Efficient compression of semantic segmentation neural networks

Maciej Kowalski

MInf Project (Part 1) Report

Master of Informatics
School of Informatics
University of Edinburgh

2021

Abstract

Semantic segmentation is the process of assigning a label to every pixel in the image. Current state-of-the-art segmentation models becoming unnecessarily large and cumbersome. Thus, an evident need for compression emerges. The majority of literature on compression focuses on image classification, which is widespread in Computer Vision. This project explores the applicability of compression approaches designed for image classification in semantic segmentation.

For this purpose, the DeepLabV3+, a dominant semantic segmentation model, has been implemented to support further experiments. Depthwise separable convolution and bottleneck blocks were tested in the Atrous Spatial Pyramid Pooling module. Knowledge Distillation and Attention Transfer has been attempted with the compressed, cheaper variants of the models to restore their accuracy. Finally, a novel Compressed ASPP module has been studied, with a deeper but more efficient architecture.

The results suggest that distillation performs inadequately in pixel-wise classification. Conversely, "Cheaper convolution" blocks give promising results, with their efficient structure allowing for additional layers. The deeper architecture with Compressed ASPP achieves improved IoU with comparable compression.

Acknowledgements

First and foremost, I would like to thank Professor Amos Storkey for supervising my project, despite his decision not to take any supervisions this year. The ongoing support and guidance have allowed me to grow in a way I would not have believed beforehand.

I would also like to thank Wojtek Adamczyk and Karolina Konicka for proofreading my work and providing valuable insights and my parents for being very forgiving the last couple of months.

Lastly, a special thanks to Marcin Rybok. Your feedback, continuous discussions about Machine Learning and endless jokes have definitely made it easier to survive this crazy year.

Table of Contents

Chapter 1

Introduction

Tremendous advances in the field of Machine Learning, in particular in Deep Learning, have occurred in the last decade. The growth of new models' efficiency is outpacing the famous Moore's Law, with the number of floating-point operations required to achieve a similar performance as AlexNet [Krizhevsky et al., 2012] decrease by a factor of 44x between 2012 and 2019 [Hernandez and Brown, 2020].

Notwithstanding, the size and complexity of the models have grown even further. GPT-3 [Brown et al., 2020] has a record-breaking 175 billion parameters. The state-of-the-art Transformer-based models like BERT [Devlin et al., 2019], or ViT [Dosovitskiy et al., 2020] are now too large to be trained from scratch by most of the researchers. This trend has a negative effect on our environment, privacy, and the difficulty of research.

There are successful attempts at the compression of these large models such that they can be deployed to small but powerful mobile devices. Local computing has the benefit of being more secure, energy-efficient, and usually faster. However, smaller models tend to be substantially less accurate since they have worse generalisation ability and, unfortunately, because the majority of the research is directed towards making the models larger, not smaller.

1.1 Focus on semantic segmentation

This project's initial goal was simply the compression of neural networks, with semantic segmentation in mind. However, the project quickly took a different direction when further research into Computer Vision's compression techniques revealed the disparity between existing approaches directed at image classification and other tasks. For example, the widely popular Attention Transfer method in Computer Vision has not been created or studied with pixel-wise classification like semantic segmentation in mind. Generally, the prevalent assumption is that each discovery for the standard image classification will be easily transferable to other domains, which might not be accurate in the real world.

Therefore, this project's ultimate goal was to determine whether there are some intrinsic differences in how the semantic segmentation networks should be compressed.

1.2 Main contributions

- DeepLabV3+, the famous, state-of-the-art semantic segmentation model, has been adopted to support a range of compressed, more efficient networks.
- Two efficient, substitute convolutional blocks: depthwise separable convolution and bottleneck have been implemented and researched to compress the expensive ASPP block of the DeepLabV3+ model.
- The support for Kernel-Sharing Atrous Convolution has been added to the network's decoder module, following a recent paper.
- Knowledge Distillation along with the atypical Attention Transfer on the decoder module of the DeepLabV3+ has been implemented and studied.
- A novel, Compressed ASPP module has been proposed that achieves better accuracy than the naive compression methods.
- Finally, a comprehensive set of experiments has been performed to verify the assumptions on the compression techniques and provide data for further discussion.

Chapter 2

Background research

This chapter will cover the technical background of various Computer Vision tasks, focusing on image classification and image segmentation. After that, a brief introduction to Convolutional Neural Networks will be presented, alongside some of their key features. Having discussed the popular architectures, the focus will be on the efficiency of these implementations, more efficient network designs, and an overview of some of the popular approaches to achieve network compression and reduce computation costs.

2.1 Computer Vision

The classical Computer Vision (CV) has historically been focused on the careful extraction of features from the images with tools like edge or corner detectors [Canny, 1986, Harris and Stephens, 1988], or SIFT descriptors [Lowe, 2004]. Notwithstanding, Convolutional Neural Networks (CNN) have been used sporadically in various CV tasks like recognition of hand-written characters [LeCun et al., 1998] with great success for many years.

However, CNNs have not been the prevalent Computer Vision architecture until 2012, and the breakthrough achieved with AlexNet [Krizhevsky et al., 2012]. Using one of the first successful Deep Neural Networks with convolution layers (Figure 2.1), AlexNet achieved a spectacular accuracy boost in the ImageNet competition [Deng et al., 2009]. Having realised the potential of Deep Neural Networks with convolutional layers as local feature extractors, there has been an increase in the robustness and applicability of DNNs in Computer Vision and other tasks.

CNN's power as a flexible model that captures equivariant properties of visual data can be taken advantage of in other applications, even if designed for a different one in mind. However, despite all CV tasks tackling the problems connected with visual data and typically extracting the features in a similar fashion, their architecture can diverge substantially.

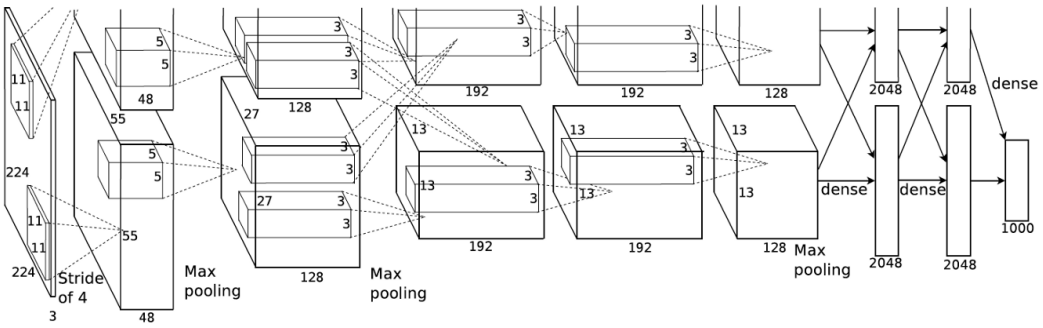

Figure 2.1: *Illustration of the CNN architecture used by AlexNet, [Krizhevsky et al., 2012].*

2.1.1 Image classification

Image classification is the most popular CV task that has become the benchmark for most Computer Vision models these days. This popularity is primarily due to the inherently straightforward problem that it answers - given an image that can consist of millions of pixels, what single class does the image belong to?

