Y_i)\}$, where both X_i and Y_i are 3-dimensional tensors with equal spatial and arbitrary third dimensions. In this work, X_i is a 3-channel RGB image, while Y_i is a C-channel one-hot segmentation mask with C being equal to the number of classes. Furthermore, we rely on a mapping $f : X \to Y$ with parameters θ, that is represented by a fully convolutional neural network. We assume that the network f can further be decomposed into two parts: e – representing the encoder, and d – for the decoder. We initialise the encoder e with weights from a pre-trained classification network consisting of multiple down-sampling operations that reduce the spatial dimensions of the input. The decoder part, on the other hand, has access

to several outputs of the encoder with varying spatial and channel dimensions. The search goal is to choose which feature maps to use and what operations to apply to them. We next describe the decoder search space in full detail.

5.2.3 Search Space

As mentioned above, the decoder has access to multiple layers from the pre-trained encoder with varying dimensions. To keep sampled architectures compact and of approximately equal size, each encoder output undergoes a single 1×1 convolution with the same number of output channels. We rely on a recurrent neural network, controller, to sequentially produce pairs of indices of which layers to use, and what operations to apply on them. In what follows, we use the term *block* to denote a single layer or a sequence of several layers and the term *branch* – to denote a set of one or two operations; in turn, multiple connected branches give rise to a *cell*.

Connectivity Structure

To form connections inside the decoder part, we i.) first sample with replacement a pair of indices of the encoder layers, ii.) apply the same set of operations (*cell*) on each sampled index, iii.) sum up the outputs (Fig. 5.2), and iv.) add the resultant layer into the sampling pool such that it can be chosen during the next step. In the end, all non-sampled summation outputs are concatenated, before being fed into a single 1×1 convolution to reduce the number of channels followed by the final classification layer.

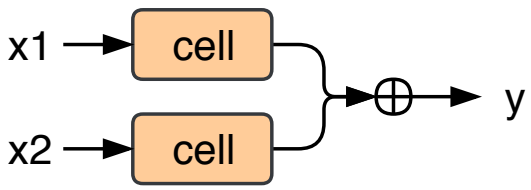


Figure 5.2: Block structure of the decoder. The same cell operation is applied to two different layers specified by the connectivity configuration. If the two layers have different sizes, the smaller one is scaled up via bilinear upsampling to match the larger one.

Cell Structure

The cell design is similarly generated by sampling a set of operations and corresponding indices. Nevertheless, there are several notable differences:

1. The operation at each position can vary;

2. A single operation is applied to the input without any aggregation operator;

3. After that, two indices and two operations are being sampled with replacement, with the corresponding outputs being summed up;

4. The outputs of each operation along with their summation layer are added into the sampling pool and can be sampled during the next step.

An example of the cell structure with its complete search space is illustrated in Fig. 5.3.

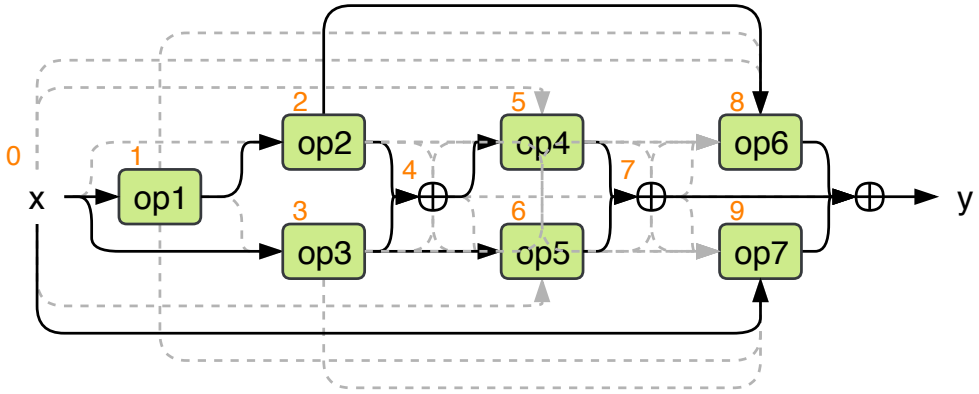


Figure 5.3: Example of the cell structure. Digits on top of the upper left corner of each **operator** are the indices of layers. Except for the layer after the first operator, the output layers of all other operators can be sampled at any subsequent step. The solid black lines indicate the used paths and dashed grey lines are other unused possible paths. The cell configuration for the above cell is [op1, [1, 0, op2, op3], [4, 3, op4, op5], [2, 0, op6, op7]], where the first two values in each block of four (except for *op1*) indicate the chosen layers and the last two – the corresponding operations applied to each layer, *e.g.* the branch [4, 3, op4, op5] is equivalent to *op4(layer4) + op5(layer3)*.

Operations

Based on the existing research in semantic segmentation, we consider 11 operations:

- conv 1×1,
- conv 3×3,
- separable conv 3×3,[2]
- separable conv 5×5,
- global average pooling[3] followed by upsampling and conv 1×1,
- conv 3×3 with dilation rate 3,
- conv 3×3 with dilation rate 12,
- separable conv 3×3 with dilation rate 3,
- separable conv 5×5 with dilation rate 6,
- skip-connection,
- zero-operation that effectively nullifies the path.

We present an example of the search layout with 2 decoder blocks and 2 cell branches in Fig. 5.4.

5.2.4 Search Strategy

We randomly divide the training set into two disjoint sets – meta-train and meta-val. The meta-train subset is used to train the sampled architecture on the given task (i.e., semantic segmentation), whereas meta-val – to evaluate the trained architecture and provide the controller with a scalar reward. Given the sampled sequence, its log-probabilities and the reward signal, the controller is optimised via a popular RL algorithm – proximal policy optimisation (PPO) (Schulman et al., 2017). Hence, there are two training processes present: inner – optimisation of the sampled architecture on the given task, and outer – optimisation of the controller. In other words, the inner loop evaluates how well a chosen architecture can perform

[2] A separable $k \times k$ convolution is a $k \times k$ depthwise convolution (Fig. 4.1) followed by 1×1 standard convolution.

[3] A global pooling performs its operation (*e.g.* average or max) over the whole spatial size of the input.

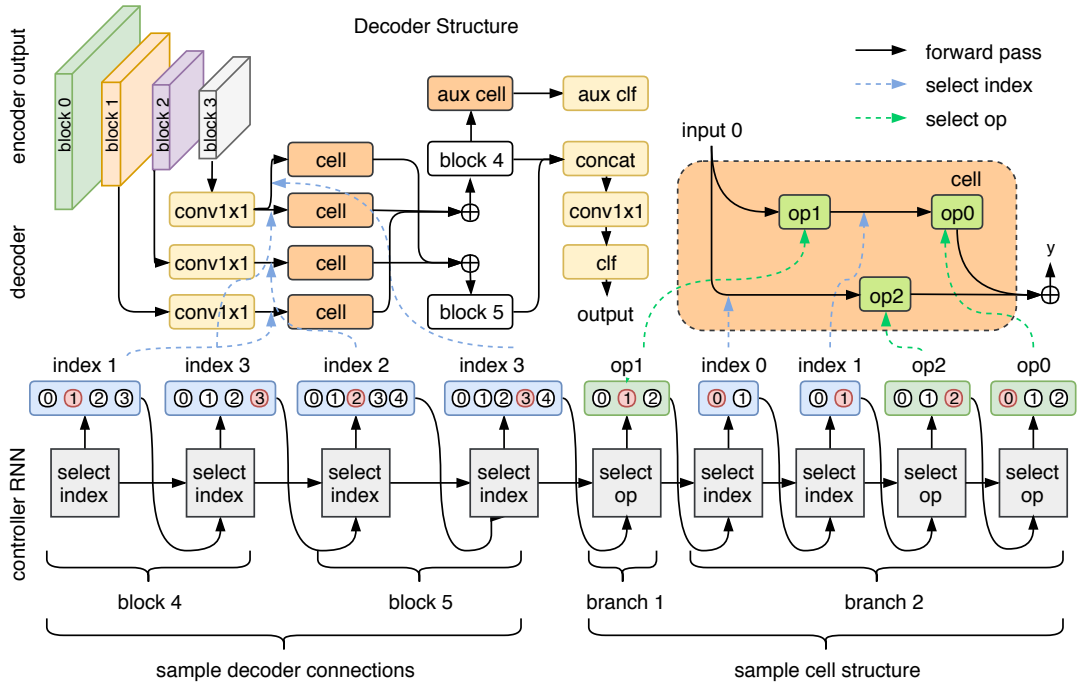


Figure 5.4: Example of the encoder-decoder auxiliary search layout. The controller RNN (*bottom*) first generates connections between the encoder and the decoder (*top left*), and then samples locations and operations to use inside the cell (*top right*). All the cells (including the auxiliary cell) share the emitted design. *clf* stands for *classifier*, and it is a single convolutional layer with the number of output channels equal to the number of output classes. The decisions made by the controller RNN are highlighted in red.
In this example, the controller first samples two indices (*block1* and *block3*), both of which pass through the corresponding cells, before being summed up to create *block4*. The controller then samples *block2* and *block3* that are merged into *block5*. Since *block4* was not sampled, it is concatenated with *block5* and fed into 1×1 convolution followed by the final classifier. The output of *block4* is also passed through an auxiliary cell for intermediate supervision. To emit the cell design, the controller starts by sampling the first operation applied on the cell input (*op1*), followed by sampling of two indices – *index0*, corresponding to the cell input, and *index1* of the output layer after the first operation. Two operations – *op2* and *op0* – are applied on each index, respectively, and their summation serves as the cell output.

by optimising its weights, while the outer loop learns to sample good architectures by optimising the controller via RL. We next concentrate on the inner loop.

Progressive Stages

We divide the inner training process into two stages. During the first stage, the encoder weights are fixed and its outputs are pre-computed with only the decoder being trained. Requiring only few seconds for each architecture, this leads

to a quick adaptation of the decoder weights and a reasonable estimate of the performance of the sampled design.

We exploit a simple heuristic to decide whether to continue training the sampled architecture in the second stage or not. Concretely, we proceed to the second stage if the current reward value is larger than the running mean of rewards seen so far. Otherwise, we terminate the training process with probability $1 - p$; p is initially set to 0.9 and annealed throughout the search. This heuristic filters out unpromising architectures after the first stage, while still encouraging exploration during early iterations akin to the ϵ-greedy strategy often used in the multi-armed bandit problem (Watkins, 1989).

Fast Training via Knowledge Distillation and Weights' Averaging

Semantic segmentation models are notable for requiring many iterations to converge. Partially, this is addressed by initialising the encoder part from a pre-trained classification network. Unfortunately, no such thing exists for the decoder.

Fortunately, though, we can explore several alternatives that provide faster convergence. Besides tailoring our optimisation hyper-parameters, we rely on two more tricks: firstly, we keep track of the running average of the parameters during each stage and apply them before the final validation. This is a so-called *Polyak averaging* (Polyak and Juditsky, 1992), an effective and efficient way of quickly attaining good optima. Secondly, we append an additional l_2-loss term between the logits of the current architecture and a pre-trained teacher network, encouraging the current architecture to mimic the teacher – it is an example of knowledge distillation (Hinton et al., 2014) also used in Chapter 4 to acquire proxy ground truth data. We can either pre-compute the teacher's outputs beforehand or acquire them on-the-fly in case the teacher's computations are negligible.

The combination of both of these approaches allows us to receive a very reliable estimate of the performance of the semantic segmentation model in just few minutes.

Intermediate Supervision via Auxiliary Cells

We further look for ways of easing optimisation during the fast search, as well as during a longer training of semantic segmentation models. Thus, still aligning with the goal of having a compact but accurate model, we aim to find ways of performing steps that are beneficial during training and obsolete during evaluation.

One approach that we consider here is to rely on intermediate supervision where the supervisory signal is present at some middle layers. As visualised in Fig. 5.4, we apply intermediate supervision on the outputs of auxiliary cells after each summation between the pairs of the main cells. The auxiliary cell is identical to the main cell and can either be conditioned to output ground truth directly or to mimic the teacher's network predictions (or the combination of the above two). At the same time, it does not influence the output of the main classifier either during the training or testing and merely provides better gradients for the rest of the network. In the end, the reward per the sampled architecture will still be decided by the output of the main classifier. For simplicity, we only apply the segmentation loss on all the auxiliary outputs.

The motivation behind searching for cells that may also serve as intermediate supervisors stems from ever-growing empirical (and theoretical under certain assumptions) evidence that deep networks benefit from over-parameterisation during training (Soltanolkotabi et al., 2018; Du and Lee, 2018). While auxiliary cells provide an implicit notion of over-parameterisation, we could have explicitly increased the number of channels and then resorted to pruning. The downside of the pruning methods is that they tend to result in unstructured networks often carrying no tangible benefits in terms of the runtime, whereas our solution simply permits omitting unused layers during inference.

5.3 Experiments

We conduct extensive experiments on PASCAL VOC (Everingham et al., 2010; Hariharan et al., 2011). As commonly done, we keep 10% of those images for

validation of the sampled architectures and computation of the reward signal. For the first stage, we pre-compute the encoder outputs on 4000 images and store them in-memory for faster processing.

The controller is a two-layer recurrent LSTM (Hochreiter and Schmidhuber, 1997) neural network with 100 hidden units. All the units are randomly initialised from a uniform distribution. We use PPO (Schulman et al., 2017) with the learning rate of 0.0001 for the controller optimisation.

The encoder part of our network is MobileNet-v2 (Sandler et al., 2018), pretrained on MS COCO (T. Lin et al., 2014) for semantic segmentation using the Light-Weight RefineNet decoder (Chapter 3). We omit the last layers and consider four outputs from layers $2, 3, 6, 8$ as the inputs to the decoder; 1×1 convolutional layers used for the adaptation of the encoder outputs have 48 output channels during search and 64 during training.

The number of times pairs of layers are sampled in the decoder is set to 3, allowing the controller to recover such encoder-decoder architectures as FCN (Long et al., 2015) and RefineNet (G. Lin et al., 2017). To further keep the number of all possible architectures to a feasible amount, we also set to 3 the number of times the locations are sampled inside the cell. All decoder weights are randomly initialised using the Xavier scheme (Glorot and Bengio, 2010). To perform knowledge distillation, we use Light-Weight RefineNet-152 and apply ℓ_2-loss with the coefficient of 0.3 which was set using the grid search. The knowledge distillation outputs are pre-computed for the first stage and omitted during the second one in the interests of time. Polyak averaging is applied with the decay rates of 0.9 and 0.99, correspondingly. Batch normalisation statistics are updated during both stages.

5.3.1 Search Results

For the inner training of the sampled architectures, we devise a fast and stable training strategy: we exploit the Adam learning rule (Kingma and Ba, 2014) for the decoder part of the network, and SGD with momentum – for the encoder. In particular, we use the learning rates of $3e-3$ and $1e-3$, respectively (lower for the

encoder since it is already pre-trained). We pre-train each sampled architecture for 5 epochs on the first stage, and for 1 on the second (in case the stopping criterion is not triggered). As the reward signal, we consider the geometric mean of three quantities: namely,

i.) mean intersection-over-union (mean IoU) (Sect. 2.1), or Jaccard Index (Everingham et al., 2010),

ii.) frequency-weighted IoU, that scales each class IoU by the number of pixels present in that class, and

iii.) mean-pixel accuracy (Sect. 2.1), that averages the number of correct pixels per each class. When computing these metrics, we do not include the background class as it tends to skew the results due to a large number of pixels belonging to the background. As mentioned above, we keep the running mean of rewards after the first stage to decide whether to continue training a sampled architecture.

We visualise the reward progress during both stages in Figure 5.5 and compare it with the random selection of architectures. The quality of the architectures discovered by the RL-based controller grows with time, while the quality of those selected randomly stays nearly the same as expected.

We further evaluate the impact of the inclusion of Polyak averaging, auxiliary cells and knowledge distillation across each training stage. To this end, we randomly sample and train 140 architectures. We visualise the distributions of rewards in Fig. 5.6. All the tested settings significantly outperform the baseline in both stages, and the highest rewards in the second stage are attained when all the above components are present.

5.3.2 Effect of Intermediate Supervision via Auxiliary Cells

At the end of the search process, we select 10 discovered architectures with the highest rewards and proceed by carrying out additional ablation studies aimed to estimate the benefit of the proposed auxiliary scheme.

In particular, we train each architecture for longer: *i.e.* for 20 epochs on BSD together with PASCAL VOC followed by another 30 epochs on PASCAL VOC only.

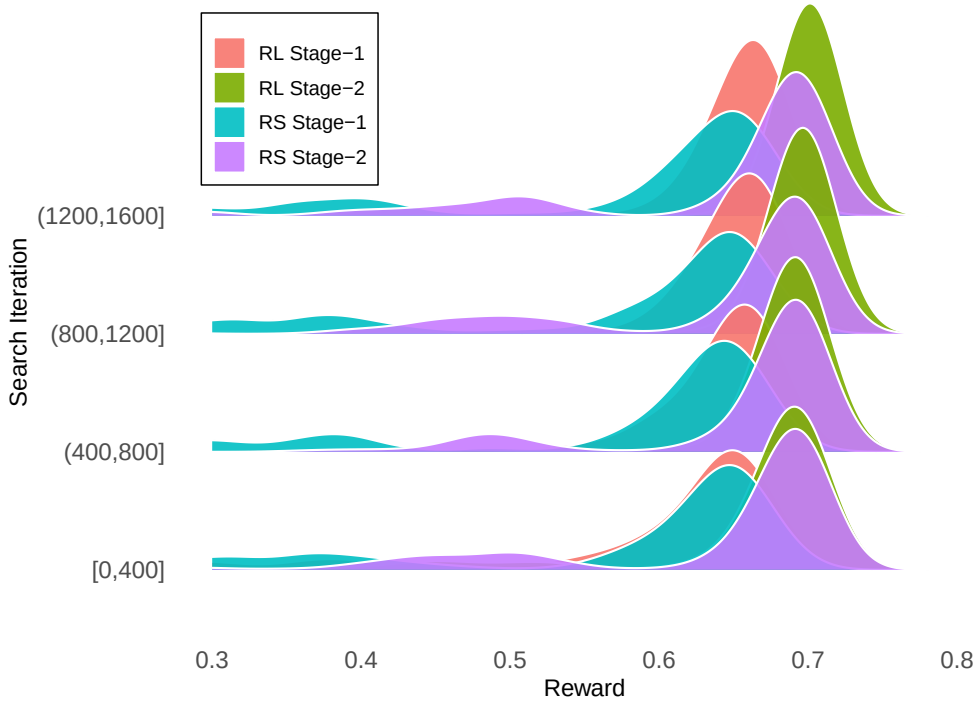


Figure 5.5: Distribution of rewards per each training stage for reinforcement learning (RL) and random search (RS) strategies. Higher peaks correspond to higher density.