It has dominated the Computer Vision industry because it is challenging and yet easy when compared to other tasks. Typically, the findings can apply to other branches of Computer Vision as well. With the ImageNet competition, [Deng et al., 2009] a fierce rivalry has started among researchers to produce a model that can yield better accuracy. The achievements done by AlexNet, GoogLeNet [Szegedy et al., 2014] or ResNet [He et al., 2015], winners or ImageNet competition in 2012, 2014 and 2015, respectively, have been propagated to other fields of Machine Learning and are still used up to this day.

2.1.2 Image segmentation

Image segmentation is a pixel-level classification task where each pixel in the image has to be classified into one of the available classes. This task can be further described as partitioning the image into multiple segments, where each segment contains a collection of neighbouring pixels belonging to the same class.

Segmentation is typically split into two types: *semantic segmentation* and *instance segmentation*. The definition interpreted from [Arnab et al., 2018] describes them as follows:

Semantic segmentation *is the process of assigning a label to every pixel in the image, where multiple objects of the same class are treated as the same entity*

Instance segmentation *is the process of assigning a label to every pixel in the image, but here multiple objects of the same class are treated as distinct individual objects (or instances).*

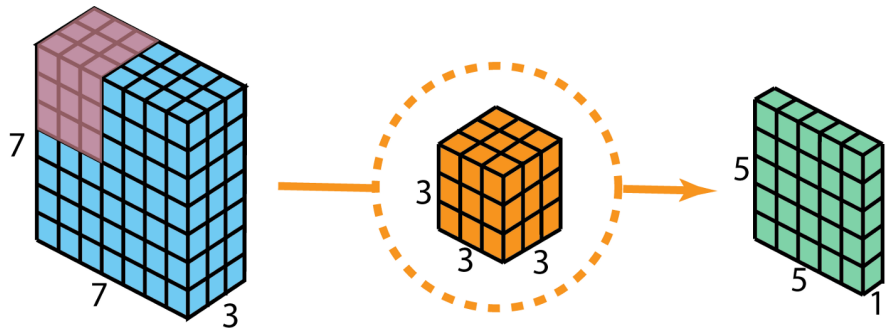

Figure 2.2: *Illustration of a single-kernel convolution. Image from [Bai, 2019].*

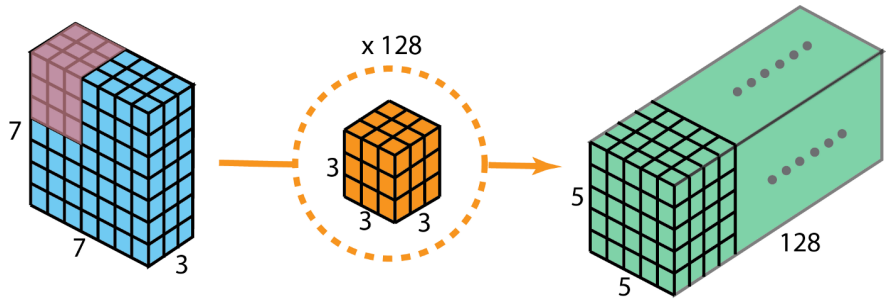

Figure 2.3: *Illustration of a convolution with 128 kernels. Each kernel corresponds with one output channel. Image from [Bai, 2019].*

Due to the additional difficulty of separating the segmented mask into distinct objects, instance segmentation is considered a much harder task than semantic segmentation. Instance segmentation models tend to share common architecture features with **object recognition** models like Faster R-CNN [Ren et al., 2016].

In segmentation, the output has to have the same resolution as the input and can represent multiple classes at once, with each class's location and shape clearly shown in the image. Hence, it is not enough for the model to compress the input data into a more abstract form and predict a final single-digit output, as do the image classification models. Therefore, segmentation models, despite sharing common DNN backbones with models designed for other tasks, are inherently different and require a more sophisticated approach.

In this project, only the semantic segmentation will be studied to reduce the research scope to a purely pixel-wise classification domain.

2.2 Convolutional Neural Networks

Convolution is an efficient way of describing transformations that apply the same linear transformation of a small local region across the entire input [Goodfellow et al., 2016]. Convolutional Neural Networks in Computer Vision take advantage of this property and apply convolution of several **kernels** to a region of an image (Figure 2.2 and 2.3), sliding this convolution window across the whole image. In CNNs, instead of all fully connected layers, convolution layers are used to extract more local, location invariant

features. The intrinsic locality of convolutions works well with pixel dependencies in images. Thanks to parameter-sharing, CNNs are also easier to train thanks to the lower amount of connections needed to capture the same local features compared to fully-connected networks. Convolution layers are usually combined with pooling layers, and Batch Normalisation [Ioffe and Szegedy, 2015].

The ability to abstract larger patches of data into more miniature representation induces the need for deeper rather than wider networks. It is one of the reasons why this feedforward class of neural networks has been a keystone architecture in modern Deep Neural Networks.

2.2.1 Residual Neural Networks

Deep CNNs tend to suffer from **vanishing or exploding gradient** problem, where the loss is not backpropagated correctly to lower layers. This obstacle can make it increasingly hard to train a network as it gets deeper, requiring more computations due to, i.e. lower learning rate and quick overfitting. Residual Neural Networks, mostly recognised by ResNet [He et al., 2015], introduce *residual blocks* which usually consist of 2 convolution layers with a ReLU [Nair and Hinton, 2010] activation function. The identity mapping from the first layer's input is added to the second layer's output, a piece-wise sum of which then goes through another ReLU function. This mapping could also be considered a "shortcut" connection where the additional connection skips a few layers. Residual connections between convolutional layers allow for easier backpropagation of gradient since the gradient can bypass convolution layers. The improved flow, in turn, results in quicker convergence and allows for deeper networks to be created.

In semantic segmentation, residual connections play a crucial role. With an even deeper architecture that is usually built on top of the ResNet backbone [Chen et al., 2017], the better gradient's propagation aids the extraction of the low-level features that help with proper contour predictions.

2.2.2 Pre-training

Large and deep neural networks like ResNet contain millions of parameters. Since the optimisers' goal in neural networks is to find the minimal loss and, hence, the global minimum of the underlying transformation functions, the parameters' initialisation plays a crucial role in achieving good performance [He et al., 2018]. The more parameters there are and the deeper the network, the more critical it is to initialise them to reasonable values to get an initial boost of performance. What is more, DNNs are known to be burdensome to train, even with the addition of residual connections. Pre-training helps alleviate both of these issues by allowing new models to share weights with a different model that has already been pre-trained on a different dataset or a different task, a useful property in the context of segmentation.

ResNets have become so popular because the ability to use pre-trained models has allowed these first, really deep neural networks to be used and adapted without the need to train them on other, larger datasets. In the case of semantic segmentation, the

models are typically built on top of a good performing classification model pre-trained on a large dataset like ImageNet, with only the last classification layers removed to fit the segmentation architecture [Chen et al., 2017, Chen et al., 2018].

In the context of this project, despite initially limiting the ability to change the backbone of the chosen model, pre-training offers higher accuracy and shorter training times, reducing the overall computation. This would not have been possible if all of the layers had to be trained from scratch.

2.2.3 Pooling and striding

Pooling layers are often used to reduce the spatial size of the feature representation and, thus, to reduce the number of parameters and computation in the network. To achieve this, a stride between pools is introduced, effectively allowing the models to extract the defining features of the larger representation and condense the information to a smaller format. The most popular pooling versions are max pooling, average pooling and their adaptive version - adaptive max pooling and adaptive average pooling.