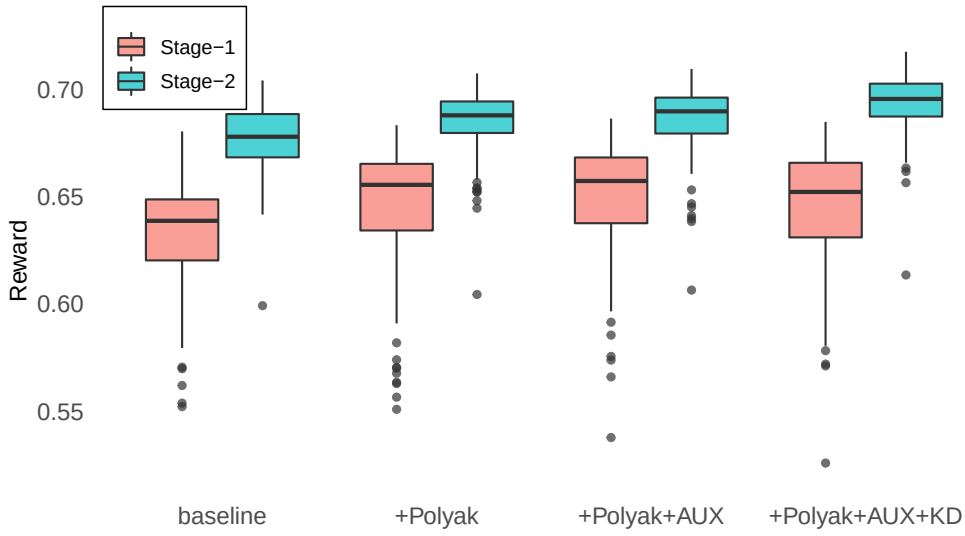


Figure 5.6: Distribution of rewards during each training stage of the search process across setups with Polyak averaging ($Polyak$), intermediate supervision through auxiliary cells (AUX) and knowledge distillation (KD).

For simplicity, we omit Polyak averaging and knowledge distillation. Three distinct setups are being tested: whether intermediate supervision helps at all and whether the auxiliary cell is superior to a plain single-layer classifier.

The results for these ablation studies are in Fig. 5.7, where we compare the effect of intermediate supervision with an auxiliary cell (*cell*) against the supervision with a single layer classifier (*clf*) and no intermediate supervision at all (*none*). Intermediate supervision helps to achieve significantly higher mean IoU values for all the architectures, with the inclusion of the auxiliary cell leading to 8 best results out of 10 and 3 absolute best numbers across all the setups with nearly 73% mean iou.

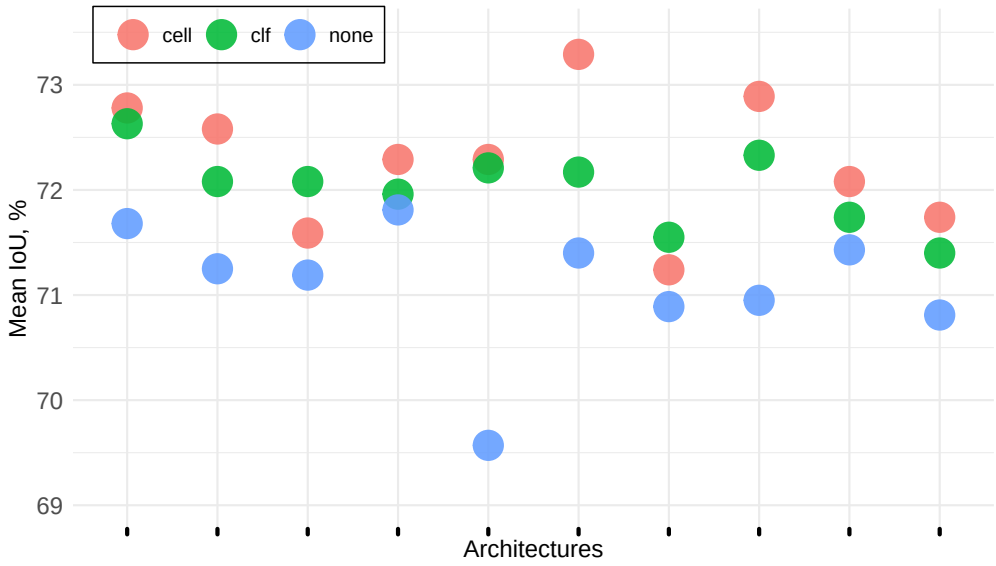
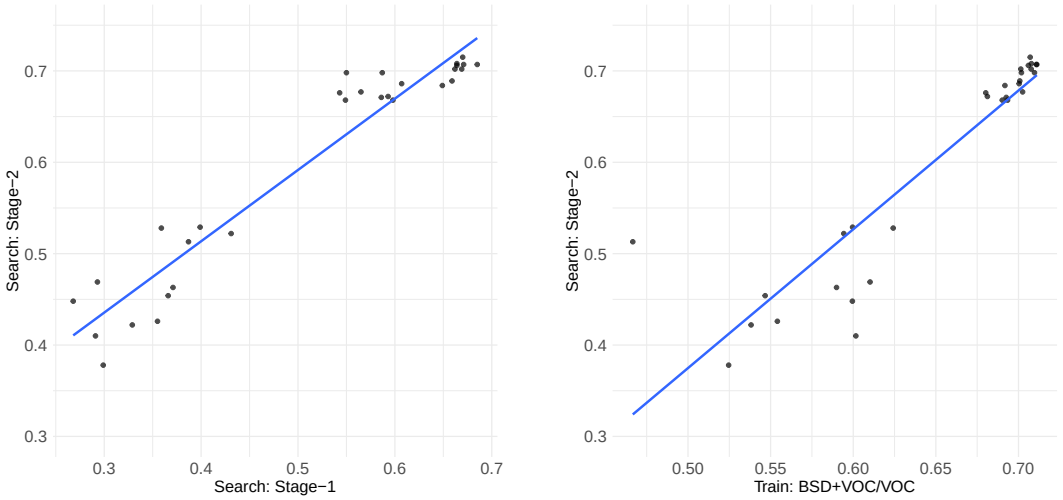


Figure 5.7: Ablation studies on the value of intermediate supervision with an auxiliary cell (*cell*), a single-layer classifier (*clf*), or no intermediate supervision (*none*). Each tick on the x-axis corresponds to a different architecture.

5.3.3 Relation between search rewards and training performance

We further measure the correlation between the rewards acquired during the search and mean IoU attained by the same architectures trained for longer. To this end, we randomly sample 30 architectures out of those explored by the controller: for a fair comparison, we sample 10 architectures with poor search performance (with rewards being less than 0.4), 10 with medium rewards (between 0.4 and 0.6), and 10 with high rewards (> 0.6). We train each architecture on BSD+VOC and VOC as in Sect. 5.3.2, rank each according to its rewards and mean IoU, and measure the Spearman's rank correlation coefficient. As visible in Fig. 5.8, there is a strong positive correlation of

more than 0.92 present between the rewards after each stage, as well as between the final reward and mean IoU. This signals that our search process can reliably differentiate between poor-performing and well-performing architectures.



(a) $\rho = 0.9341$ **(b)** $\rho = 0.9239$

Figure 5.8: Correlation between rewards acquired during search stages ***(a)*** and mean IoU after full training ***(b)*** of 30 architectures on BSD+VOC/VOC.

5.3.4 Full Training Results

Finally, we choose 3 best performing architectures from Sect. 5.3.2 and train each on the full training set, augmented with annotations from MS COCO (T. Lin et al., 2014). The training setup is analogous to the aforementioned one with the first stage being trained for 30 epochs (on COCO+BSD+VOC), the second stage – for 50 (BSD+VOC), and the last one – for 100 (VOC only). After each stage, the learning rates are halved. Additionally, halfway through the last stage we freeze the batch norm statistics and divide the learning rate in half. We exploit intermediate supervision via auxiliary cells with the coefficients of $0.3, 0.25, 0.2, 0.15$ across the stages.

Quantitative results are given in Table 5.1.[4] The architectures discovered by our method achieve competitive performance in comparison to other state-of-the-art compact models (with the increase of 1.1−1.8% in mean iou) and even do so with a

[4]Per-class measures are provided in *Appendix B*.

Model	Val mIoU,%,	MAdds,B	Params,M	Output Res	Runtime,ms (JetsonTX2/1080Ti)	
DeepLab-v3-ASPP (Sandler et al., 2018)	75.7	5.8	4.5	32×32	69.67±0.53	**8.09±0.53**
DeepLab-v3 (Sandler et al., 2018)	75.9	8.73	**2.1**	64×64	122.07±0.58	11.35±0.43
RefineNet-LW	76.2	9.3	3.3	128×128	144.85 ± 0.49	12.00±0.26
arch0	**78.0**	4.47	2.6	128×128	109.36±0.39	14.86±0.31
arch1	77.1	**2.95**	2.8	64×64	67.57±0.54	11.04±0.23
arch2	77.3	3.47	2.9	64×64	**64.60±0.33**	8.86±0.26

Table 5.1: Results on the validation set of PASCAL VOC after full training on COCO+BSD+VOC. All networks share the same backbone - MobileNet-v2. FLOPs and runtime are being measured on 512 × 512 inputs. For DeepLab-v3 we use official models provided by the authors.

significantly lower number of floating point operations for the same output resolution – *e.g.* with 2.95B operations for 64×64 output resolution in *arch1* against 8.73B operations in DeepLab-v3 (Sandler et al., 2018). At the same time, the found architectures can be run in real-time both on a generic GPU card (here 1080Ti) and an embedded device such as JetsonTX2.[5] Qualitatively (Fig. 5.9), our model is able to better recognise similar and easily confused classes (*e.g.* horse – dog in row 3, and cat – dog in row 5), better segment foreground from background and avoid spurious predictions (rows 1,2,4,5).

We visualise the structure of the highest performing architecture (*arch0*) in Fig. 5.10.[6] With multiple branches encoding information of different scales, it resembles several prominent blocks in semantic segmentation, notably the ASPP module (L. Chen et al., 2018c). Importantly, the cell found by our method differs in the way the receptive field size is controlled. Whereas ASPP solely relies on various dilation rates, here convolutions with different kernel sizes arranged in a cascaded manner allow more flexibility. Furthermore, this design is more computationally efficient and has a higher expressiveness as intermediate features are easily re-used.

5.3.5 Transferability to depth estimation

We further apply the top-discovered architectures to the task of depth estimation on the NYUDv2 (Silberman et al., 2012) dataset. As in Chapter 4, we only use 25K training images with depth annotations from the Kinect sensor, and report

[5] Please refer to *Appendix C* for the note regarding the Jetson's runtime.
[6] Other architectures are visualised in *Appendix D*.

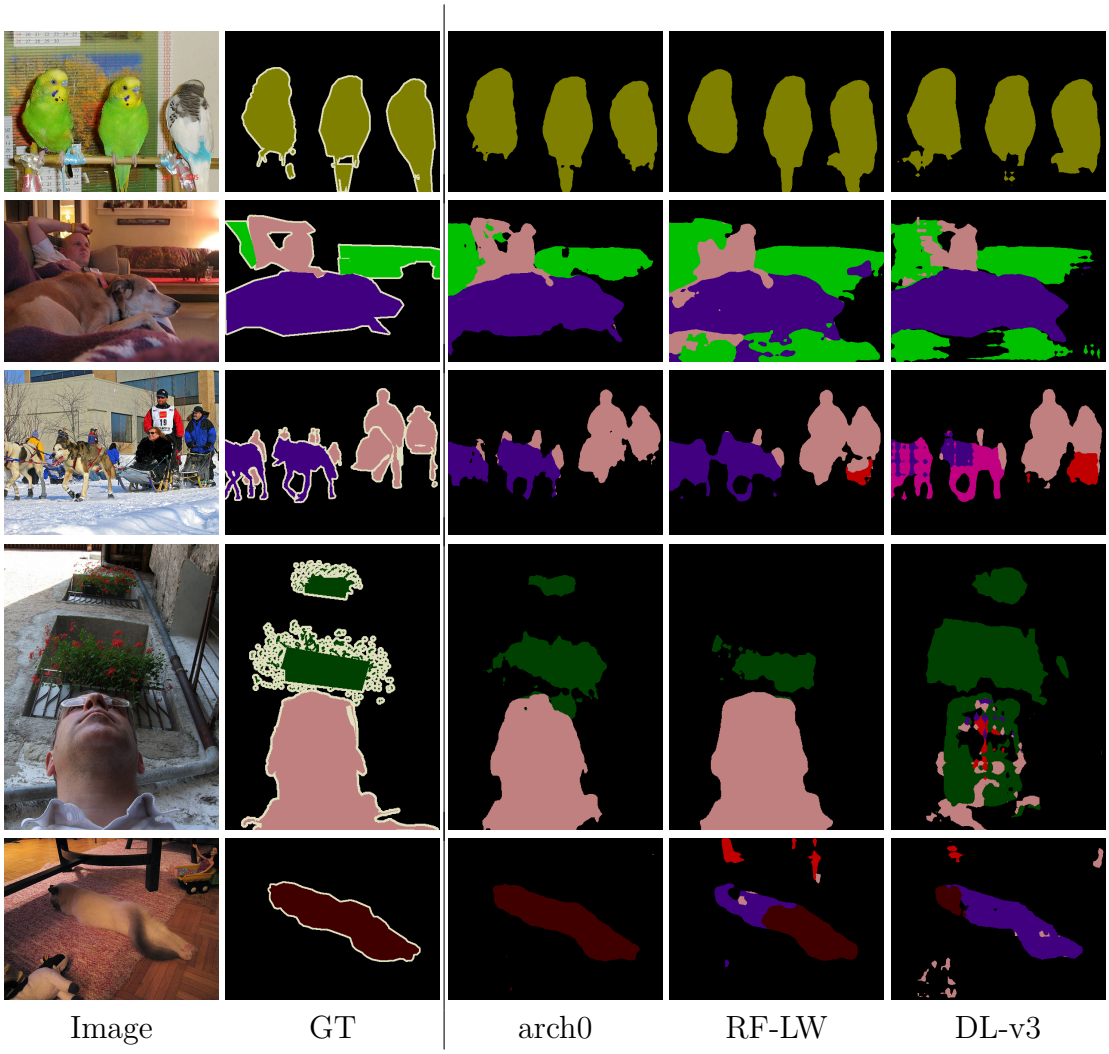


Figure 5.9: Inference results of (*arch0*) on the validation set of PASCAL VOC, together with Light-Weight-RefineNet (*RF-LW*) and DeepLab-v3 (*DL-v3*). All the models rely on MobileNet-v2 as the encoder. 'GT' stands for 'ground truth'.

validation results on 654 images in Table 5.2. Among other compact real-time networks, we achieve significantly better results across all the metrics without any additional tricks and no extra information. In particular, *arch1* attains absolute best results on 6 out 8 metrics, showing the joint best in accuracy at the threshold of 1.25^3 and slightly inferior results in RMSE (lin) – 0.526m against 0.523m of *arch0* and 0.525m of *arch2*. While CReaM (Spek et al., 2018) is still the smallest network with just 1.5M parameters (against 2.6M, 2.8M and 2.9M for each *arch*, respectively), its numbers are significantly lower than those of the discovered architectures. The discovered architectures further outperform the manually designed Multi-Task Light-

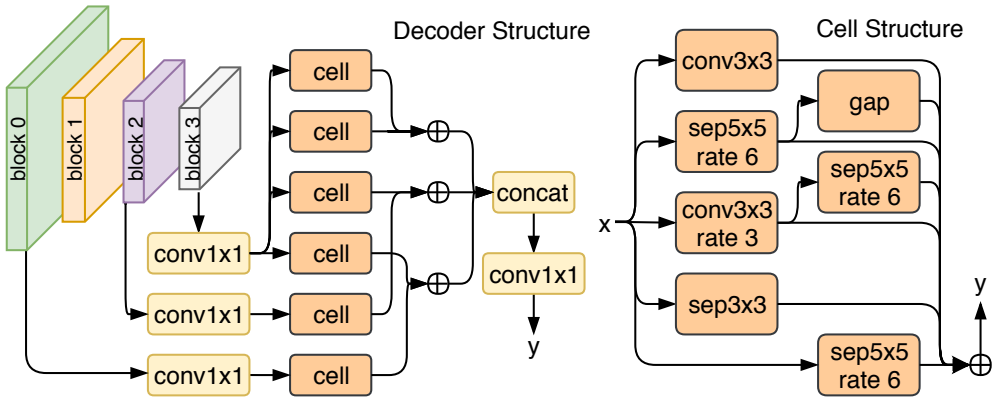


Figure 5.10: Automatically discovered decoder architecture (*arch0*). We visualise the connectivity structure between encoder and decoder (*left*), and the cell design (*right*). $\bigoplus$ represents an element-wise summation operation applied to each branch scaled to the highest spatial resolution among them (via bilinear interpolation), while *'gap'* stands for global average pooling.

Weight RefineNet (Chapter 4), which also used extra information for training in the form of semantic segmentation annotations. A few qualitatives examples of our approach are provided in Appendix B.

Metric / Model	arch0	arch1	arch2	Multi-Task RF-LW	CReaM (Spek et al., 2018)
RMSE (lin)	**0.523**	0.526	0.525	0.565	0.687
RMSE (log)	0.184	**0.183**	0.189	0.205	0.251
abs rel	0.136	**0.131**	0.140	0.149	0.190
sqr rel	0.089	**0.086**	0.093	0.105	–
$\delta < 1.25$	0.830	**0.832**	0.820	0.790	0.704
$\delta < 1.25^2$	0.967	**0.968**	0.966	0.955	0.917
$\delta < 1.25^3$	**0.992**	**0.992**	**0.992**	0.990	0.977
Parameters, M	2.6	2.8	2.9	3.0	**1.5**

Table 5.2: Quantitative results on the validation set of NYUDv2. For RMSE, abs rel and sqr rel lower values are better, whereas for accuracy (δ) higher values are better.