Pooling helps make the representation approximately invariant to small translations of the input [Goodfellow et al., 2016]. The use of pooling can be viewed as adding an infinitely strong prior that the function the layer must learn. Invariance to local translation can be a valuable property if we *care more about whether a feature is present than where exactly it is*. This invariance is helpful in image classification where we are not interested in the location of the predicted class or its boundaries. However, in the case of segmentation, this is not a desirable effect. Therefore, pooling should be used only in strategic locations in the architecture, not as an afterthought taken from image classification.

Striding itself is not exclusive only to pooling layers. More generally, it is a method where the filter is not applied to each consecutive patch of data. Stride modifies the amount of movement over the input data, skipping a number of positions specified by the stride parameter. Stride s, in the case of Computer Vision, means that the filter gets applied to every s pixels. With $k > 2s$, where k is one of the dimensions of a square kernel, every single pixel is still considered during convolution, but there is less overlap between each patch of data.

The goal of striding is primarily to reduce the resolution of the output feature maps and allow for deeper network architectures. This reduction, in turn, results in a broader context of subsequent layers, which helps with better feature abstraction and limits the computational requirements. Smaller resolution can have, however, an undesirable effect in segmentation tasks [Chen et al., 2017] where the model struggles to translate these condensed features into clear segmentation boundaries.

2.2.4 Atrous convolution

Atrous convolution or dilated convolution allows the model to enlarge the field of view of filters to incorporate multi-scale context [Chen et al., 2017], without reducing the resolution as done by typical pooling layers or striding in convolution. The number of

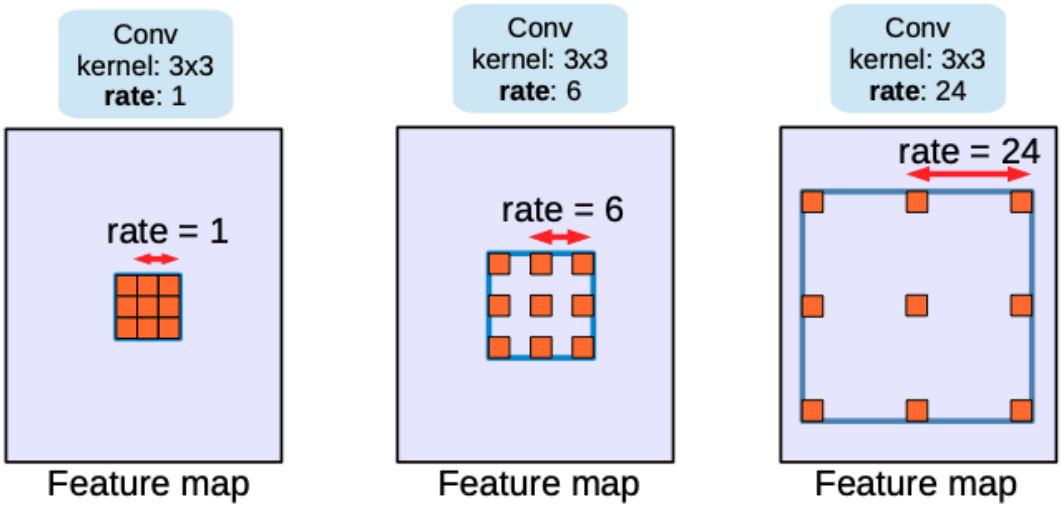

Figure 2.4: 3×3 *atrous convolution with the rates (1, 6, 24). Larger rates offer a wider receptive field while having the same number of parameters. Image taken from [Chen et al., 2017].*

weights in the filter remains the same, but with the increase in dilation rate, they can cover more area (Figure 2.4). Atrous convolutions are particularly useful in segmentation models as they help achieve a wider field of view without the need to do multiple, stacked convolutions or use large kernels.

For each location i on the output $\mathbf{y}$ and a filter $\mathbf{w}$, the atrous convolution is applied as in Equation 2.1, where r is the atrous rate.

$$\mathbf{y}[\mathbf{i}] = \sum_{\mathbf{k}} \mathbf{x}[\mathbf{i} + r * \mathbf{k}]\mathbf{w}[\mathbf{k}] \tag{2.1}$$

The atrous rate or dilation rate r corresponds to convolving the input $\mathbf{x}$ with upsampled filter produced by inserting $r-1$ zeros between the consecutive filter values along each dimension (Figure 2.4). Standard convolution is a special case where $r = 1$.

The exciting feature of atrous convolution is that it offers the benefits of the wider context while using the same number of parameters as the standard convolution. Therefore, it is possible, as in DeepLabV3 [Chen et al., 2017], to use a pre-trained ResNet50 model and change in the last layers the convolution with a stride to atrous convolution while still using pre-trained weights. However, the larger resolution causes the subsequent layers to process larger feature maps, effectively increasing the amount of computation by a large margin.

Moreover, atrous convolutions can also be used in parallel with a range of rates, as in Atrous Spatial Pyramid Pooling (ASPP) introduced by DeepLabV3 (Section 4.1).

2.2.5 Kernel-Sharing Atrous Convolution

Kernel-Sharing Atrous Convolution (KSAC), introduced by [Huang et al., 2019], takes a different approach to the atrous convolution in the Atrous Spatial Pyramid Pooling module, describer further in Section 4.1. Instead of using a separate kernel for each dilation rate, KSAC uses the same weights for all kernels being applied in parallel. It is possible since the atrous convolutions share the kernel size (i.e. 3×3, 5×5) but only differ in the applied dilation, as seen in Figure 2.4.

The addition of kernel-sharing helps improve the accuracy and reduce the number of parameters in the meantime [Huang et al., 2019]. In the paper, the author explains that a low atrous rate might be unable to extract features for a large object, while a large atrous rate can be ineffective in capturing local and more minor details. With the varying scale of the objects only a subset of the kernels can detect them. This limitation has an adverse effect on training, effectively reducing the number of training examples per kernel. With a shared kernel, despite having fewer parameters in total, all images can contribute to that layer's training since at least one of the atrous rates should activate.

2.3 Efficiency

Neural networks require sizeable resources, especially the fully-connected layers. Convolution has allowed to substantially reduce the number of operations and take advantage of the data's locality when extracting abstract features. However, the field of neural networks is growing so rapidly that any new, more efficient architectures eventually result in more parameters and layers added to the model rather than settling for the same accuracy with a more economic design. This phenomenon is known as the **rebound effect**[1] - the gains from a new, more efficient technology are usually lower than expected due to other changes diminishing the returns. Therefore, a more prominent focus should be put on the models' flexibility, with their size and complexity being easily scaled according to the needs.

2.3.1 Cost of networks

The cost of networks is becoming an important factor in their design. Recently, there is a significant movement of people noticing the impact of the models on the environment and our society. Training of neural networks requires lots of computation, and computation requires energy. Not all of the energy being used at the moment is carbon-neutral, with a large part of it still requiring the burning of fossil fuels.

The social impact of these is also worth noticing. Larger and larger models have, in recent years, become unattainable for most researchers. Not everyone can train a model on ImageNet from scratch, not to mention any of the recent Transformer-based models. The power has shifted unfavourably to large corporations that are able to run experiments on thousands of high-end GPUs for months.

[1] https://en.wikipedia.org/wiki/Rebound_effect_(conservation)

2.3.2 Efficient designs

A possible option is to take advantage of the research community's improvements while still being aware of the diminishing returns of some of the state-of-the-art models. Therefore, it is essential to look into more efficient designs of the popular models.