5.4 Conclusion

There is little doubt that a manual design of neural architectures is a tedious and difficult task to handle, especially without significant domain knowledge. While we did rely on the manual design in the previous chapters, we only considered a

limited set of changes that could be applied to an existing network. In contrast, in this chapter we introduced a much larger search space of compact architectures and used reinforcement learning to traverse it. The key component of our approach that enabled an efficient traversal is a quick evaluation of the potential of a given segmentation network. In particular, a combination of Polyak averaging, knowledge distillation, auxiliary intermediate supervision and progressive training led to significantly faster convergence of promising architectures and to the early stopping of unpromising ones. We achieved competitive performance to manually designed state-of-the-art compact architectures on PASCAL VOC while searching only for 4 days on 2 GPU cards. Moreover, the best found architectures also attained excellent results on another dense per-pixel task of depth prediction.

The major limitation of the current approach lies in its reliance on the fully pre-trained encoder that already contains nearly 2M parameters. At the same time, we cannot simply extend the algorithm to also search for the encoder part, as it would require to make an excessively large number of decisions by the controller. Instead of that, in the next chapter we will re-think the definitions of the search space and the cell design, which would allow us to search for high-performing and extremely compact segmentation architectures with less than half a million parameters.

6

Neural Architecture Search of Compact Segmentation Networks

In the last chapter, we presented an automated method of finding real-time and high-performing dense per-pixel networks. We did so by searching for the decoder part given a fully pre-trained encoder. Because of that, the lower bound on the number of parameters was already defined by the size of the encoder with nearly $2M$ weights. In this chapter, we will propose several modifications to the search space that would reduce the number of required pre-trained parameters to just $60K$.

Instead of searching for a single cell design with several sets of operations inside it, we will introduce the notion of a template consisting of just 3 operations. For each generated architecture, we will generate a set of templates and predict which template to apply to each pair of input layers (also predicted). In addition to that, we will predict the number of times the given template must be stacked and whether to spatially downsample feature maps inside the template. All the decisions will once again be made by the reinforcement learning-based controller.

As in the previous chapters, our focus will be on both real-time and high-performing networks, but, unlike the previous chapters, we will aim to discover extremely compact networks with fewer than a million training parameters. Because of that we will consider two semantic segmentation benchmarks with a relatively

low number of diverse training samples – namely, popular urban driving datasets CamVid (Brostow et al., 2009) and CityScapes (Cordts et al., 2016).

The results outlined in this chapter first appeared in the proceedings of the IEEE/PAMI-TC Winter Conference on Applications of Computer Vision (WACV 2020) under the title of 'Template-Based Automatic Search of Compact Semantic Segmentation Architectures' (Nekrasov et al., 2020b).[1]

6.1 Overview

In the previous chapter, we introduced the neural architecture search problem, where the space of thousands of different architectures is traversed with the help of reinforcement learning (Fig. 5.1). On top of that, we built an automated approach for discovering segmentation architectures. Since we relied on the pre-trained classifier in the encoder part, our discovered architectures already contained a large number of parameters.

A naive approach of overcoming this limitation may first include the search of extremely compact classification networks followed by the search of operations specific for semantic segmentation. While potentially working, it would require a significant amount of resources to carry out two instantiations of NAS and might be sub-optimal as the structures found for image classification may still possess redundant operations that are not necessary for semantic segmentation. Another way might include an end-to-end search of segmentation architectures: in such a case our reinforcement learning-based methodology will not fare well as it would require to make an exceedingly large number of sequential decisions.

As a reminder, in the previous chapter we introduced the notion of a *cell*, which is a sequence of several operations arranged in a certain design. The design together with the operations were predicted by the RL-based controller. Importantly, the same cell design was used across various locations in the decoder, and individual combinations of cell operations could not be re-applied elsewhere

[1] The algorithm implementation and pre-trained discovered models are available here: https://github.com/DrSleep/nas-segm-pytorch

in the network. Here instead we will propose a more flexible way of generating sequences of operations. In particular, in place of a single long sequence (*i.e.* cell), we will generate several short sequences (of 3 operations), which we will call *templates*. Each template can be applied at any of the predicted locations and can be stacked multiple times. Concretely, at each decision step, the search algorithm will predict two locations of layers to be used as template inputs, the template to use, the number of repetitions that the template will be used for and the downsampling factor (stride) of the first operations in the template. As we will demonstrate below, such an approach leads to flexible representations of end-to-end architectures for semantic segmentation and reduces the number of decisions to make, hence allowing us to rely only on few layers of a pre-trained classifier.

6.2 Methodology

As the stem of our architecture we consider three initial residual blocks of MobileNet-v2 (Sandler et al., 2018) – in total, they contain only 60K parameters and reduce the spatial resolution to $\frac{1}{8}$. This aggressive downsampling at the beginning is a common feature among most fully convolutional networks. Notably, we will generate a significantly larger portion of the network automatically.

From the stem, we record two outputs with $\frac{1}{4}$ and $\frac{1}{8}$ spatial resolutions of the input, respectively. Given those, for the rest of the network we use a recurrent neural network, controller, to predict actions a_j^i at each step i of block j. We begin from the sequence of actions used in the last chapter (Sect. 5.2.3) that can be written as follows: $A = [loc_1, loc_2, op_1, op_2]$, where loc_i is the layer the operation op_i will be applied to.[2]

6.2.1 Hierarchical template modelling

First of all, we note that the definition of the action sequence above is restrictive: if we were to re-apply the same arrangement of operations somewhere in the network again, we would need to sample it again, too – which would be wasteful. To this end, we separate the sampling process of locations and operations. In

[2]From here on we omit the block indices and use capitals to denote the sequence of tokens.

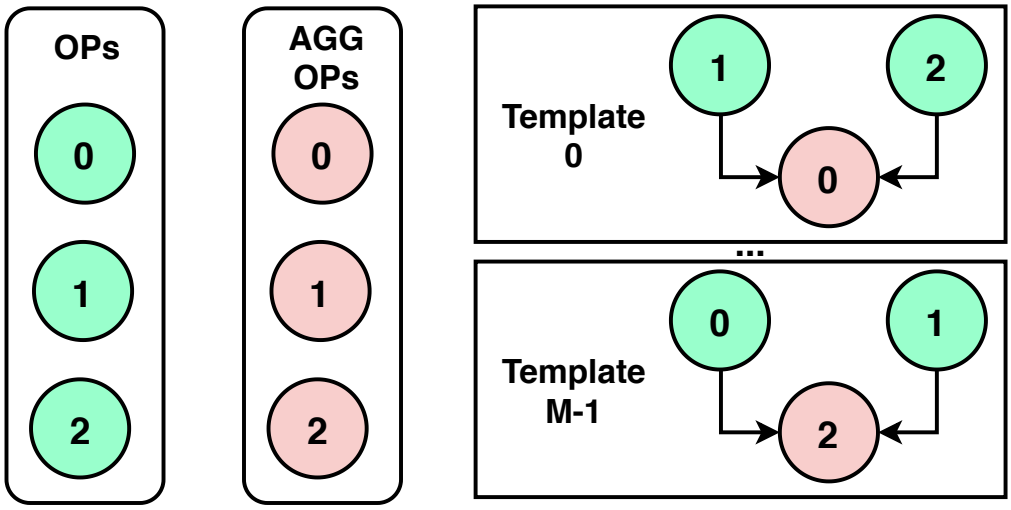


Figure 6.1: We generate M templates each of which takes two inputs and produces one output. The template comprises two individual operations (**OPs**) and one aggregation operation (**AGG OP**).

particular, we generate a set of templates of operations that can be plugged into the network at multiple locations. Inside the template, we also include the aggregation operation op_{agg} (to be either per-pixel summation or channel-wise concatenation). The template takes two inputs and produces one output. Each input undergoes the corresponding operation, and the aggregation operation is used on the intermediate outputs (Fig. 6.1). Thus, any template can be written as follows: $T = [op_1, op_2, op_{agg}]$. In cases where inputs have an unequal number of channels, the output channel dimension of each operation is set to the largest among the inputs.

Having generated the templates, we move on to generating the network structure: in particular, we sample (with replacement) two locations out of the sampling pool (initialised with two stem outputs), and the index of the template (Fig. 6.2) – $A = [loc_1, loc_2, id_T]$. The template's output is added into the sampling pool and the process is repeated multiple times. In the end, we concatenate all non-sampled outputs from the sampling pool, reduce their channel dimension with the help of 1×1 convolution and predict per-pixel labels with a single 3×3 convolutional layer.

The benefit of using templates in this approach is that the number of decisions to be made increases slowly with the number of blocks: *e.g.* for N blocks and M templates the length of the sequence that describes an architecture would be equal to

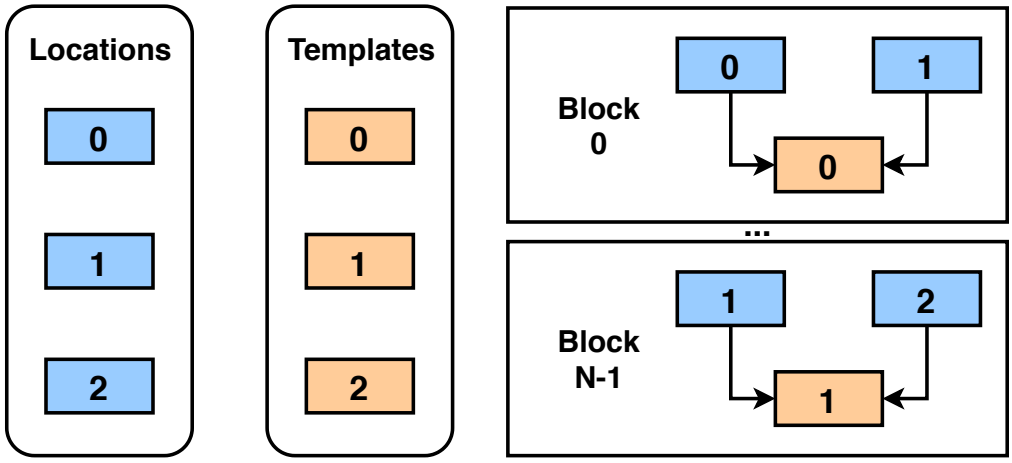


Figure 6.2: We generate N blocks by sampling two locations and one template applied to them.

$(2+3) \cdot N$ for the baseline solution introduced in the previous chapter (two locations and three operations), and to $(2+1) \cdot N + 3 \cdot M$ for the template one (two locations and one template index together with three operations per each template). Hence, having the number of templates lower than 2/3 of the number of blocks would require fewer decisions for the same network depth. At the same time, the template modelling would allow the controller to efficiently re-use already predicted designs.

6.2.2 Increasing the number of templates

While the template modelling has its benefits, it still possesses one significant disadvantage: in order to generate a deeper network, we can only increase the number of blocks, which, in turn, would lead to significantly more decisions to be made and would be no different from the way of scaling up the approach from Chapter 5. Hence, we propose to include an additional parameter in the structure generator at the cost of having N more decisions to be made. Concretely, we generate k – the number of times the template must be repeated[3]; accordingly, the action sequence then becomes $A = [loc_1, loc_2, id_T, k]$. While we could have also abstracted away k into the template definition to reduce the number of decisions, we chose not to because at different blocks it might be preferable to vary k.

[3]To keep the number of parameters of the generated architectures low we will consider k to take values from 1 to 4 in the experiments.

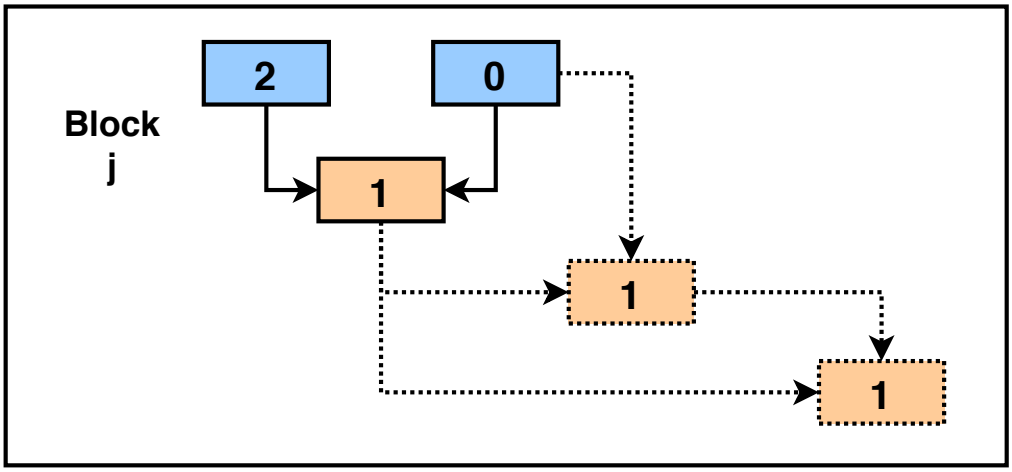


Figure 6.3: A template can be recursively applied multiple times (with non-shared weights): the output of the previous template becomes the first input to the current one, and the final output is considered as the block's output. New instantiations of the template together with connections are depicted with dotted lines.

The process of repeating the template is as follows: after the template is applied on two sampled locations and the template's output is recorded, the second input and the template's output are considered as another two inputs to a new instantiation of the same template with non-shared weights (Fig. 6.3). This is repeated k times, and the last output is appended to the sampling pool.

6.2.3 Adding strides

Up until now we did not make any assumptions about the strides of the operations (apart from the fixed stem block). Nevertheless, in order to generate a complete segmentation architecture it is important to consider the downsampling factors. Keeping all consecutive operations at a constant stride would not fare well as the common wisdom suggests that in order to extract better features, we need to keep reducing the spatial dimensions while increasing the number of channels until a certain point.

Taking this into account, we append an additional decision to make in the sequence definition: the stride prediction – either 1 or 2. If the stride is 2, we decrease spatial dimensions and multiply the channel dimension by a predefined constant that would allow us to control the compactness of the generated network if

needed. We only predict strides for the first half of all the blocks and assume that the rest of them uses stride 1. Analogously, to deal with the inputs of varying resolutions when applying an aggregation operation, we downsample the inputs to the lowest resolution among them for the first half and upsample to the largest resolution – for the rest. This is similar to the encoder-decoder architecture design that fares well in semantic segmentation. The stride prediction is taken out of the template definition to allow the same template to be used with varying strides at different locations. Likewise, it is straightforward to add the prediction of dilation rates per template to recover the dilated decoder approach (L. Chen et al., 2018b) if needed.

Our final string describing a complete architecture can be written as follows: $[loc_1, loc_2, id_T, k, s]\times N$, where N is the number of blocks and $\{T^i = [op_1^i, op_2^i, op_{agg}^i]\}_{i=1}^M$ are M templates. As can be easily seen, we decomposed the original decision sequence into multiple reusable components.

6.3 Search Experiments

6.3.1 NAS setup

As the stem of the network, we use three first blocks of MobileNet-v2 (Sandler et al., 2018) pre-trained on ImageNet (Deng et al., 2009); two outputs from the second and third blocks with 24 and 32 channels and the spatial resolution of $\frac{1}{4}$ and $\frac{1}{8}$ from the original size, respectively, are added into the initial sampling pool. We double the number of channels after each spatial downsampling operation in the discovered structure. In the beginning, all the layers in the sampling pool are transformed to have 48 channels with the help of 1×1 convolution. The pre-classifier layer has the same number of 48 channels.

To keep the number of potential architectures to discover at a reasonable level, we set the number of templates and the number of layers to 3 and 7, correspondingly. The maximum number of times the template can be used sequentially is set to 4.

As the search dataset, we consider the training split of CityScapes (Cordts et al., 2016) with 2975 images randomly divided into meta-train (2677, or 90%) and meta-val (298). We resize all the images to have a longer side of 1024 and train on

square crops of 321×321. Each sampled architecture is trained for 10 epochs and validated twice every 5 epochs. For the first five epochs, we pre-compute the stem outputs and only train the generated part of the network; for the second five, the whole network is trained end-to-end. As the reward we employ the geometric mean of mean iou, mean accuracy and frequency-weighted iou as done in the previous chapter (Sect. 5.3.1). To speed up the convergence of sampled architectures, we rely on Adam with the learning rate of $7e{-}3$, used on mini-batches of 32 examples.

We consider the following set of operations

- separable conv 3×3,
- separable conv 5×5,
- global average pooling followed by upsampling and conv 1×1,
- max-pool 3×3,
- separable conv 5×5 with dilation rate 6,
- skip-connection.

Two aggregation operations that we rely upon are summation and channel-wise concatenation.

To train the controller, we use PPO (Schulman et al., 2017) with the learning rate of 0.0001. In total, we sample, train and evaluate over 2000 architectures in 8 days using two 1080Ti GPUs.

6.3.2 Analysis of Search Results

Does the controller discover better architectures with time?

We first study the progress of rewards through time. The median reward is steadily increasing with more epochs as can be inferred from Fig. 6.4. While most architectures are tightly clustered together, there are several notable outliers on the far right with the reward of near 0.55 that we will explore in the full training experiments in Sect. 6.4.

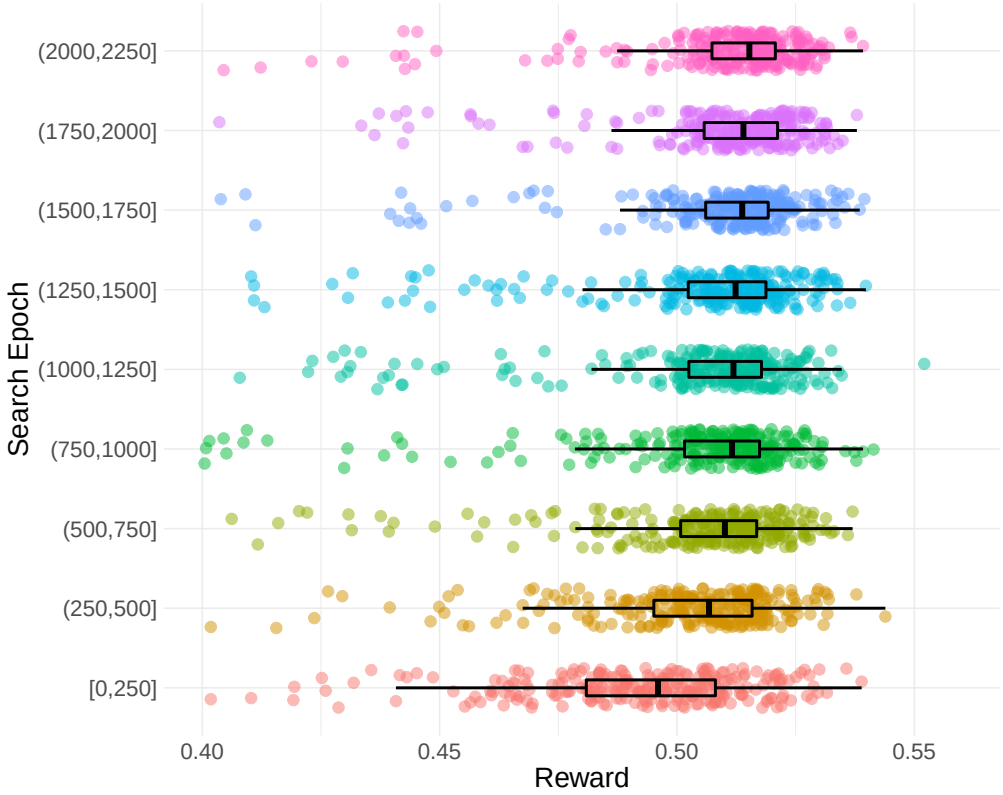


Figure 6.4: Distribution of rewards attained by architectures sampled by the controller. For compactness of the plot, we only visualise rewards greater than or equal to 0.40.

What is the controller policy with respect to downsampling?

We also consider how the resolution of the architecture is related to its reward score. To this end, we first analyse how often the controller chose to downsample[4]. Considering that all outcomes are equally likely in the beginning, one would expect the extreme resolutions of 1 and $\frac{1}{8}$ to be less present. As seen in Fig. 6.5, it is indeed what happens with the controller at the start of the training. Nevertheless, by the end of the search process, the controller becomes far too conservative with regards to downsampling and is more likely to pass on actions that reduce the spatial dimensions. This raises the question of **how the downsampling factor in the generated architecture relates to the reward?**

In an attempt to answer that question, we visualise the reward distribution of

[4]With 7 blocks the maximum number of times downsampling can be performed is $\lfloor 7/2 \rfloor = 3$. Note that here we do not take into account the resolution of two stem layers ($\frac{1}{4}$ and $\frac{1}{8}$, respectively).

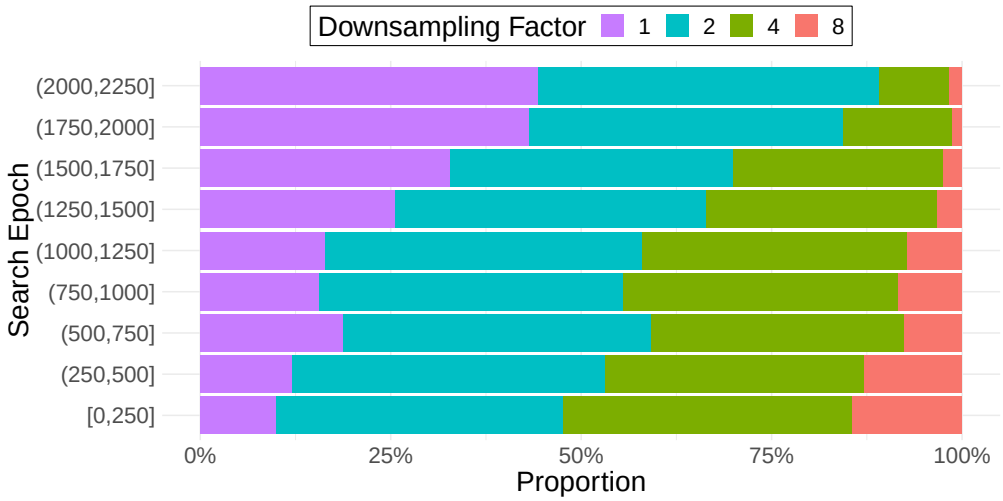


Figure 6.5: Proportion of downsampling factors through time. The minimum downsampling of 1 happens when the controller chooses to use stride=1 everywhere; the maximum downsampling of 8 happens when the controller uses stride=2 three times (hence, $2^3 = 8$) Note that here we do not take into account the resolution of two stem layers ($\frac{1}{4}$ and $\frac{1}{8}$, respectively).

architectures with different downsampling factors in Fig. 6.6. As the plot implies, the rewards for architectures with a higher resolution output tend to be larger, hence, the controller becomes biased towards sampling fewer downsampling actions.

More parameters = better performance?

We further consider the distribution of rewards based on the number of parameters (Fig. 6.7). From it, the size of the architecture appears to have no connection with its reward. A more detailed plot though tells a different story (Fig. 6.8): while for small architectures ($\leq$ 250K) the rewards are almost identically distributed, a negative trend is seen when the number of parameters is growing. It is possible that this effect occurs due to the utilised training strategy favouring compact architectures.

What rewards does each template achieve?

We remind the reader that during the search process the controller samples 3 template structures that can be applied at any of 7 blocks recursively 1−4 times. Since each template consists of two individual operations and one aggregation operation, with 6 unique operations and 2 unique aggregation operations, the total

Figure 6.6: Distribution of rewards attained by architectures sampled by the controller with varying downsampling factors. For compactness of the plot, we only visualise rewards greater than or equal to 0.40. Note that here we do not take into account the resolution of two stem layers ($\frac{1}{4}$ and $\frac{1}{8}$, respectively).

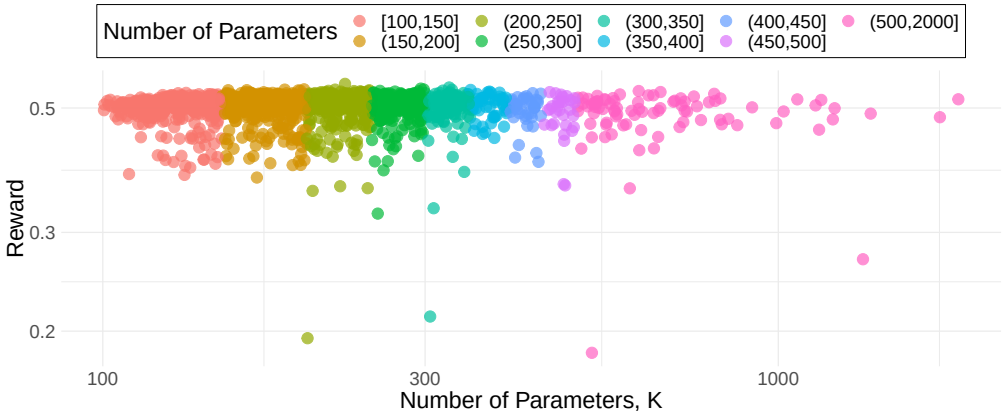


Figure 6.7: Reward as a function of the size of the architectures.

number of unique templates is $C^2_{6+1} = \frac{7!}{5!2!} = 42$ after taking into account the symmetry in the order of individual operations.

We consider a particular template to be sampled during the search process, if its structure was sampled and it was chosen by the controller at least once. We visualise the distribution of rewards for each of 42 templates sampled during the search process in Fig. 6.9 – note that several templates can share the same reward as they might belong to a single architecture. Overall, around half of the templates steadily achieve rewards higher than 0.5.

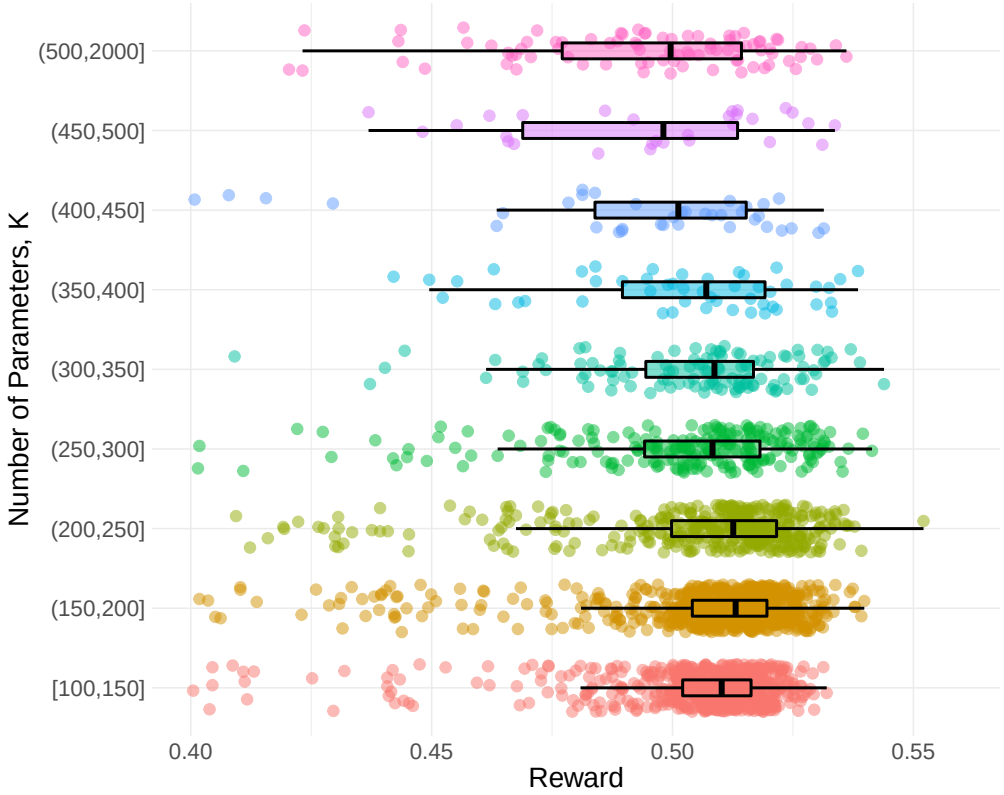


Figure 6.8: Distribution of rewards attained by architectures with varying size. For compactness of the plot, we only visualise rewards greater than or equal to 0.40.

We further depict each of top-5 templates with the highest rewards in Fig. 6.10. Interestingly, all top-performing templates rely on separable 5×5 convolution, and some of them differ only in the aggregation operation used (*e.g. Template 0* and *Template 4*, or *Template 1* and *Template 3*).

6.3.3 Comparison with Random Search

As in the previous chapter (Sect. 5.3.1), an important question to ask with NAS is whether the trained controller performs reliably better than a naive baseline – *e.g.* random search. Relevant to that is the question of whether the ranking of architectures based on rewards achieved during the search is well-correlated with the ranking of architectures based on their scores during a longer training.

To answer these questions, we perform experiments similar to those in the last chapter: concretely, we sample two sets with 20 architectures each – the first

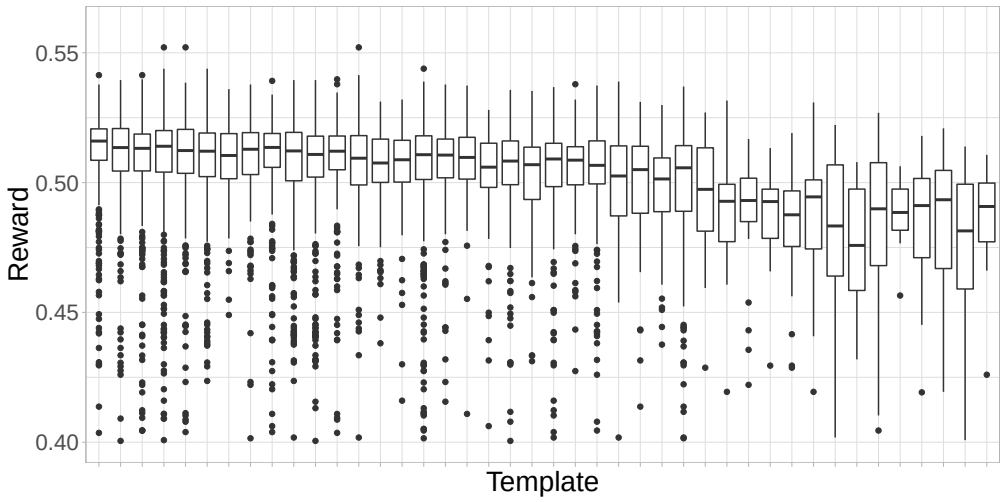


Figure 6.9: Distribution of rewards for each of 42 unique templates. For compactness of the plot, we only visualise rewards greater than or equal to 0.40.

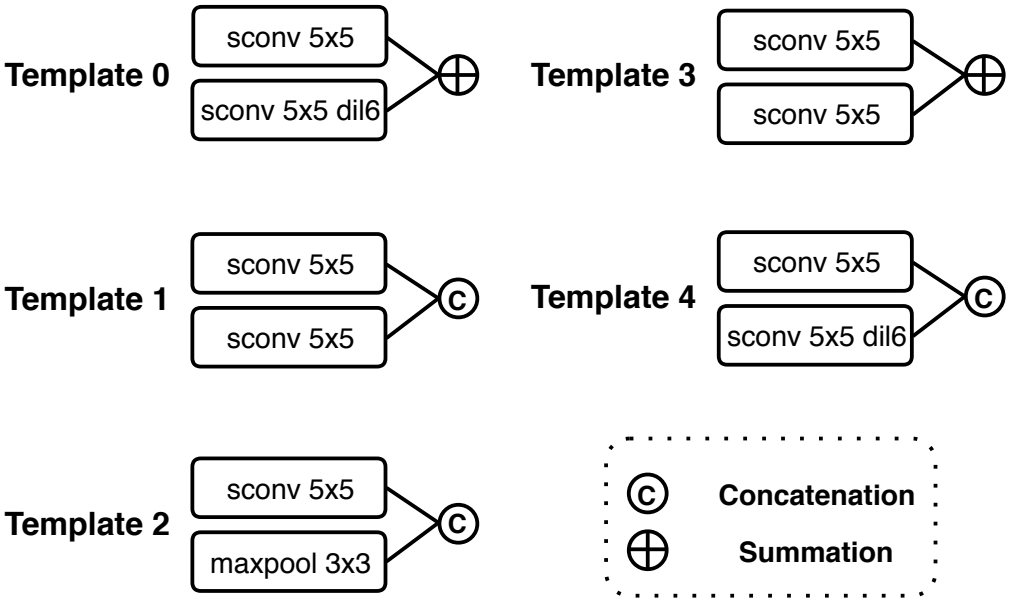


Figure 6.10: Top-5 templates with the highest average reward. Each template takes 2 inputs, applies 2 corresponding operations and produces a single output via an aggregation operator (concatenation or summation).

set is coming from the pre-trained controller, while the second set is sampled randomly. Each set is trained and evaluated on two setups – the search one as described in Sect. 6.3.1 and the one where the training continues for more epochs (we increase it to 20 for the second stage).

We visualise the performance results in Fig. 6.11. The architectures sampled by

the trained controller reliably achieve higher rewards – both within the search and longer training setups.

Figure 6.11: Rewards distribution of 40 architectures, 20 of which are sampled by the trained controller and 20 by random search.

We further plot corresponding rankings in Fig. 6.12. As evident from the plot, when trained for longer the architectures tend to be ranked similarly to the order attained during the search process. In particular, high values of Spearman's rank correlation – $\rho = 0.734$ for the controller, and $\rho = 0.904$ for random search – indicate that the rewards achieved during the searching process may serve as a reliable estimate of the architecture's potential.

6.4 Training Experiments

6.4.1 Setup

To evaluate the best discovered architectures we consider two common urban driving benchmarks for semantic segmentation – CityScapes (Cordts et al., 2016) and CamVid (Brostow et al., 2009). Reliably solving both of them is important for such a practical application as autonomous driving, and our choice is further motivated by the fact that the majority of hand-designed compact networks is being extensively tested on these two datasets.

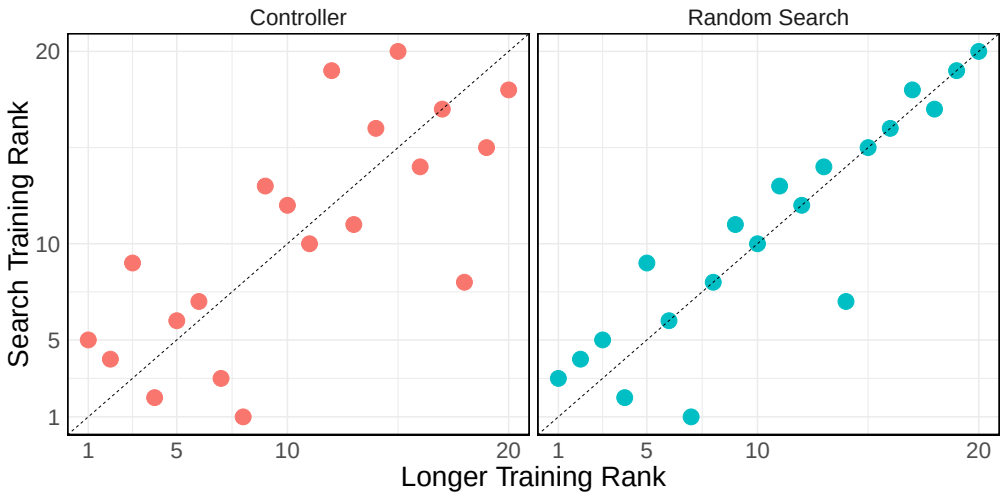


Figure 6.12: Ranks of architectures based on rewards during the longer training setup (x-axis) and the search training setup (y-axis). All the architectures on the left are sampled by the RL-based controller, while all the architectures on the right are randomly sampled.

The training setup slightly differs from the searching one: in particular, we use the 'poly' training schedule (L. Chen et al., 2018b) – $lr_{init} \cdot (1 - \frac{epoch}{n_{epochs}})^{0.9}$ – with SGD with the initial learning rate of $5e{-}2$ and the momentum value of 0.9. The weight decay is set to $1e{-}5$, the initial number of channels to 64.

6.4.2 CityScapes

We train on the full training split and test on the validation split of 500 images. Here we use a square crop size of 769 and train for 1000 epochs with mini-batches of 6 examples.

As given in Table 6.1, two of our automatically discovered models achieve 67.7% and 67.8% mean iou, respectively, with both having less than 300K trainable parameters. We outperform all other compact methods with the exclusion of ERFNet (Romera et al., 2018) that comprises 7× more parameters than any of our models, and BiSeNet (C. Yu et al., 2018) – with more than 20× more parameters. We overcome ICNet (Zhao et al., 2018) on the validation set, but fall behind on the test set as their method was further trained on the validation set, too. Furthermore, we significantly surpass the results of ESPNet (Mehta et al.,

2018) and ESPNet-v2 (Mehta et al., 2019) by more than 5.4% in both cases while requiring considerably fewer parameters.