MobileNetV2 [Sandler et al., 2019] is based on the ResNet architecture but introduces a more efficient, inverted residual structure. In general, even with a different design, it is heavily inspired by the larger ResNet models.

A slightly different approach was proposed by [Tan and Le, 2020]. The EfficientNet architecture allows for uniformly scaling of the width, depth and input resolution with a single compound coefficient. With a single parameter, it is possible to control the overall complexity of the model. EfficientNet achieves state-of-the-art results while being smaller and faster than other best networks. However, with the being model designed for image classification, there is still room for improvement in other tasks like semantic segmentation.

2.3.3 Separable convolution

As large fully-connected layers are no longer commonplace, convolutions make up almost all of the parameters of modern networks. It is, therefore, desirable to make them smaller. [Crowley et al., 2019a] describes a concept of "Cheap Convolution", a group of convolutional blocks that can be introduced as a substitute of a standard convolution block in a network to reduce its memory footprint and inference time significantly. "Cheap Convolutions" will be studied extensively in this project.

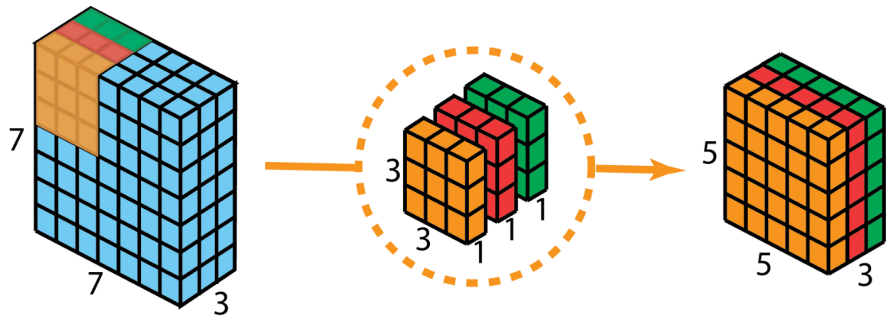

Figure 2.5: *Illustration of a depthwise convolution. The number of kernels corresponds to the number of channels. Image by [Bai, 2019].*

Depthwise separable convolution The standard convolutional layer uses N_{out} filters of $N_{in} \times k \times k$ shape, where N_{out} is the number of output channels, N_{in} is the number of input channels, and $k \times k$ is the kernel's size. This layer can be seen in Figure 2.3.

Depthwise convolutional layer uses just N_{in} kernels of $1 \times k \times k$ shape (Figure 2.5), where each kernel is applied to only one channel of the input. However, the number of output channels is N_{in}, not N_{out} as it was previously. Moreover, no information is being exchanged between the channels since the convolution's output is separated per channel. To accommodate for that, another 1×1 convolution is applied, called

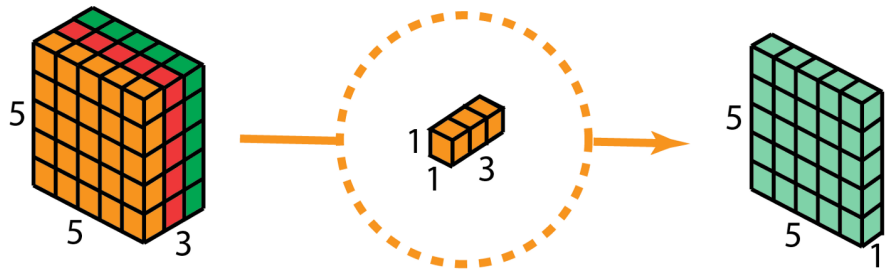

Figure 2.6: *Illustration of a* 1×1 *pointwise convolution. Three channels are mixed into one. Image by [Bai, 2019].*

pointwise convolution (Figure 2.6). The additional layer, shown in Figure 2.7, uses additional N_{out} kernels of $N_{in} \times 1 \times 1$ shape to produce the desired number of channels as in standard convolutional layer and allows for cross-channel information sharing [Crowley et al., 2019a].

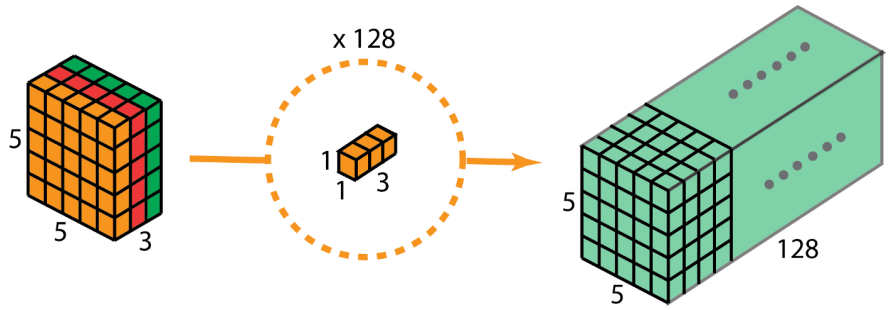

Figure 2.7: *The depthwise separated channels are mixed using 128 kernels to produce the expected 128 output channels. Image by [Bai, 2019].*

Grouped convolution Grouped convolution was introduced in the AlexNet paper [Krizhevsky et al., 2012]. The model required 3GB of GPU memory to fit, which in 2012 was impossible to satisfy. Therefore, the authors have used grouped convolution to split the model into two separate convolutional paths, as shown in Figure 2.1.

Grouped convolution is similar to depthwise separable convolution, although there are some key differences. Grouped convolution is parallelisable, with larger possible batch sizes. The output of the grouped convolution has already N_{out} channels (Figure 2.8), therefore no pointwise convolution is required to extend it from N_{in}, as in depthwise separable convolution. Moreover, each group can learn a separate representation of the data. However, without cross-channel mixing, this can also have adverse effects.

2.3.4 Bottleneck

In the original ResNet paper [He et al., 2015], the authors introduced a bottleneck block. The block's input has its channels decreased by a factor of b via a 1×1 pointwise convolution before a full 3×3 convolution is carried out. Finally, another 1×1 pointwise convolution brings the representation back up to the desired resolution. An illustration of this can be found in Figure [Wu and Lee, 2018].

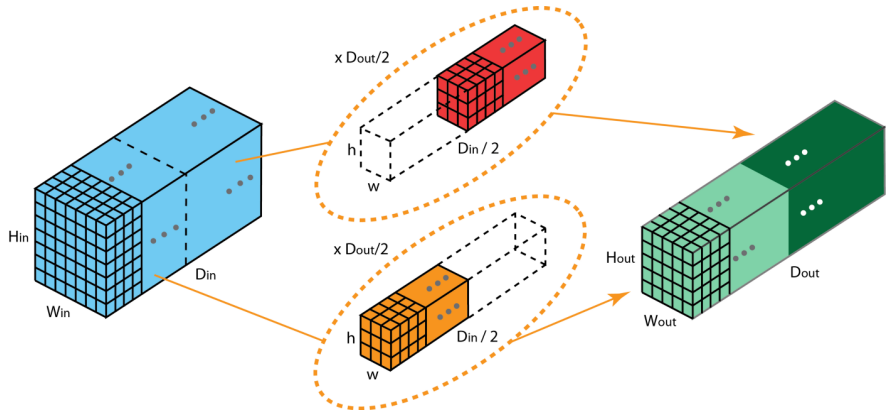

Figure 2.8: *Grouped convolution produces N_{out} channels in k separate, parallelisable operations. Illustration by [Bai, 2019].*

The 1×1 layers are used to reduce and then increase (restore) dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions. This combination reduces the number of parameters and computation, with only a slight drop in performance. Furthermore, with its three consecutive convolutional layers instead of one, the bottleneck's architecture is a prime candidate to use the residual *skip connections* that help with gradient propagation, hence the architecture of models like ResNet. [Sandler et al., 2019] uses inverted residual connections, where the skip connections are between the shrunken layers with an expanded block in between, further decreasing the number of parameters and computation required to achieve similar accuracy.

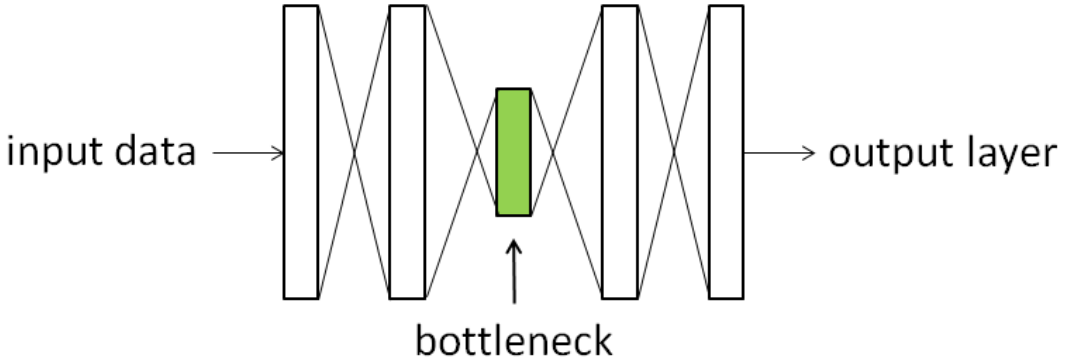

Figure 2.9: *Illustration of Bottleneck by [Wu and Lee, 2018]. The width of the layers is reduced in the bottleneck to perform the convolution on a smaller input and restored back to the original resolution afterwards, potentially saving parameters and computation in the process.*

2.4 Compression

The "Cheap Convolutions" mentioned in the previous section are a great example of a more efficient architecture. However, there are cases where it is more straightforward or necessary to **compress** an existing, successful architecture rather than designing a

new model from scratch.

2.4.1 Pruning

Pruning is a neural network compression method where a subset of weights is pruned from the network. The criteria for deciding which weights to remove creates two pruning approaches: **weight pruning** and **filter pruning**.

Weight pruning, or **unstructured pruning**, focuses on removing specific connections with weights below a certain threshold. [Han et al., 2015] introduced ℓ_2 regularisation to induce smaller weights further, improving the pruning potential. Such a simple strategy, performed several times iteratively, can achieve significant speed-ups but at the cost of generating a sparse model. Compression with weight pruning, although successful in reducing the number of parameters and hence the memory footprint, results in a sparse, non-structured model that cannot be used with popular deep learning libraries and widely available hardware due to insufficient data locality [Han et al., 2016, Luo et al., 2017, Crowley et al., 2019b]. Instead, specialised software and hardware are required, diminishing the benefits of such compression and increasing the cost of implementation.

While modern Convolutional Neural Networks contain a large variety of layers, many parameters and long inference time are spent on convolutional layers. Whole feature maps can be pruned from the network to remove a subset of layers least contributing to the final result, maintaining the original structure of CNNs.

Filter pruning, or **structured pruning**, is a pruning method where the whole channels are removed from the convolutional layers. The pruning yields an unchanged architecture that becomes *thinner* due to having fewer channels in each layer but has the same depth as the uncompressed network. Channel pruning is one of the most popular methods of accelerating over-parametrised CNNs by pruning [Luo et al., 2017, Chen et al., 2020] because it can be directly applied on any off-the-shelf hardware.

Notwithstanding, pruning is not as prevalent in recent years due to a relatively low transferability across different models. Pruning is also consistently beaten by reduced networks [Crowley et al., 2019b], further reducing its usefulness.

2.4.2 Knowledge distillation

A relatively different method of compressing neural networks is creating a smaller or more efficient *student* version of a successful model and transferring the accumulated knowledge from the *teacher*. Geoffrey Hinton compares distillation in Machine Learning to an insect transitioning from a "larval form that is optimized for extracting energy and nutrients from the environment and a completely different adult form that is optimized for the very different requirements of traveling and reproduction" [Hinton et al., 2015].

In Knowledge Distillation, students use a weighted average of two objective functions to calculate the loss for backpropagation. Typically, the first one is a standard cross-entropy loss with the correct labels. In contrast, the second one is a cross-entropy with

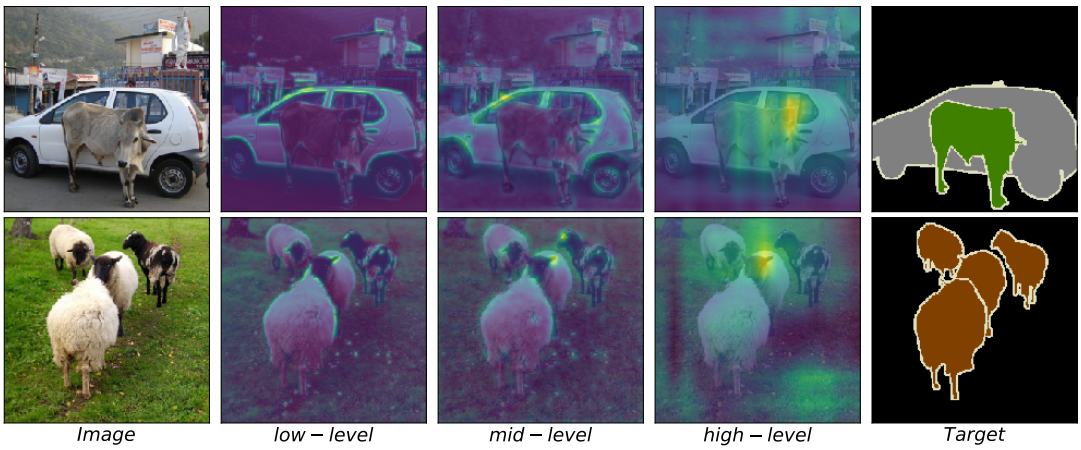

Figure 2.10: *Attention maps at different levels of a network. Low-level attention maps pay more attention to contours and small features, high-level attention happens more holistically and corresponds to the whole object.*

the soft targets - outputs of the teacher model, commonly produced using a `softmax` function. The idea behind it is that the soft labels can contain more information than the typical ground truth and allow the student to learn the teacher's pre-computed generalisations directly. The relative probabilities produced by the teacher tell us a lot more about the data compared to the raw labels. An image of a cat, for example, has a slight chance of being mistaken for a dog but is not nearly as likely to be mistaken for an apple.

[Crowley et al., 2019a] uses Knowledge Distillation in image classification to transfer the knowledge from a larger teacher model to a compressed, efficient student model with *Cheap Convolutions* in place of standard convolution. Thus, better accuracy is achieved by a student than the model being trained purely on the target labels.

2.4.3 Attention transfer

Knowledge Distillation focuses purely on the network's output, taking advantage of the additional information contained in the soft targets of the teacher models. However, it is also possible to extract useful information from the internal activations of the models. One of these methods is Attention Transfer.