Method	val miou,%	test miou,%	Params,M
ENet (Paszke et al., 2016)	-	58.3	0.37
ESPNet (Mehta et al., 2018)	61.4	60.3	0.36
ESPNet-v2 (Mehta et al., 2019)	62.7	62.1	0.72
ICNet (Zhao et al., 2018)	67.7	69.5	6.7
ERFNet (C. Yu et al., 2018)	**72.0**	**71.4**	5.8
arch0	68.1	67.7[5]	**0.28**
arch1	69.5	67.8[6]	**0.27**

Table 6.1: Quantitative results on the validation and test sets of CityScapes among compact models (<10M parameters). Note that opposed to what is commonly done, for simplicity we did not train our models on the val set and did not use any post-processing for test evaluation.

We plot the test mean iou as a function of the number of parameters across other popular methods in Fig. 6.13. The discovered architectures belong to the Pareto front as they attain the highest performance with the fewest number of parameters. We further visualise qualitative results in Fig. 6.14. Both architectures are able to correctly segment most parts of the scenes and even identify thin structures such as traffic lights and poles (rows 1−2). Nevertheless, they tend to misclassify large objects, such as trains (row 3).

6.4.3 CamVid

CamVid (Brostow et al., 2009) is another outdoor urban driving dataset that contains 367 images for training and 233 – for testing with 11 semantic classes and resolution of 480×360. We train for 1000 epochs with 10 examples in mini-batch.

[5]Link to test results: https://bit.ly/2HItlwm

[6]Link to test results: https://bit.ly/2FlAfEW

[7]ENet (Paszke et al., 2016), ESPNet (Mehta et al., 2018), ESPNet-v2 (Mehta et al., 2019), SegNet (Badrinarayanan et al., 2015), FRRN (Pohlen et al., 2017), ERFNet (Romera et al., 2018), ICNet (Zhao et al., 2018), FCN-8s (Long et al., 2015), Dilation8 (F. Yu and Koltun, 2016), BiSeNet (C. Yu et al., 2018), PSPNet (Zhao et al., 2017), RefinetNet (G. Lin et al., 2017), DeepLab-v3+ (L. Chen et al., 2018c)

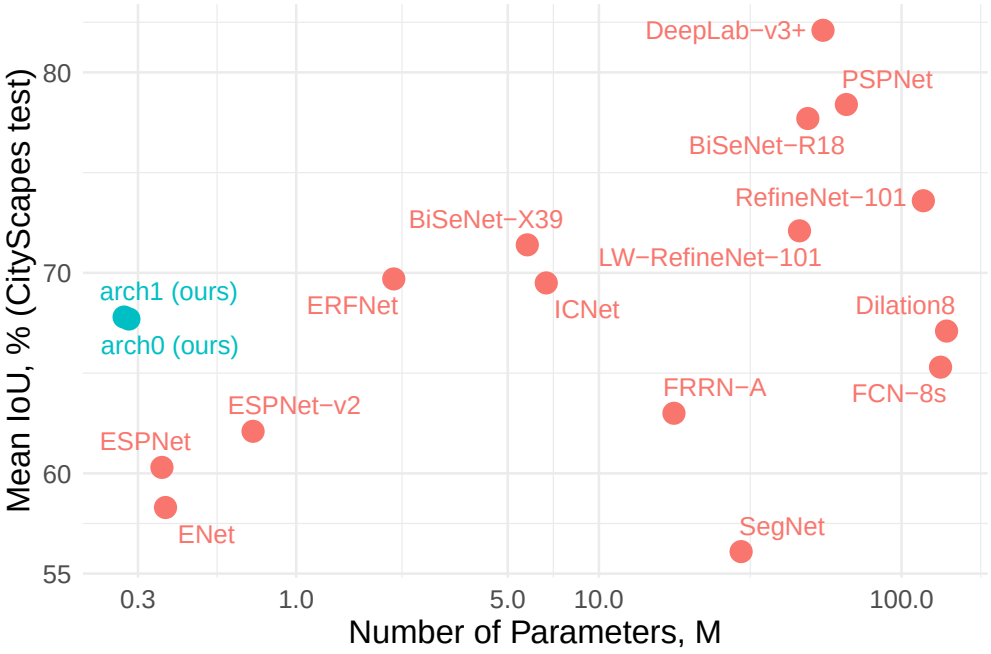


Figure 6.13: Comparison of our networks to other methods[7] on the test set of CityScapes (Cordts et al., 2016) with respect to the number of parameters and mean iou.

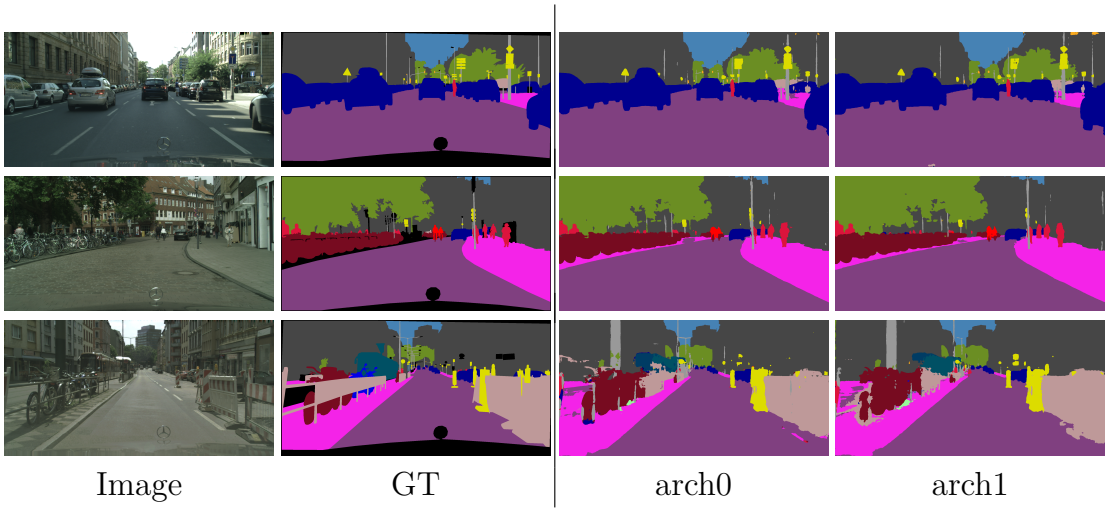


Figure 6.14: Qualitative results of the discovered models – (*arch0* and *arch1*) – on the validation set of CityScapes. The last row shows failure cases. 'GT' stands for 'ground truth'.

The architectures discovered by our method attain mean iou values of 63.9% and 63.2%, respectively, once again exceeding the majority of other compact and even larger models. We outperform both SegNet (Badrinarayanan et al., 2015) and ESPNet (Mehta et al., 2018) by more than 7%, DeepLab-LFOV (L. Chen

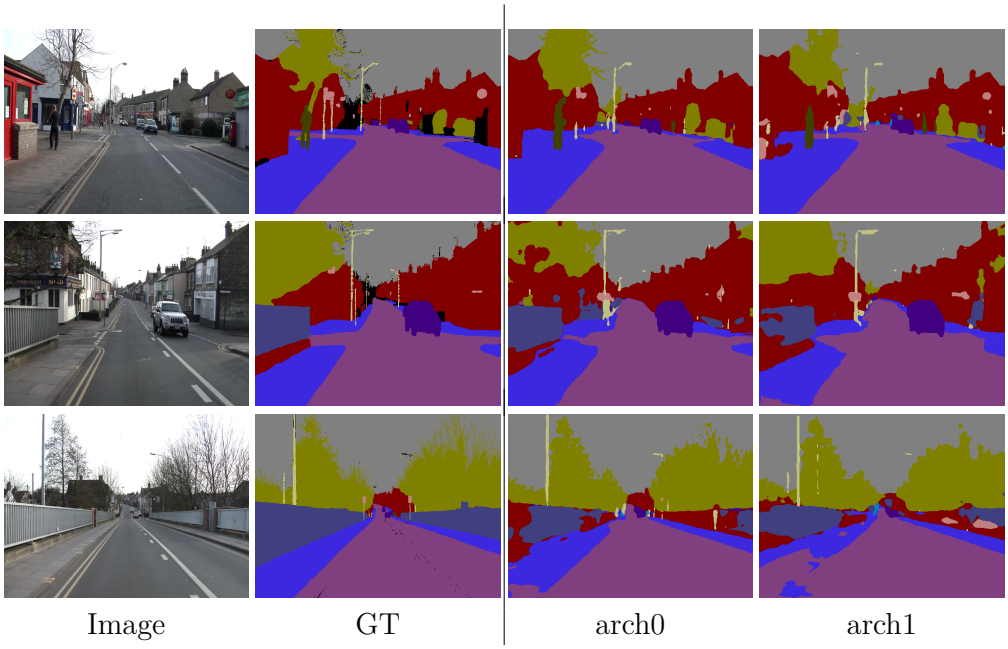


Figure 6.15: Qualitative results of the discovered models - (*arch0* and *arch1*) - on the test set of CamVid. Last row includes failure cases. 'GT' stands for 'ground truth'.

et al., 2018b) – by more than 1.6%, and fall behind BiSeNet (C. Yu et al., 2018) and ICNet (Zhao et al., 2018), both of which exploited higher resolution images of 960×720.

Method	miou,%	Params,M
SegNet (Badrinarayanan et al., 2015)	55.6	29.7
ESPNet (Mehta et al., 2018)	55.6	0.36
DeepLab-LFOV (L. Chen et al., 2018b)	61.6	37.3
†BiSeNet (C. Yu et al., 2018)	**65.6**	5.8
†ICNet (Zhao et al., 2018)	**67.1**	6.7
Ours (arch0)	63.9	**0.28**
Ours (arch1)	63.2	**0.26**

Table 6.2: Quantitative results on the test set of CamVid. (†) means that 960×720 images were used opposed to 480×360.

With the lower quality and resolution of annotations, the predictions are no longer sharp as on CityScapes (Fig. 6.15). Overall, both architectures capture the semantics of the scenes well (rows 1−2) and only fail significantly at delineating long chunks of the same class on the edges of the image (row 3).

6.4.4 Architecture Characteristics

We provide quantitative details of the trained architectures in Table 6.3. Two models possess similar qualities – they are light-weight, both in terms of parameters and the size on disk, and have output resolution of $\frac{1}{4}$. Interestingly, even though *arch0* is slightly larger than *arch1*, it is still faster with almost 20FPS on high-resolution 2048×1024 images. This likely happens because *arch1* uses high-resolution *stem0* more often than *arch0* – four times against three.

Characteristic	arch0	arch1
Parameters,K	280.15	**268.24**
Latency, ms (2048×1024 inputs)	**52.25±0.03**	97.11±0.24
Latency, ms (480×360 inputs)	**8.97±0.10**	11.51±0.14
Output Resolution	1/4	1/4
Size on disk, MB	1.46	**1.41**

Table 6.3: Quantitative characteristics of discovered architectures. All the numbers are measured using a single 1080Ti GPU.

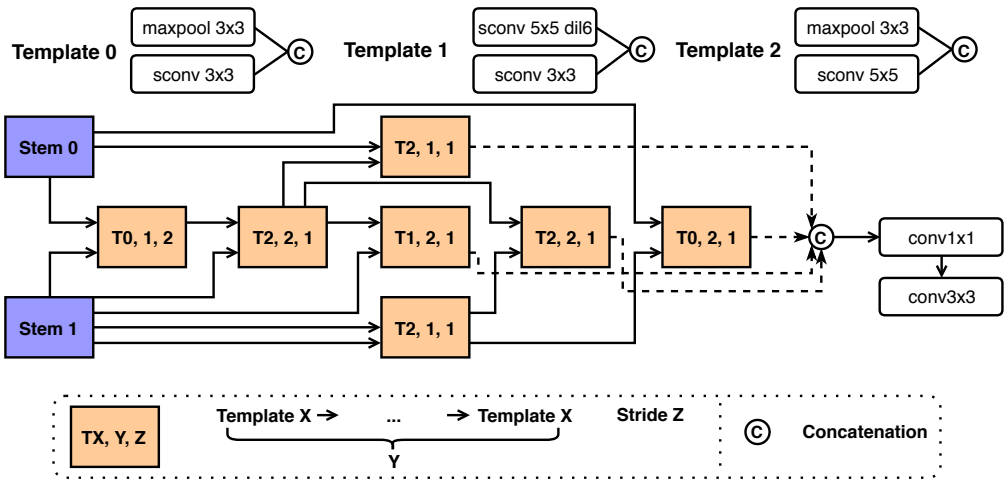


Figure 6.16: Depiction of *arch0*. We visualise the sampled templates at the top of the figure. Each non-stem block, **TX, Y, Z** is the template **X** (*either 0, 1, or 2*) applied **Y** times (*either 1, 2, 3, or 4*) with the stride **Z** in the first operation (*either 1 or 2*).

We further visualise *arch0* in Fig. 6.16 and *arch1* in Fig. 6.17. Notably, all the templates in *arch0* rely on concatenation with the generated structure having multiple connections between intermediate blocks that allows the network to

simultaneously operate on high-level and low-level features; *arch1* instead fully relies on the summation as its aggregation operation and tends to duplicate the templates more often, which, thanks to the light-weight layers in each template, does not lead to a significant growth in the number of parameters.

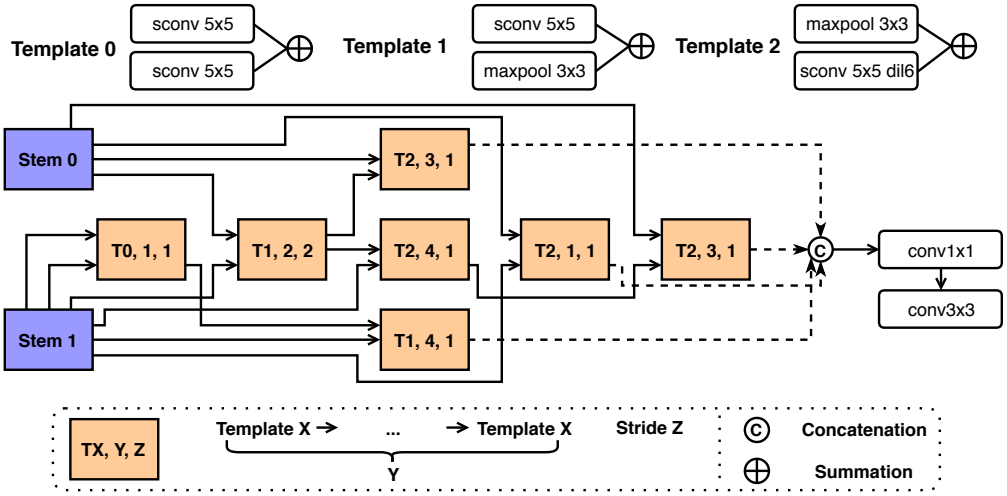


Figure 6.17: Depiction of *arch1*. We visualise the sampled templates in the top of the figure. Each non-stem block, **TX, Y, Z** is the template **X** (*either 0, 1, or 2*) applied **Y** times (*either 1, 2, 3, or 4*) with the stride **Z** in the first operation (*either 1 or 2*).

6.5 Conclusion

In this chapter we mitigated one of the limitations of the neural architecture search approach from Chapter 5 – namely, its reliance on the fully pre-trained encoder network. Here, we overcame this issue by decomposing the sequence of decisions to make into templates which are sets of operations that can be applied anywhere in the network. With that we also predicted how many times the template must be repeated and what downsampling factor to use inside the template. As the results of those changes, we were able to discover extremely compact and highly competitive networks with less than 300K parameters, achieving 68% mean IoU on the test set of CityScapes and 63−64% mean IoU on the test set of CamVid.

Overall, we devoted the last two chapters to demonstrate the value and simplicity of setting up an automated approach to find compact and accurate networks using only a handful of GPU-cards. In contrast to the manual design that we practised in Chapters 3 and 4, it allowed us to consider a much larger set of possible architectures, which would have been difficult to traverse manually. Nevertheless, the automatic search is not a silver bullet, as it does require spending significant efforts on manually designing the search space and manually designing efficient ways of traversing that space. Thus, it is important to keep this in mind when approaching a new problem and deciding whether to rely on your own expertise in adapting an existing architecture or to spend time on designing a search space for an automatic discovery of high-performing networks.

7 Conclusion

We started our journey towards building practical dense per-pixel systems by manually adapting an existing large and slow segmentation network into a smaller and faster architecture with nearly equal performance (Chapter 3). Since solving a single task might not be enough in certain applications, we further extended that approach to support multi-task learning (Chapter 4) and to work equally well with depth prediction and surface normals estimation. We then shifted gears and concentrated on the automatic search of compact and high-performing architectures (Chapters 5 and 6). In particular, we first searched only for the decoder part given a fully-pretrained encoder (Chapter 5) and then amended the search space in a way that permitted discovering extremely compact networks using only a handful of pre-trained encoder layers (Chapter 6).

Along this way, we made the following concrete contributions:

- in Chapter 3 we analysed computationally expensive parts of RefineNet (G. Lin et al., 2017) and proposed a light-weight version of it – *Light-Weight RefineNet* – with more than twofold decrease in runtime and in the number of parameters, while nearly equal performance across five different semantic segmentation benchmarks;

- we further coupled Light-Weight RefineNet with two existing compact classification networks – MobileNet-v2 (Sandler et al., 2018) and NASNet-Mobile (Zoph et al., 2018), achieving 76.2% miou and 77.4% miou on the validation set of PASCAL VOC (Everingham et al., 2010), respectively (against 78.5% miou of a much larger network with the ResNet-50 backbone);

- in Chapter 4 we altered the design of Light-Weight RefineNet to perform multiple dense per-pixel tasks at once in real-time solving semantic segmentation, depth prediction and surface normals estimation at nearly 100FPS on 640×480 input images;

- to compensate for missing annotations of a certain modality, we generated proxy labels based on the knowledge distillation concept (Hinton et al., 2014), where we made use of Light-Weight RefineNet with the ResNet-152 backbone;

- in Chapter 5 we designed the search space of compact architectures with a pre-trained encoder and concentrated on speeding-up the training of an encoder-decoder segmentation network in order to estimate its potential in only few minutes;

- this, in turn, allowed us to automatically discover fast and accurate networks using reinforcement learning in only 8 GPU-days, with the best-performing one achieving 78% miou on the validation set of PASCAL VOC (against 75.9% miou of the manually designed DeepLab-v3 (Sandler et al., 2018)) and the fastest one with 77.3% miou showing nearly 20FPS on 512×512 on the low-compute JetsonTX2 platform (against 7FPS of the manually designed Light-Weight RefineNet);

- in Chapter 6 we proposed a novel design of the search space that reduced the dependency on a fully-pretrained encoder, allowing to discover tiny networks with less than 0.3M parameters;

- two top-discovered architectures from this space achieved nearly 68% miou on the test set of the popular urban driving dataset CityScapes (Cordts et al., 2016) – against 72.1% miou of Light-Weight RefineNet (with 153× more parameters) and 73.6% miou of RefineNet (with 393× more parameters).

—

Throughout the thesis, our primary focus was on the simplicity; since deep learning networks comprise extremely large numbers of parameters and hyperparameters (both structural – *e.g.* layer connectivity, – and optimisation-related – *e.g.* learning rates), it is important to have simple and reliable solutions available. In this thesis we did so by largely concentrating on structural patterns and operations used inside while abstracting away the rest of choices. Intuitively, if we find a well-performing neural network on a single problem, chances are that its structure can be transferred to a variety of other problems – *e.g.* see VGG (Simonyan and Zisserman, 2015) and ResNets (He et al., 2016), or what we did with Light-Weight RefineNet in Chapters 3 and 4, or transfer learning experiments in Chapter 5. In that sense, we may say that structures tend to *generalise* across various problems. On the other hand, optimisation-related hyperparameters tend to vary a lot across various problems and rarely generalise well. Furthermore, since we were concerned with the inference time, different optimisation strategies could not affect it in any way. What they did affect were the training time and convergence of the networks, and in Chapter 5 we focused on ways of speeding those up.

As we presented both the manual and automatic sides of designing compact networks, one may be left wondering which of them to use in practice? A good rule of thumb is to stick to the manual way and to adapt an existing network if the domain is well-known and the results are needed as soon as possible. Conversely, the automatic search may require several days of experimentations, especially on unexplored domains, since the user must first come up with a reasonable search space design.

Future Work

While in this manuscript we did lay the groundwork for creating practical dense per-pixel networks, several important directions were understandably omitted and are worth mentioning as future research directions:

1. **3D segmentation, video segmentation and instance segmentation.** Although we only considered a 2D semantic segmentation, the world around us is 3D and temporal with instances of various classes being present at all times.

   While for videos we can simply predict segmentation masks independently per each frame, such an approach will lack any temporal cohesion. How to propagate temporal information? Most solutions tend to rely on the optical flow that describes the motion in the scene between adjacent frames (Gadde et al., 2017; Zhu et al., 2017). Since the optical flow is usually expensive to compute and does not work with occlusions and newly appeared objects, better ways are needed. In our recent work we instead applied the NAS approach from Chapter 5 to search for a dynamic cell that propagates information between adjacent frames (Nekrasov et al., 2020a). To tackle long-term propagation, we can also search for a fully-differentiable way of storing and retrieving some information about the current frame – perhaps, as done in Neural Turing machines (Graves et al., 2014).

   Arguably the most well-known approach for instance segmentation, Mask R-CNN (He et al., 2017), is a direct combination of semantic segmentation and object detection (or bounding box prediction). Hence, it is an obvious question to ask whether existing improvements in semantic segmentation lead to similar improvements in instance segmentation (both in accuracy and runtime)? Moreover, recent fully-convolutional ways of solving object detection (Tian et al., 2019) make it even more straightforward to adapt semantic segmentation networks for the task of instance segmentation.

In Sect. 4.4.3 we showed a way of acquiring 3D segmentation in a SLAM system with a sequence of 2D predictions of segmentation and depth. This approach required lots of image frames and was far from perfect in capturing 3D structures. What if we already have a 3D shape (for example, in the form of a point cloud) and now want to semantically label it in real-time? While there are networks that support 3D inputs, *e.g.* PointNet (C. R. Qi et al., 2017), it is still an open challenge to make them practical to the same extent as current 2D segmentation networks are.

In addition to the above, we should not only consider each of the above tasks in isolation, but also all of them at the same time – i.e. *how to solve a 3D video instance semantic segmentation in real-time?*

2. **Uncertainty and diagnostics.** In this thesis we did not consider any notions of uncertainty in network predictions. Nevertheless, uncertainty is crucial when using these networks in real situations (Kendall and Gal, 2017). At its core, uncertainty quantifies the confidence in the network predictions. Since uncertainty estimates tend to require lots of computations, having small and fast networks can dramatically reduce these costs. On the other hand, small networks are also amenable to quick diagnostics on the given datasets that would allow to spot weaknesses of the networks before using them in real situations. Some preliminary work in this direction is already available (Nekrasov et al., 2018a).

   An important question to ask is how to build dense per-pixel networks that are aware of their own limitations and that can deal with uncertainties in their predictions?

3. **Semi-supervised and unsupervised learning.** In Chapter 4 we first encountered the problem of missing labels. We overcame it through knowledge distillation using an existing powerful network, or teacher. What should we do if we do not have a good teacher network? How to acquire missing labels in such cases? Or, more generally, how to reduce the costs of annotations?

For certain dense per-pixel tasks, such as depth estimation, it is possible to train a network in a fully-unsupervised manner by exploiting geometrical constraints (Garg et al., 2016; Godard et al., 2017). For semantic segmentation, one may rely on domain adaptation and use simulated data to pre-train the network (Y. Zhang et al., 2017).

With an abundance of unlabelled data in the world, can we find a way of leveraging that data? For example, it has been shown that tracking and dense correspondences naturally appear by learning to colourise video frames (Vondrick et al., 2018; Lai and Xie, 2019). It is highly likely that the features learned in that process can also be helpful for semantic segmentation and depth estimation. On the related note, representations extracted through contrastive learning (T. Chen et al., 2020; Sohn et al., 2020) lead to improvements not only in image classification (Khosla et al., 2020), but also in reinforcement learning (Srinivas et al., 2020). In contrastive learning, the network is tasked to pull closer representations of the same sample (*e.g.* of the input image and its augmented version), while pushing apart representations of different samples (*e.g.* of two different input images). Can this paradigm improve performance in dense per-pixel tasks, too?

Overall, we should be asking how to build practical dense per-pixel systems without the need for large fully-annotated datasets?

Appendices

A

Qualitative Results of Light-Weight RefineNet

In addition to the results presented in Chapter 3, here we include more qualitative figures. We denote Light-Weight RefineNet with different ResNet backbones (ResNet-50, ResNet-101 and ResNet-152) as *RF-50-LW*, *RF-101-LW* and *RF-152-LW*, correspondingly, and ground truth masks as 'GT'.

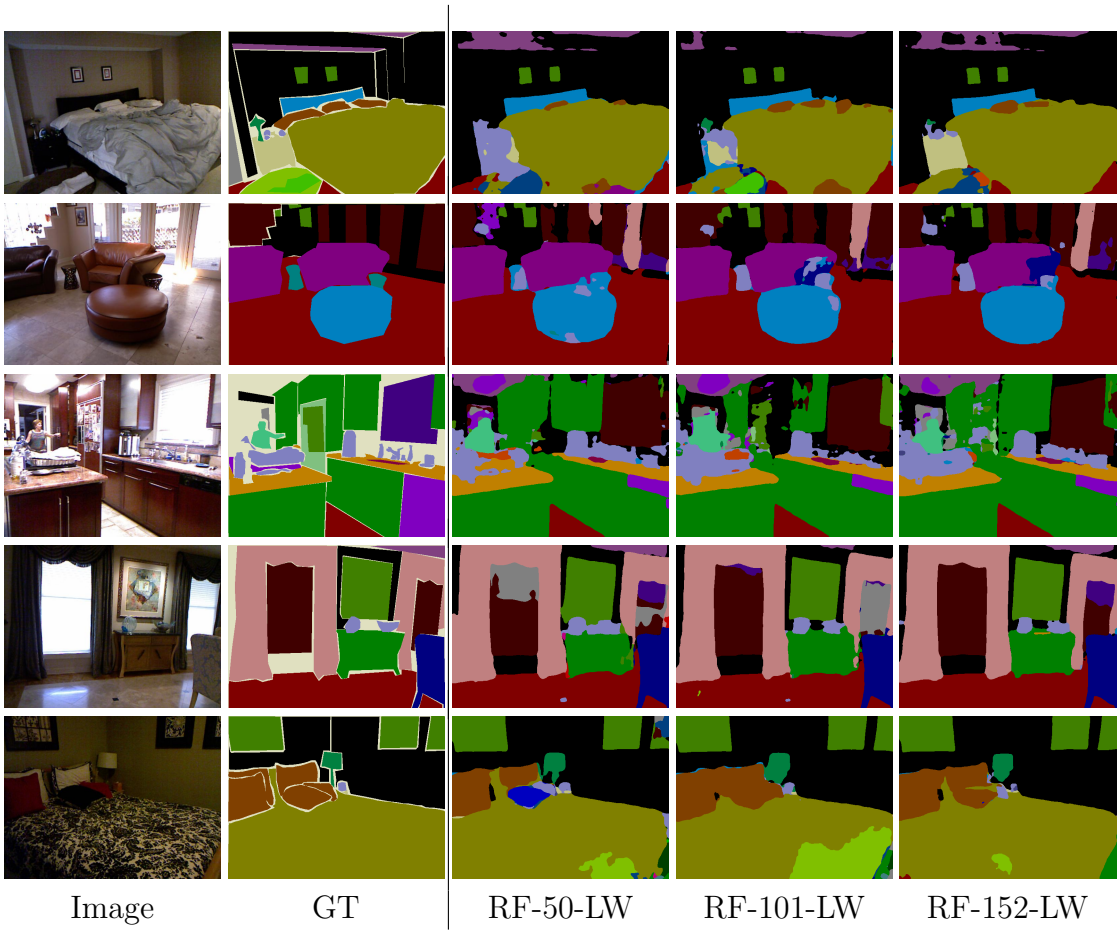


Figure A.1: Visual results on validation set of NYUDv2 with ResNet-backbones.

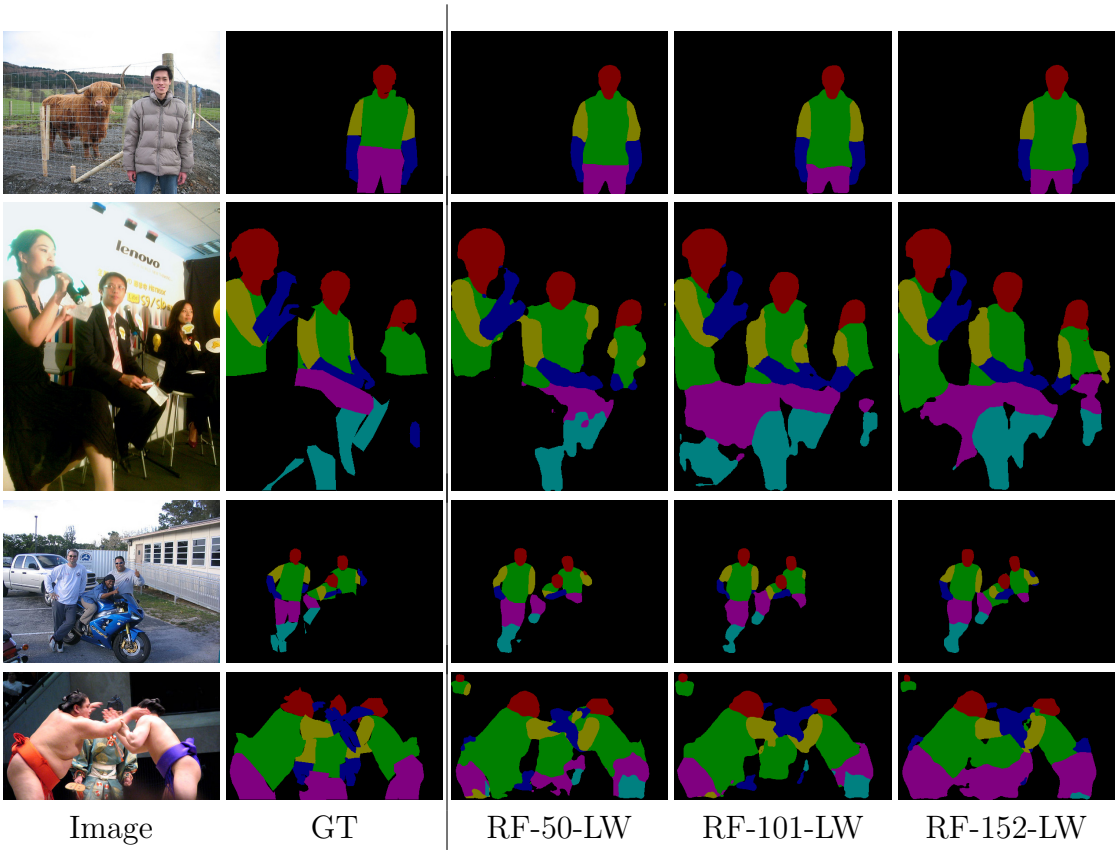


Figure A.2: Visual results on validation set of PASCAL Person-Part with ResNet-backbones.

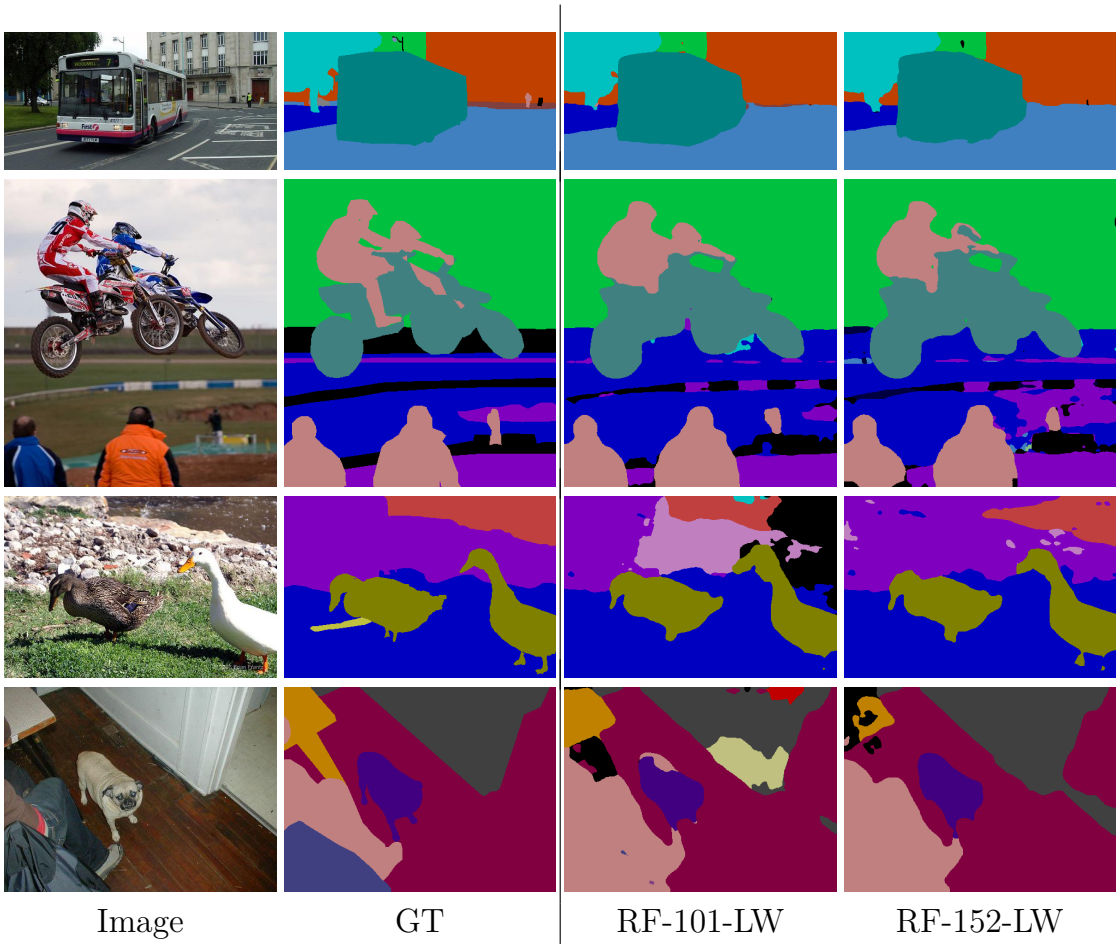


Figure A.3: Visual results on validation set of PASCAL-Context with ResNet-backbones.

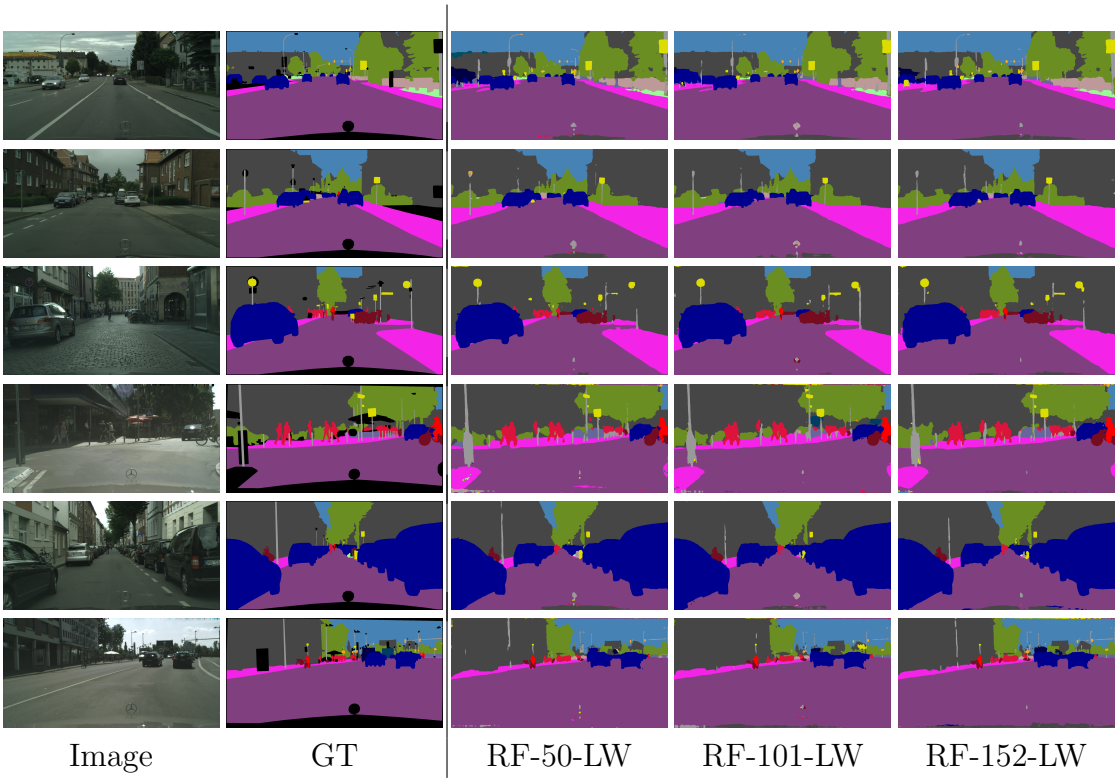


Figure A.4: Visual results on validation set of CityScapes with ResNet-backbones.