[Zagoruyko and Komodakis, 2017] considers attention as a set of spatial maps that essentially represent which spatial areas of the input the network focuses on most when taking its output decision.

Typically, with the activation per layer being a 3-dimensional tensor, a normalised sum across the channel dimension is taken, outputting a 2D map of the activations. The equation 2.2 shows an example mapping F, where the sum can be raised to the power p, where $1 \leq p$ and C is the number of channels.

$$F^{p}_{sum}(A) = \sum_{i=1}^{C} |A_i|^p \tag{2.2}$$

The attention can be taken at various layers of the network, with lower layers having higher resolution and focusing on smaller features. The layers closer to the output tend to have a more holistic approach (Figure 2.10). To use this information, student models compare their attention maps with the more knowledgeable teachers and produce a cross-entropy loss used in the backpropagation phase.

Attention is relatively intuitive to explain in the case of image classification: what part of the image should we pay more attention to? It is interesting, however, how does that translate to semantic segmentation with a pixel-level classification task. Correctly predicting the mask of the object is just as important as the class prediction.

What is more, it is vital to choose the correct layers for the Attention Transfer. Thus, with the additional layers in most semantic segmentation architectures [Chen et al., 2017], it is possible that the attention in decoder layers (e.g. in ASPP [Chen et al., 2017]) can have more impact compared to the traditional approach of using the backbone layers.

Surprisingly, the literature on Attention Transfer focuses mainly on image classification, which does not have the additional classifier modules used in semantic segmentation. Therefore, it is a part of this project to determine whether attention is as valuable for segmentation as other tasks.

Chapter 3

Dataset and task

3.1 PASCAL VOC

The experiments are evaluated on the PASCAL VOC 2012 [Everingham et al., 2015] semantic segmentation dataset, which contains $1,464$ train, $1,449$ validation and $1,456$ test images. The images have a 513×513 resolution. Each image is labelled on a pixel level, with 20 foreground object classes and one common background class. The dataset is commonly augmented with the extra contour annotations [Hariharan et al., 2011] of the PASCAL VOC 2011 dataset and referred to as PASCAL VOC 2012aug. Therefore, the larger dataset is used for this project, yielding a total of $10,582$ training (*trainaug*) images.

Another popular dataset for semantic segmentation is the Cityscapes [Cordts et al., 2016] dataset, containing $5,000$ images in total, for training, validation and testing purposes, with an additional $20,000$ coarsely annotated images. However, due to the large size of each image (1024×510) and hardware limitations of the project, described further in Section 4.3, only the PASCAL VOC dataset has been studied.

3.2 Metrics

3.2.1 Intersection over Union

In image classification, the **accuracy** metric is used to report the model's performance. Given the output vector of $1 \times C$ shape, the predicted class is the one with the highest probability. The accuracy is the number of correctly predicted classes over the total number of predictions. However, this benchmark does not make much sense with semantic segmentation. We cannot simply count the number of the correctly predicted pixels and divide them by the total number of pixels in the image. This approach would favour the most significant class and not be size invariant. If the predicted object was small, the model could skip the class and still achieve good accuracy. Instead, we are interested in the **overlap** of the prediction or how much the predicted mask matches the ground truth.

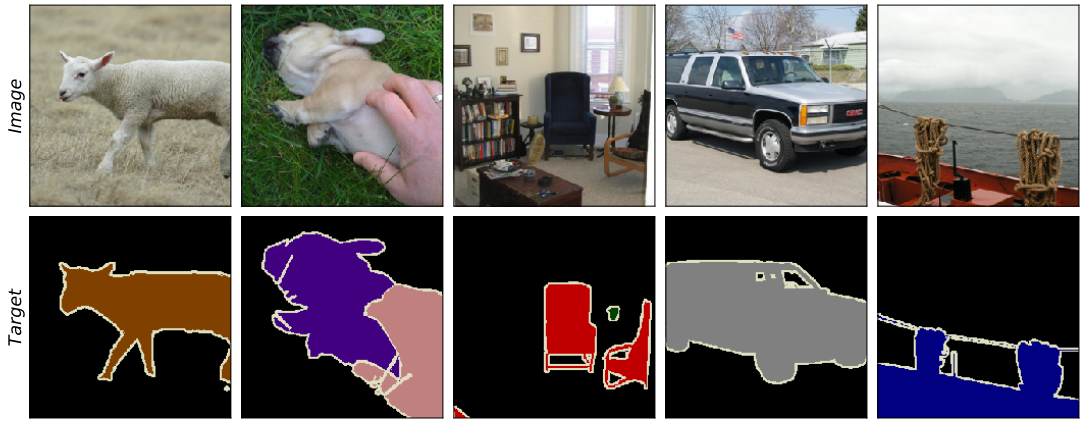

Figure 3.1: *Example images and ground truth masks from the PASCAL VOC 2012aug dataset. Black colour in the targets corresponds to the background and is also one of the 21 classes.*

The standard accuracy metric that is used by the research community to verify the performance of the model for segmentation is the Intersection over Union averaged across all classes or **mIoU** (Equation 3.1), where $0 \leq mIoU \leq 1$.

$$mIoU = \frac{\sum_{i=1}^{C} \frac{A_i \cap B_i}{A_i \cup B_i}}{C} \tag{3.1}$$

For each class i, the intersection $A_i \cap B_i$ indicates the number of shared pixels found both in the prediction mask and ground truth mask. The union $A_i \cup B_i$ indicates the number of the pixels found in either of the masks. The ratio of the intersection and the union represents how well the predicted mask overlaps the target mask - a too small prediction mask will lead to the intersection being small, a too-large prediction will lead to the union being large, making the mIoU smaller in both cases.

The calculated overlap for each class is averaged out over the number of classes C, yielding the mean IoU. The score can be balanced on the number of examples for each class, although typically, the unbalanced version is used [Heffels and Vanschoren, 2020], as seen in Equation 3.1.

It is common to report the IoU instead of mIoU, with $IoU = 100 * mIoU$. Thus, the IoU version will be used to report the results of the experiments.

3.2.2 Memory and computation

In the case of this project, the IoU performance is not the only important metric. With the rising size of a model, it is worth looking at the memory footprint and the computation amount needed for a single inference.

Due to the possibility of storing the parameters at lower bit widths than typical, the memory requirements are usually not represented in *bytes* but rather in the number of parameters.

Similarly, the model's speed is usually not represented by the inference time due to the wide range of hardware capabilities of each system. Moreover, even the same hardware and software setup can produce different results, depending on the system's utilisation. Therefore, a **Multiply-Add** (MAdd) metric is used to count the number of operations of the model. Note that, despite the Floating-Point-Operations (FLOPs) being commonly used in Computer Science, modern GPU architectures perform the multiplication and addition in one operation, as in $f = Ax + b$, whereas this takes two FLOPs in theory. Thus, the MAdd will be used to report the number of computations, as is usually done in Machine Learning.

Chapter 4

Methodology

4.1 DeepLabV3+

A baseline state-of-the-art semantic segmentation model is needed to compare the proposed approaches and methods of compression for semantic segmentation, provide a solid base for further exploration and expansion, and the results in terms of mIoU, parameters and MAdds. Therefore, DeepLabV3+ [Chen et al., 2018] has been selected as the base semantic segmentation model.

The family of DeepLab models for semantic segmentation has been one of the most popular architectures thanks to its straightforward architecture, based on a pre-trained backbone model like VGG [Simonyan and Zisserman, 2015] or ResNet and atrous convolution. Base DeepLabV3 [Chen et al., 2017] model uses ResNet50 as the encoder to extract visual features that later get decoded with an Atrous Spatial Pyramid Pooling (ASPP) module. The encoder extracts high-level visual features while multi-scale atrous convolutions, using their wide receptive field, help transform them into a segmentation map.

The original ASPP module contains four parallel convolutions, captured by curly brackets in Figure 4.1, with one 1×1 convolution and three 3×3 atrous convolutions with different atrous rates. To incorporate additional global-level features, crucial for accurate semantic segmentation, DeepLabV3 introduces global average pooling on the backbone's last layer, a 5^{th} operation in the ASPP. These five outputs in the ASPP module are then concatenated, passed through a final classifier module and upscaled to the final image resolution.

DeepLab introduces the notion of **output stride**, which signifies how much smaller the model's output resolution is. A typical ResNet model reduces the output resolution by 32x before it performs final classification. For example, using atrous convolution instead of stride of 2 in the last layer of the ResNet, DeepLabV3 achieves the same field-of-view while preserving the resolution of the second last backbone feature map. Therefore, by removing one stride operation, the output is 16x smaller than the image's resolution, yielding *output_stride=16*. Replacing further stride operations with atrous convolution can create the *output_stride* of 8, 4, and smaller, at the cost of slower

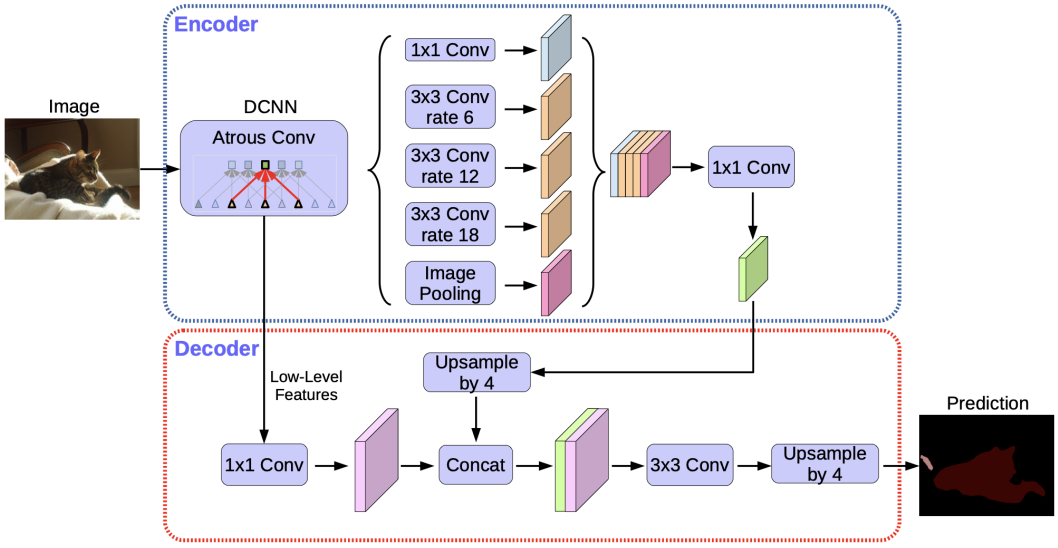

Figure 4.1: *High-level architecture of DeepLabV3+. The DCNN backbone extracts high and low-level features. The ASPP module and the decoder capture and combine the multi-scale information and detailed contours, which then gets upsampled in steps to the output resolution. Image taken from [Chen et al., 2018]*

computation.

DeepLabV3+ improves upon the encoder structure of its predecessor, DeepLabV3. Instead of using the naive method of bilinearly upsampling the output features to the original resolution of the image from the specified output stride, DeepLabV3+ first upsamples the output by a factor of 4. The upsampled features are then concatenated with the encoder's low-level features of the same spatial resolution and upsampled to the final shape (Figure 4.1).

4.1.1 Implementation details

The authors of DeepLabV3+ have provided a reference code implementation of the model in Tensorflow. However, PyTorch, a different framework for Machine Learning, was used for this project. Therefore, implementation has been based on the PyTorch code from [Fang, 2019], which contains the implementation for DeepLabV3 and DeepLabV3+ models with ResNet50, ResNet101 and MobileNetV2 as the optional backbones models.

Learning rate policy: A standard learning rate of 0.01 [Chen et al., 2017] has been used. However, as implemented by [Chen et al., 2017, Chen et al., 2018], the learning rate was decreased to 0.001 for the backbone when using pre-trained weights from ResNet or MobileNetV2. The lower learning rate for the encoder part ensures the model relies on the visual features extracted from the image and prevents it overfitting, thus promoting healthier learning of the classifier.

A multiplicative "poly"[1] learning rate scheduler is used for segmentation tasks as in [Chen et al., 2017, Chen et al., 2018]. In this scheme, the learning rate is updated on every iteration via the equation 4.1, where η_{init} stands for the learning rate at the beginning of training.

$$\eta_{iter} = \eta_{init} * \left(1 - \frac{iter}{max\_iter}\right)^{0.9} \tag{4.1}$$

$$max\_iter = N_{epochs} * N_{batches} \tag{4.2}$$

Data augmentation: Data augmentation is an industry standard that allows the models to train on more coherent yet diverse data. The setup used for data preprocessing consists of a applying normalisation with a mean [0.485, 0.456, 0.406] and std [0.229, 0.224, 0.225], a standard procedure for the PASCAL VOC2012aug [Chen et al., 2017].

Additionally, a random scaling of the input images by a factor in the range [0.5, 2] has been applied to each image, following by a random horizontal flip and a random crop of 256×256. The choice of a smaller random crop size is against the findings of [Chen et al., 2017] which showed that the large receptive field of the atrous convolution required a non-padded image. Hence, a smaller crop size will result in atrous convolution layers being applied to the zero-padded regions. However, an issue with the available hardware described further in Section 4.4 has influenced the decision to use a smaller crop size for efficiency reasons. This crop size effectively reduces the input resolution of the image to 256×256. Although this change of the resolution does not change the number of the parameters, since the filters' size does not change, it does change the overall footprint of the model during the training phase.

With a larger resolution, the filters are applied to a larger number of features, yielding a more significant number of function activations. All activations have to be stored in memory to perform backpropagation so that their gradient can be computed when propagating the loss backwards. Hence, a larger input size can easily cause the model to require excessive amounts of GPU memory. In the case of the DeepLabV3+ with 513×513 input size, at least 16GB of GPU memory is required, whereas the largest GPU widely available to Informatics students has 11GB of memory (RTX 2080Ti) while the GPUs dedicated to undergraduate students have only 6GB (GTX 1060). Notwithstanding, the outcomes of the experiments using a smaller crop size should be directly transferable to the larger, 513×513 crop sized used for PASCAL VOC in DeepLabV3.

4.2 Compression with "Cheap Convolutions"

Following the findings of [Crowley et al., 2019a], the implementation of DeepLabV3+ is extended to support "cheap convolution" blocks that can be put in place of any

[1]The scheduler has been called "poly" by the authors of DeepLabV3, even though the learning rate is not, in fact, a polynomial [Wu et al., 2019].