B

Experimental Results of Efficient Neural Architecture Search

Here we include additional details about experimental results from Chapter 5.

Semantic Segmentation

We start training with the learning rates of 1e-3 and 3e-3 – for the encoder and the decoder, respectively. The encoder weights are updated using SGD with the momentum value of 0.9, whereas for the decoder part we rely on Adam (Kingma and Ba, 2014) with default parameters of β_1=0.9, β_2=0.99 and ϵ=0.001. We exploit the batch size of 64, evenly divided over two 1080Ti GPU cards. Each image in the batch is randomly scaled in the range of $[0.5, 2.0]$, randomly mirrored, before being randomly cropped and padded to the size of 450×450. During training, in order to calculate the loss term, we upsample the logits to the size of the target mask.

In addition to the results presented in the main text, we provide per-class intersection-over-union values across the models in Table B.1.

Model	bg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
DL-v3 (Sandler et al., 2018)	0.94	0.873	0.416	0.849	0.647	**0.753**	0.937	0.86	**0.904**	0.391	**0.893**	0.564	**0.847**	**0.892**	0.831	0.844	0.578	0.859	0.525	0.852	0.677	0.759
RF-LW	0.942	**0.895**	0.594	0.872	0.761	0.669	0.912	0.85	0.876	0.383	0.801	**0.605**	0.804	0.886	0.835	0.854	**0.603**	0.843	0.479	0.834	**0.703**	0.762
arch0	**0.947**	0.885	0.558	0.885	0.748	0.74	**0.944**	0.868	0.898	**0.429**	0.863	0.604	0.846	0.842	0.866	0.86	0.592	**0.869**	**0.593**	**0.875**	0.669	**0.780**
arch1	0.944	0.888	**0.615**	0.866	**0.781**	0.733	0.933	0.865	0.894	0.394	0.828	0.603	0.833	0.848	0.854	0.855	0.568	0.829	0.555	0.85	0.662	0.771
arch2	**0.947**	0.873	0.589	**0.887**	0.753	0.75	0.943	**0.885**	0.895	0.372	0.829	0.635	0.845	0.832	**0.867**	**0.866**	0.555	0.843	0.537	0.851	0.671	0.773

Table B.1: Per-class intersection-over-union on the validation set of PASCAL VOC.

Depth estimation

For depth estimation, we start training with the learning rates of $1e$-3 and $7e$-3 - for the encoder and the decoder, respectively. For both we use SGD with the momentum value of 0.9, and anneal the learning rates via the *'Poly'* schedule: $lr * (1 - \frac{epoch}{400})^{0.9}$. The training is stopped after 300 epochs. We exploit the batch size of 32, evenly divided over two 1080Ti GPU cards. Each image in the batch is randomly scaled in the range of $[0.5, 2.0]$, randomly mirrored, before being randomly cropped and padded to the size of 500×500. We upsample the logits to the size of the target mask and use the reverse Huber loss (Laina et al., 2016) for optimisation, ignoring pixels with missing depth measurements.

We visualise qualitative results on the validation set in Fig. B.1.

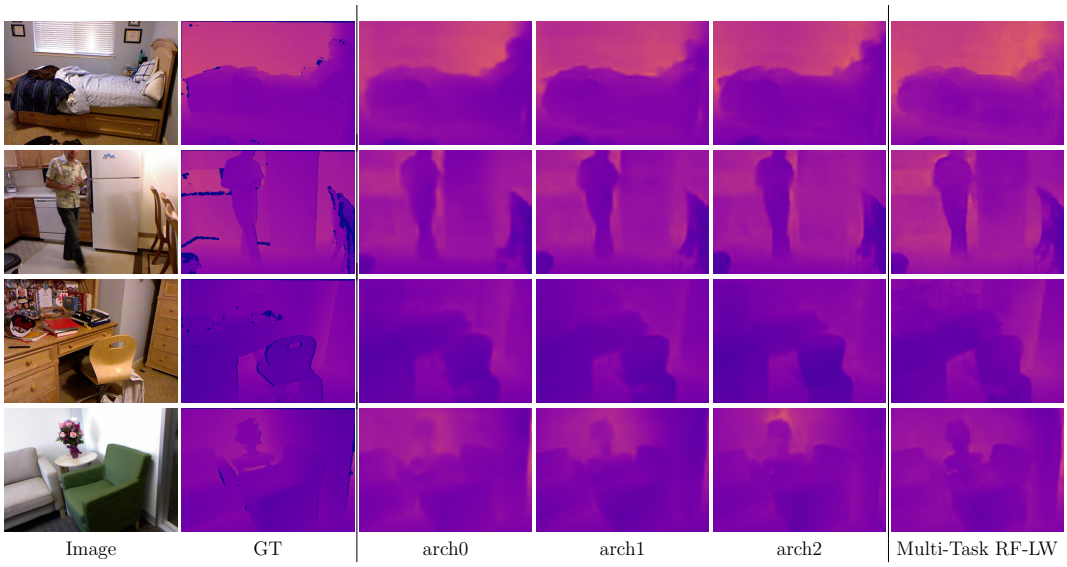


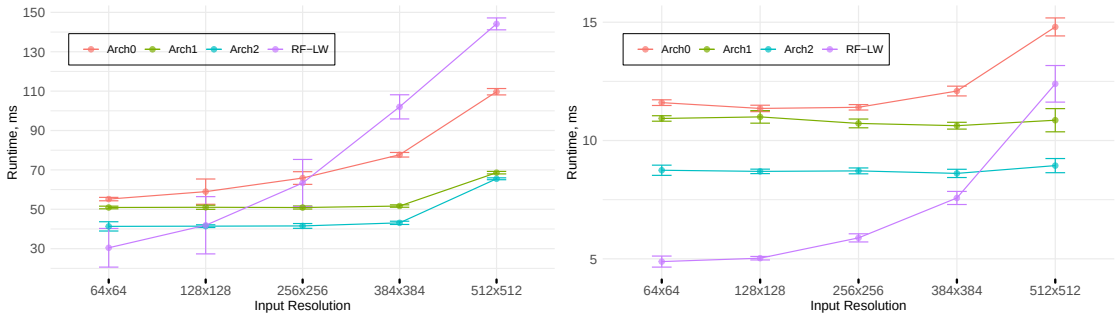
Figure B.1: Depth estimation qualitative results on NYUDv2. Dark-blue pixels in ground truth are pixels with missing depth measurements. 'GT' stands for 'ground truth'.

C

Jetson TX2 runtime

During our experiments in Chapter 5 we observed a significant difference between models' runtime on JetsonTX2 and 1080Ti. To better understand it, we additionally measured the runtime of each discovered architecture from Chapter 5 together with Light-Weight RefineNet varying the input resolution.

As evident from Fig. C.1, the models with a larger number of floating point operations (i.e., *Arch0* and *RF-LW*) do not scale well with the input resolution. The effect is even more pronounced on JetsonTX2, as been independently confirmed by an NVIDIA employer in a private conversation.



(a) JetsonTX2 **(b)** 1080Ti

Figure C.1: Models' runtime on JetsonTX2 (**a**) and 1080Ti (**b**). We visualise mean together with standard deviation values over 100 passes of each model.

D

Automatically Discovered Decoder Structures

Here we provide visualisations of architectures discovered in Chapter 5.

Architecture description

We use a list of integers to encode a single architecture found by the controller, corresponding to the output sequence of the RNN. Specifically, the list describes the connectivity structure followed by the cell configuration. For example, the following connectivity structure $[[c_1, c_2], [c_3, c_4], [c_5, c_6]]$ contains three pairs of digits, indicating the input index c_k of a corresponding layer in the sampling pool. The cell configuration, $[o_1, [i_2, i_3, o_2, o_3], [i_4, i_5, o_4, o_5], [i_6, i_7, o_6, o_7]$, comprises the first operation o_1 followed by three cell branches with the operation o_j applied on the index i_j.

We provide the description of operations in Table D.1, and visualise the discovered structures in Fig. D.1 (*arch0*), Fig. D.2 (*arch1*), and Fig. D.3 (*arch2*). Note that inside the cell only the final summation operator is displayed as intermediate summations would lead to identical structures.

Index	Abbreviation	Description
0	conv1x1	conv 1×1
1	conv3x3	conv 3×3
2	sep3x3	separable conv 3×3
3	sep5x5	separable conv 5×5
4	gap	global average pooling followed by upsampling and conv 1×1
5	conv3x3 rate 3	conv 3×3 with dilation rate 3
6	conv3x3 rate 12	conv 3×3 with dilation rate 12
7	sep3x3 rate 3	separable conv 3×3 with dilation rate 3
8	sep5x5 rate 6	separable conv 5×5 with dilation rate 6
9	skip	skip-connection
10	zero	zero-operation that effectively nullifies the path

Table D.1: Operation indices and abbreviations used to describe the cell configuration.

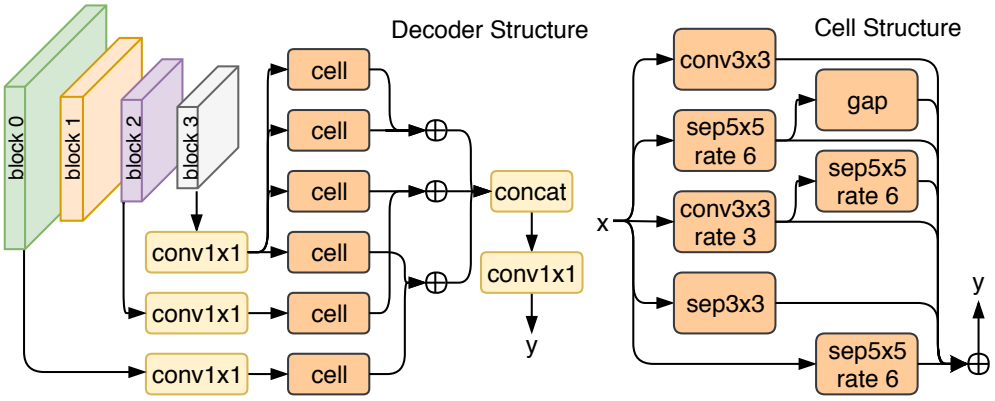


Figure D.1: arch0: $[[[3,3],[3,2],[3,0]],[8,[0,0,5,2],[0,2,8,8],[0,5,1,4]]]$

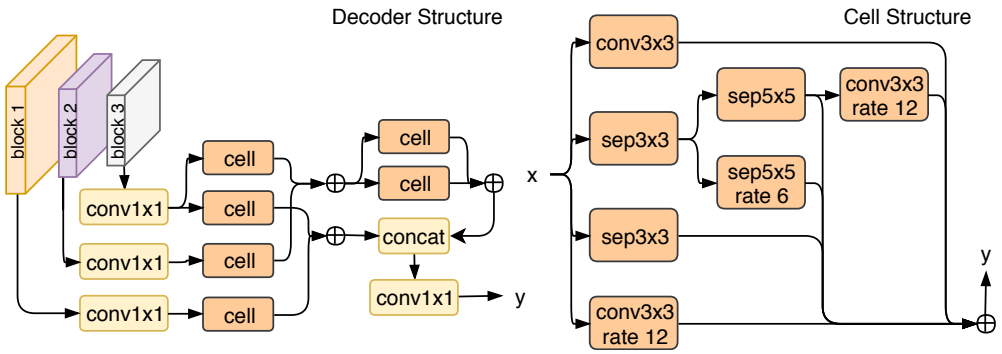


Figure D.2: arch1: $[[[2,3],[3,1],[4,4]],[2,[1,0,3,6],[0,1,2,8],[2,0,6,1]]]$

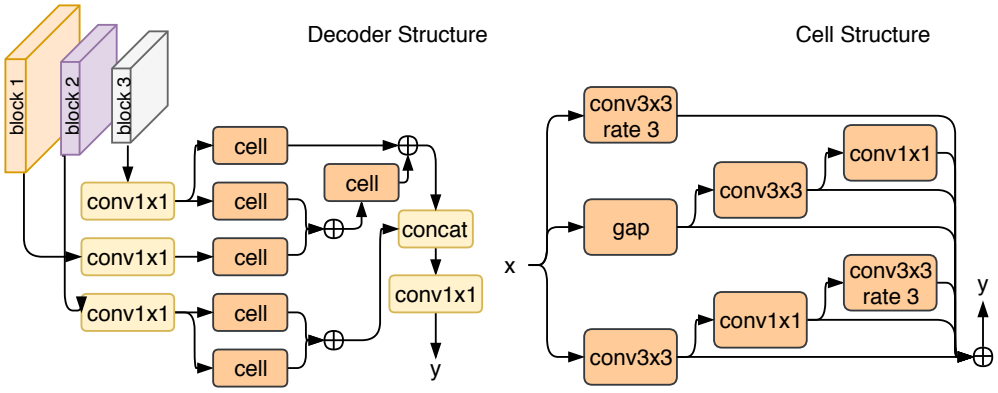


Figure D.3: arch2: $[[[1, 3], [4, 3], [2, 2]], [5, [0, 0, 4, 1], [3, 2, 0, 1], [5, 6, 5, 0]]]$

Bibliography

Angeline, Peter J., Gregory M. Saunders, and Jordan B. Pollack (1994). "An evolutionary algorithm that constructs recurrent neural networks". In: *IEEE Trans. Neural Netw.*

Ba, Jimmy and Rich Caruana (2014). "Do Deep Nets Really Need to be Deep?" In: *Proc. Advances in Neural Inf. Process. Syst.*

Badrinarayanan, Vijay, Ankur Handa, and Roberto Cipolla (2015). "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling". In: *arXiv: Comp. Res. Repository* abs/1505.07293.

Baker, Bowen, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar (2017). "Designing Neural Network Architectures using Reinforcement Learning". In: *Proc. Int. Conf. Learn. Representations*.

Baxter, Jonathan (2000). "A Model of Inductive Bias Learning". In: *J. Artif. Intell. Res.*

Bergstra, James, Daniel Yamins, and David D. Cox (2013). "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proc. Int. Conf. Mach. Learn.*

Brostow, Gabriel J., Julien Fauqueur, and Roberto Cipolla (2009). "Semantic object classes in video: A high-definition ground truth database". In: *Pattern Recognition Letters* 30.

Bucila, Cristian, Rich Caruana, and Alexandru Niculescu-Mizil (2006). "Model compression". In: *ACM SIGKDD*.

Caruana, Rich (1993). "Multitask Learning: A Knowledge-Based Source of Inductive Bias". In: *Proc. Int. Conf. Mach. Learn.*

Chen, Liang-Chieh, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens (2018a). "Searching for Efficient Multi-Scale Architectures for Dense Image Prediction". In: *Proc. Advances in Neural Inf. Process. Syst.*

Chen, Liang-Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille (2015). "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: *Proc. Int. Conf. Learn. Representations*.

Chen, Liang-Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille (2018b). "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *IEEE Trans. Pattern Anal. Mach. Intell.*

Chen, Liang-Chieh, George Papandreou, Florian Schroff, and Hartwig Adam (2017). "Rethinking Atrous Convolution for Semantic Image Segmentation". In: *arXiv: Comp. Res. Repository* abs/1706.05587.

Chen, Liang-Chieh, Yi Yang, Jiang Wang, Wei Xu, and Alan L. Yuille (2016). "Attention to Scale: Scale-Aware Semantic Image Segmentation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Chen, Liang-Chieh, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam (2018c). "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *Proc. Eur. Conf. Comp. Vis.*

Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton (2020). "A Simple Framework for Contrastive Learning of Visual Representations". In: *arXiv: Comp. Res. Repository* abs/2002.05709.

Chen, Xianjie, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan L. Yuille (2014). "Detect What You Can: Detecting and Representing Objects Using Holistic Models and Body Parts". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Chollet, François (2017). "Xception: Deep Learning with Depthwise Separable Convolutions". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Ciresan, Dan C., Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber (2012). "Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images". In: *Proc. Advances in Neural Inf. Process. Syst.*

Cordts, Marius, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele (2016). "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Couprie, Camille, Clément Farabet, Laurent Najman, and Yann LeCun (2013). "Indoor semantic segmentation using depth information". In: *Proc. Int. Conf. Learn. Representations.*

Csurka, Gabriela and Florent Perronnin (2008). "A Simple High Performance Approach to Semantic Segmentation". In: *Proc. British Machine Vis. Conf.*

Dalal, Navneet and Bill Triggs (2005). "Histograms of Oriented Gradients for Human Detection". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li (2009). "ImageNet: A large-scale hierarchical image database". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Denton, Emily L., Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus (2014). "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation". In: *Proc. Advances in Neural Inf. Process. Syst.*

Dharmasiri, Thanuja, Andrew Spek, and Tom Drummond (2017). "Joint Prediction of Depths, Normals and Surface Curvature from RGB Images using CNNs". In: *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems.*

— (2018). "ENG: End-to-end Neural Geometry for Robust Depth and Pose Estimation using CNNs". In: *Proc. Asian Conf. Comp. Vis.*

Du, Simon and Jason Lee (2018). "On the Power of Over-parametrization in Neural Networks with Quadratic Activation". In: *Proc. Int. Conf. Mach. Learn.*

Dvornik, Nikita, Konstantin Shmelkov, Julien Mairal, and Cordelia Schmid (2017). "BlitzNet: A Real-Time Deep Network for Scene Understanding". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Eigen, David and Rob Fergus (2015). "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Eigen, David, Christian Puhrsch, and Rob Fergus (2014). "Depth map prediction from a single image using a multi-scale deep network". In: *Proc. Advances in Neural Inf. Process. Syst.*

Everingham, Mark, Luc J. Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman (2010). "The Pascal Visual Object Classes (VOC) Challenge". In: *Int. J. Comput. Vision.*

Farabet, Clément, Camille Couprie, Laurent Najman, and Yann LeCun (2013). "Learning Hierarchical Features for Scene Labeling". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35.8, pp. 1915–1929.

Felzenszwalb, Pedro F. and Daniel P. Huttenlocher (2004). "Efficient Graph-Based Image Segmentation". In: *Int. J. Comput. Vision* 59.2.

Fulkerson, Brian, Andrea Vedaldi, and Stefano Soatto (2009). "Class segmentation and object localization with superpixel neighborhoods". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Gadde, Raghudeep, Varun Jampani, and Peter V. Gehler (2017). "Semantic Video CNNs Through Representation Warping". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Garg, Ravi, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid (2016). "Unsupervised cnn for single view depth estimation: Geometry to the rescue". In: *Proc. Eur. Conf. Comp. Vis.*

Geiger, Andreas, Philip Lenz, Christoph Stiller, and Raquel Urtasun (2013). "Vision meets robotics: The KITTI dataset". In: *I. J. Robotics Res.*

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proc. Int. Conf. Artificial Intell. & Stat.*

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep Sparse Rectifier Neural Networks". In: *Proc. Int. Conf. Artificial Intell. & Stat.*

Godard, Clément, Oisin Mac Aodha, and Gabriel J Brostow (2017). "Unsupervised monocular depth estimation with left-right consistency". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Godard, Clément, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow (2019). "Digging into Self-Supervised Monocular Depth Prediction". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Gong, Yunchao, Liu Liu, Ming Yang, and Lubomir D. Bourdev (2014). "Compressing Deep Convolutional Networks using Vector Quantization". In: *arXiv: Comp. Res. Repository* abs/1412.6115.

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). "Neural Turing Machines". In: *arXiv: Comp. Res. Repository* abs/1410.5401.

Gupta, Saurabh, Pablo Arbelaez, and Jitendra Malik (2013). "Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Han, Song, Huizi Mao, and William J. Dally (2015a). "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *arXiv: Comp. Res. Repository* abs/1510.00149.

Han, Song, Jeff Pool, John Tran, and William J. Dally (2015b). "Learning both Weights and Connections for Efficient Neural Network". In: *Proc. Advances in Neural Inf. Process. Syst.*

Hariharan, Bharath, Pablo Arbelaez, Lubomir D. Bourdev, Subhransu Maji, and Jitendra Malik (2011). "Semantic contours from inverse detectors". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Hassibi, Babak and David G. Stork (1992). "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon". In: *Proc. Advances in Neural Inf. Process. Syst.*

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick (2017). "Mask R-CNN". In: *Proc. IEEE Int. Conf. Comp. Vis.*

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2014). "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *Proc. Eur. Conf. Comp. Vis.*

— (2016). "Deep Residual Learning for Image Recognition". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Hinton, Geoffrey E., Oriol Vinyals, and Jeffrey Dean (2014). "Distilling the Knowledge in a Neural Network". In: *Proc. Advances in Neural Inf. Process. Syst.*

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation*.

Holschneider, Matthias, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian (1990). "A real-time algorithm for signal analysis with the help of the wavelet transform". In: *Wavelets*. Springer, pp. 286–297.

Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv: Comp. Res. Repository* abs/1704.04861.

Iandola, Forrest N., Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer (2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size". In: *arXiv: Comp. Res. Repository* abs/1602.07360.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proc. Int. Conf. Mach. Learn.*

Jacob, Benoit, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko (2018). "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Jaderberg, Max, Andrea Vedaldi, and Andrew Zisserman (2014). "Speeding up Convolutional Neural Networks with Low Rank Expansions". In: *Proc. British Machine Vis. Conf.*

Kandasamy, Kirthevasan, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric Xing (2018). "Neural Architecture Search with Bayesian Optimisation and Optimal Transport". In: *Proc. Advances in Neural Inf. Process. Syst.*

Kendall, Alex and Yarin Gal (2017). "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In: *Proc. Advances in Neural Inf. Process. Syst.*

Kendall, Alex, Yarin Gal, and Roberto Cipolla (2018). "Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Khosla, Prannay, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan (2020). "Supervised Contrastive Learning". In: *arXiv: Comp. Res. Repository* abs/2004.11362.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *arXiv: Comp. Res. Repository* abs/1412.6980.

Kokkinos, Iasonas (2017). "UberNet: Training a Universal Convolutional Neural Network for Low-, Mid-, and High-Level Vision Using Diverse Datasets and Limited Memory". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Krähenbühl, Philipp and Vladlen Koltun (2011). "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *Proc. Advances in Neural Inf. Process. Syst.*

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proc. Advances in Neural Inf. Process. Syst.*

Kuznietsov, Yevhen, Jörg Stückler, and Bastian Leibe (2017). "Semi-supervised deep learning for monocular depth map prediction". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Ladicky, Lubor, Christopher Russell, Pushmeet Kohli, and Philip H. S. Torr (2009). "Associative hierarchical CRFs for object class image segmentation". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Lai, Zihang and Weidi Xie (2019). "Self-supervised Video Representation Learning for Correspondence Flow". In: *Proc. British Machine Vis. Conf.*

Laina, Iro, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab (2016). "Deeper depth prediction with fully convolutional residual networks". In: *Proc. IEEE Int. Conf. on 3D Vis.*

LeCun, Yann, Yoshua Bengio, and Geoffrey E. Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

LeCun, Yann, John S. Denker, and Sara A. Solla (1989). "Optimal Brain Damage". In: *Proc. Advances in Neural Inf. Process. Syst.*

Li, Xiaoxiao, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang (2017). "Not All Pixels Are Equal: Difficulty-Aware Semantic Segmentation via Deep Layer Cascade". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Lin, Guosheng, Anton Milan, Chunhua Shen, and Ian D. Reid (2017). "RefineNet: Multi-path Refinement Networks for High-Resolution Semantic Segmentation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Lin, Guosheng, Chunhua Shen, Ian D. Reid, and Anton van den Hengel (2016). "Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Lin, Tsung-Yi, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick (2014). "Microsoft COCO: Common Objects in Context". In: *Proc. Eur. Conf. Comp. Vis.*

Liu, Chenxi, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Fei-Fei Li (2019). "Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Liu, Chenxi, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy (2018). "Progressive Neural Architecture Search". In: *Proc. Eur. Conf. Comp. Vis.*

Liu, Fayao, Chunhua Shen, and Guosheng Lin (2015). "Deep convolutional neural fields for depth estimation from a single image". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Liu, Hanxiao, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu (2018). "Hierarchical Representations for Efficient Architecture Search". In: *Proc. Int. Conf. Learn. Representations*.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2019). "DARTS: Differentiable Architecture Search". In: *Proc. Int. Conf. Learn. Representations*.

Liu, Shikun, Edward Johns, and Andrew J. Davison (2019). "End-To-End Multi-Task Learning With Attention". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Liu, Ziwei, Xiaoxiao Li, Ping Luo, Chen Change Loy, and Xiaoou Tang (2015). "Semantic Image Segmentation via Deep Parsing Network". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2015). "Fully convolutional networks for semantic segmentation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Lowe, David G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision*.

Luo, Renqian, Fei Tian, Tao Qin, and Tie-Yan Liu (2018). "Neural Architecture Optimization". In: *Proc. Advances in Neural Inf. Process. Syst.*

Ma, Ningning, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun (2018). "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design". In: *Proc. Eur. Conf. Comp. Vis.*

Mallya, Arun, Dillon Davis, and Svetlana Lazebnik (2018). "Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights". In: *Proc. Eur. Conf. Comp. Vis.*

Mallya, Arun and Svetlana Lazebnik (2018). "PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

McCormac, John, Ankur Handa, Andrew J. Davison, and Stefan Leutenegger (2017). "SemanticFusion: Dense 3D semantic mapping with convolutional neural networks". In: *Int. Conf. on Robot. and Autom.*

Mehta, Sachin, Mohammad Rastegari, Anat Caspi, Linda G. Shapiro, and Hannaneh Hajishirzi (2018). "ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation". In: *Proc. Eur. Conf. Comp. Vis.*

Mehta, Sachin, Mohammad Rastegari, Linda G. Shapiro, and Hannaneh Hajishirzi (2019). "ESPNetv2: A Light-weight, Power Efficient, and General Purpose Convolutional Neural Network". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Misra, Ishan, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert (2016). "Cross-Stitch Networks for Multi-task Learning". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Molchanov, Pavlo, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz (2019). "Importance Estimation for Neural Network Pruning". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Mottaghi, Roozbeh, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan L. Yuille (2014). "The Role of Context for Object Detection and Semantic Segmentation in the Wild". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Mousavian, Arsalan, Hamed Pirsiavash, and Jana Kosecka (2016). "Joint Semantic Segmentation and Depth Estimation with Deep Convolutional Networks". In: *Proc. IEEE Int. Conf. on 3D Vis.*

Nekrasov, Vladimir, Hao Chen, Chunhua Shen, and Ian D. Reid (2019a). "Fast Neural Architecture Search of Compact Semantic Segmentation Models via Auxiliary Cells". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

— (2020a). "Architecture Search of Dynamic Cells for Semantic Video Segmentation". In: *Proc. IEEE Winter Conf. on App. of Comp. Vis.*

Nekrasov, Vladimir, Thanuja Dharmasiri, Andrew Spek, Tom Drummond, Chunhua Shen, and Ian D. Reid (2019b). "Real-Time Joint Semantic Segmentation and

Depth Estimation Using Asymmetric Annotations". In: *Int. Conf. on Robot. and Autom.*

Nekrasov, Vladimir, Chunhua Shen, and Ian D. Reid (2018a). "Diagnostics in Semantic Segmentation". In: *arXiv: Comp. Res. Repository* abs/1809.10328.

— (2018b). "Light-Weight RefineNet for Real-Time Semantic Segmentation". In: *Proc. British Machine Vis. Conf.*

— (2020b). "Template-Based Automatic Search of Compact Semantic Segmentation Architectures". In: *Proc. IEEE Winter Conf. on App. of Comp. Vis.*

Noh, Hyeonwoo, Seunghoon Hong, and Bohyung Han (2015). "Learning Deconvolution Network for Semantic Segmentation". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Opper, Manfred, Ole Winther, et al. (2001). "From naive mean field theory to the TAP equations". In: *Advanced mean field methods: theory and practice*, pp. 7–20.

Paszke, Adam, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello (2016). "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation". In: *arXiv: Comp. Res. Repository* abs/1606.02147.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Proc. Advances in Neural Inf. Process. Syst.*

Pham, Hieu, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean (2018). "Efficient Neural Architecture Search via Parameter Sharing". In: *Proc. Int. Conf. Mach. Learn.*

Pinheiro, Pedro H. O. and Ronan Collobert (2014). "Recurrent convolutional neural networks for scene labeling". In: *Proc. Int. Conf. Mach. Learn.*

Plath, Nils, Marc Toussaint, and Shinichi Nakajima (2009). "Multi-class image segmentation using conditional random fields and global classification". In: *Proc. Int. Conf. Mach. Learn.*

Pohlen, Tobias, Alexander Hermans, Markus Mathias, and Bastian Leibe (2017). "Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Polyak, Boris T and Anatoli B Juditsky (1992). "Acceleration of stochastic approximation by averaging". In: *SIAM Journal on Control and Optimization.*

Qi, Charles Ruizhongtai, Hao Su, Kaichun Mo, and Leonidas J. Guibas (2017). "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Qi, Xiaojuan, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia (2018). "GeoNet: Geometric Neural Network for Joint Depth and Surface Normal Estimation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Real, Esteban, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin (2017). "Large-Scale Evolution of Image Classifiers". In: *Proc. Int. Conf. Mach. Learn.*

Romera, Eduardo, Jose M. Alvarez, Luis Miguel Bergasa, and Roberto Arroyo (2018). "ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation". In: *IEEE Trans. Intelligent Transportation Systems* 19.

Romero, Adriana, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio (2015). "FitNets: Hints for Thin Deep Nets". In: *Proc. Int. Conf. Learn. Representations.*

Ros, G., S. Ramos, M. Granados, A. Bakhtiary, D. Vazquez, and A.M. Lopez (2015). "Vision-based Offline-Online Perception Paradigm for Autonomous Driving". In: *Proc. IEEE Winter Conf. on App. of Comp. Vis.*

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536.

Sandler, Mark, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen (2018). "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv: Comp. Res. Repository.*

Shotton, Jamie, Matthew Johnson, and Roberto Cipolla (2008). "Semantic texton forests for image categorization and segmentation". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Shotton, Jamie, John M. Winn, Carsten Rother, and Antonio Criminisi (2006). "*TextonBoost*: Joint Appearance, Shape and Context Modeling for Multi-class Object Recognition and Segmentation". In: *Proc. Eur. Conf. Comp. Vis.*

Sifre, Laurent and Stéphane Mallat (2014). "Rigid-Motion Scattering for Texture Classification". In: *arXiv: Comp. Res. Repository* abs/1403.1687.

Silberman, Nathan, Derek Hoiem, Pushmeet Kohli, and Rob Fergus (2012). "Indoor Segmentation and Support Inference from RGBD Images". In: *Proc. Eur. Conf. Comp. Vis.*

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *Proc. Int. Conf. Learn. Representations*.

Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Proc. Advances in Neural Inf. Process. Syst.*

Sohn, Kihyuk, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel (2020). "FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence". In: *arXiv: Comp. Res. Repository* abs/2001.07685.

Soltanolkotabi, Mahdi, Adel Javanmard, and Jason D. Lee (2018). "Theoretical insights into the optimization landscape of over-parameterized shallow neural networks". In: *IEEE Transactions on Information Theory*.

Spek, Andrew, Thanuja Dharmasiri, and Tom Drummond (2018). "CReaM: Condensed Real-time Models for Depth Prediction using Convolutional Neural Networks". In: *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems*.

Srinivas, Aravind, Michael Laskin, and Pieter Abbeel (2020). "CURL: Contrastive Unsupervised Representations for Reinforcement Learning". In: *arXiv: Comp. Res. Repository* abs/2004.04136.

Standley, Trevor, Amir Roshan Zamir, Dawn Chen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese (2019). "Which Tasks Should Be Learned Together in Multi-task Learning?" In: *arXiv: Comp. Res. Repository* abs/1905.07553.

Stanley, Kenneth O., David B. D'Ambrosio, and Jason Gauci (2009). "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks". In: *Artificial Life*.

Stanley, Kenneth O. and Risto Miikkulainen (2002). "Evolving Neural Network through Augmenting Topologies". In: *Evolutionary Computation*.

Strubell, Emma, Ananya Ganesh, and Andrew McCallum (2019). "Energy and Policy Considerations for Deep Learning in NLP". In: *Proc. Conf. Assoc. for Comp. Linguistics*.

Sutton, Charles A. and Andrew McCallum (2005). "Piecewise Training for Undirected Models". In: *Proc. Conf. Uncertainty in Artificial Intelli.*

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement learning - an introduction*. MIT Press.

Swersky, Kevin, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael A Osborne (2014). "Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces". In: *arXiv: Comp. Res. Repository*.

Tan, Mingxing, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le (2019). "MnasNet: Platform-Aware Neural Architecture Search for Mobile". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Tian, Zhi, Chunhua Shen, Hao Chen, and Tong He (2019). "FCOS: Fully Convolutional One-Stage Object Detection". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Vondrick, Carl, Abhinav Shrivastava, Alireza Fathi, Sergio Guadarrama, and Kevin Murphy (2018). "Tracking Emerges by Colorizing Videos". In: *Proc. Eur. Conf. Comp. Vis.*

Wang, Kuan, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han (2019). "HAQ: Hardware-Aware Automated Quantization With Mixed Precision". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Wang, Shenlong, Sanja Fidler, and Raquel Urtasun (2015). "Holistic 3D scene understanding from a single geo-tagged image". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Watkins, Christopher John Cornish Hellaby (1989). "Learning from delayed rewards". PhD thesis. King's College, Cambridge.

Whelan, Thomas, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger (2016). "ElasticFusion: Real-time dense SLAM and light source estimation". In: *I. J. Robotics Res.*

Wikipedia contributors (2020). *Lidar — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-January-2020]. URL: https://en.wikipedia.org/w/index.php?title=Lidar&oldid=935771335.

Wu, Bichen, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer (2019). "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Wu, Zifeng, Chunhua Shen, and Anton van den Hengel (2019). "Wider or Deeper: Revisiting the ResNet Model for Visual Recognition". In: *Pattern Recogn.*

Yu, Changqian, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang (2018). "BiSeNet: Bilateral Segmentation Network for Real-Time Semantic Segmentation". In: *Proc. Eur. Conf. Comp. Vis.*

Yu, Fisher and Vladlen Koltun (2016). "Multi-Scale Context Aggregation by Dilated Convolutions". In: *Proc. Int. Conf. Learn. Representations*.

Yu, Fisher, Vladlen Koltun, and Thomas A. Funkhouser (2017). "Dilated Residual Networks". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Zamir, Amir Roshan, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese (2018). "Taskonomy: Disentangling Task Transfer Learning". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Zeiler, Matthew D. and Rob Fergus (2014). "Visualizing and Understanding Convolutional Networks". In: *Proc. Eur. Conf. Comp. Vis.*

Zhang, Xiangyu, Xinyu Zhou, Mengxiao Lin, and Jian Sun (2018). "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Zhang, Yang, Philip David, and Boqing Gong (2017). "Curriculum Domain Adaptation for Semantic Segmentation of Urban Scenes". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Zhang, Zhengyou (2012). "Microsoft Kinect Sensor and Its Effect". In: *IEEE MultiMedia* 19.2.

Zhao, Hengshuang, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia (2018). "ICNet for Real-Time Semantic Segmentation on High-Resolution Images". In: *Proc. Eur. Conf. Comp. Vis.*

Zhao, Hengshuang, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia (2017). "Pyramid Scene Parsing Network". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Zheng, Shuai, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr (2015). "Conditional Random Fields as Recurrent Neural Networks". In: *Proc. IEEE Int. Conf. Comp. Vis.*

Zhou, Aojun, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen (2017). "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights". In: *Proc. Int. Conf. Learn. Representations*.

Zhou, Bolei, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba (2015). "Object Detectors Emerge in Deep Scene CNNs". In: *Proc. Int. Conf. Learn. Representations*.

Zhou, Shuchang, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou (2016). "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients". In: *arXiv: Comp. Res. Repository* abs/1606.06160.

Zhu, Xizhou, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei (2017). "Deep Feature Flow for Video Recognition". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Zoph, Barret and Quoc V. Le (2017). "Neural Architecture Search with Reinforcement Learning". In: *Proc. Int. Conf. Learn. Representations*.

Zoph, Barret, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le (2018). "Learning Transferable Architectures for Scalable Image Recognition". In: *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*

Zuo, Yan and Tom Drummond (2017). "Fast Residual Forests: Rapid Ensemble Learning for Semantic Segmentation". In: *Proc. Conf. on Robot Learn.*