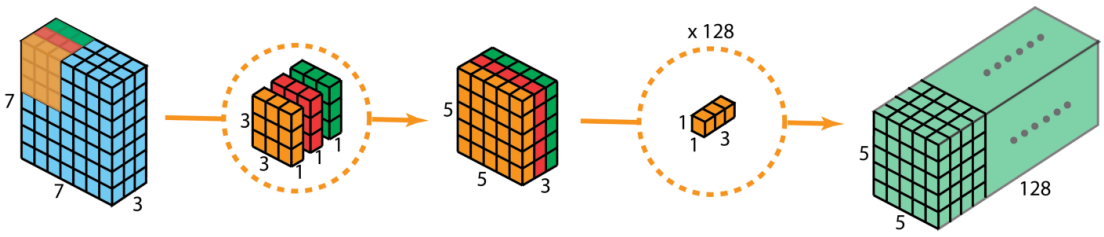

Figure 4.2: *Illustration of the depthwise separable convolution. The block of the two layers can be put in place of a standard convolution to achieve the same output shape with reduced resources. Image from [Bai, 2019].*

convolutional layers.

4.2.1 Depthwise separable convolution and bottleneck

For this project, instead of using the grouped convolution, only the depthwise separable convolution has been studied since the separable convolution has already been tested by [Chen et al., 2018] in their DeepLabV3+ model.

The number of kernels has been set to the number of input channels, N_{in}, following with the N_{out} 1×1 pointwise convolutions. The implementation is identical to the one used in DeepLabV3+, which can use the depthwise separable convolution in place of the 3×3 convolutions, but only in the added decoder module and ASPP.

Using the cheaper block in the ResNet backbone would remove the possibility of using pre-trained weights, reducing the expected benefits. Thus, only the segmentation-specific compression has been pursued in this project.

4.2.2 Knowledge Distillation with Attention Transfer

To verify the effect of distillation on semantic segmentation, Knowledge Distillation and separately Attention Transfer have been implemented.

The code for distillation has been adopted from the PyTorch implementation of the paper by [Crowley et al., 2019a].

Knowledge Distillation Let's assume we have a dataset of images, with one such image denoted as $\mathbf{x}$, where each element has a corresponding one-hot labelled image: denote the one-hot vector encoding of $\mathbf{x}$ as $\mathbf{y}$. Given $\mathbf{x}$, the teacher network $teacher(\mathbf{x}) = \mathbf{t}$ produces the prediction $\mathbf{t}$. Likewise, $student(\mathbf{x}) = \mathbf{s}$ outputs $\mathbf{s}$. Standard cross-entropy (Equation 4.3) is used to calculate the loss of the output.

$$\mathcal{L}_{CE} = -\sum_{k} p_k \log q_k \tag{4.3}$$

$\mathcal{L}_S$ (Equation 4.4) is the standard loss of the student when comparing the output $\mathbf{s}$ to the ground truth. $\mathcal{L}_T$ (Equation 4.5) is the loss between $\mathbf{s}$ and the output of the teacher

t and is minimised when the student network produces similar output to that of the teacher network.

$$\mathcal{L}_S = \mathcal{L}_{CE}(\mathbf{y}, \mathbf{s}) \tag{4.4}$$

$$\mathcal{L}_T = \mathcal{L}_{CE}\left(\frac{\mathbf{t}}{T}, \frac{\mathbf{s}}{T}\right) \tag{4.5}$$

To perform Knowledge Distillation, the student network is trained to minimise the $\mathcal{L}_{KD}$ loss function, shown by Equation 4.6.

$$\mathcal{L}_{KD} = (1-\alpha)\mathcal{L}_S + 2\alpha T^2 \mathcal{L}_T \tag{4.6}$$

The value of α is controlling the ratio of the two losses, while temperature T controls the distillation process such that the teacher emits less extreme probabilities, which are more informative for the students [Wild, 2018].

For the experiments, α has been set to 0.9 while the temperature T was set to 4, following the implementation by [Crowley et al., 2019a].

Attention transfer For the choice of N_L layers i in the teacher network, corresponding N_L layers are chosen from the student network. If it is impossible to find layers with the matching size due to, e.g. different output stride, the teacher layers are bilinearly scaled to match the resolution of student layers.

At each chosen layer i, the spatial map of the activations is calculated using the adopted version of the Equation 2.2, described in Section 2.4.3. In particular, Equation 4.7 is used, recommended by [Zagoruyko and Komodakis, 2017, Crowley et al., 2019a].

$$\mathbf{f}(A_i) = \left(\frac{1}{N_{A_i}}\right) \sum_{j=1}^{N_{A_i}} \mathbf{a}_{ij}^2 \tag{4.7}$$

Similarly to Knowledge Distillation, in Attention Transfer the student network is trained to minimise the $\mathcal{L}_{AT}$ loss function (averaged across all data items), shown by Equation 4.9. $\mathcal{L}_S$ is the same cross-entropy loss as in Equation 4.4, while $\mathcal{L}_A$ (Equation 4.8) is the normalised loss between the N_L attention layers of the teacher and student. Analogous to KD, the $\mathcal{L}_A$ is minimised when the student network produces similar attention maps to these of the teacher.

$$\mathcal{L}_A = \sum_{i=1}^{N_L} \left\| \frac{\mathbf{f}(A_i^t)}{\left\|\mathbf{f}(A_i^t)\right\|_2} - \frac{\mathbf{f}(A_i^s)}{\left\|\mathbf{f}(A_i^s)\right\|_2} \right\|_2 \tag{4.8}$$

$$\mathcal{L}_{AT} = \mathcal{L}_S + \beta \mathcal{L}_A \tag{4.9}$$

The hyper-parameter β controls the ratio of the two losses and has been set to 1000, based on the choice by [Crowley et al., 2019a].

4.2.3 Compressed Atrous Spatial Pyramid Pooling

The popular ASPP module has relatively few layers compared to the encoders like ResNet50 or MobileNetV2. In particular, only one large convolution layer is used for each of the three parallel 3×3 atrous convolutions that are supposed to extract the segmentation details from the encoder's output features. However, the ASPP module can be responsible for up to 50% of the total parameters and about 25% of the MAdds in a standard DeepLabV3+ model. Thus, it is difficult to add any more layers without drastically increasing the size, at which point it might be more efficient to use a larger encoder like ResNet101 or Xception-65.

The novel Compressed Atrous Spatial Pyramid Pooling (CASPP) is proposed with additional bottleneck layers for each atrous rate, taking advantage of the bottleneck block's significant compression factor. Thanks to the efficient convolution operations in the bottleneck module, it is possible to add more layers while still having a substantially lower number of parameters and MAdds.

With ResNet50, the number of output channels N_{out} is equal to 2048. Each atrous convolution with a rate r outputs 256 channels, with the outputs being concatenated together. CASPP performs this reduction of channels in a more gradual fashion, with more than one convolutions. CASPP-2, CASPP-3 and CASPP-4 have 2, 3, and 4 bottleneck layers, respectively. Since they use bottlenecks, the number of parameters and MAdds is still lower than in the one standard convolutional block.

The CASPP blocks are defined in Section 5.3.1, with their performance discussed in Section 5.3.

4.3 Memory and computation

The memory usage, represented by the number of parameters, and computation, represented by MAdds, were calculated using the `flops-counter.pytorch` framework [Sovrasov, 2020] for PyTorch. The library, given the model and an example input size, calculates the total number of parameters and multiply-adds per layer in CNN recursively. The results are shown per layer as in Figure 4.3, allowing for a thorough analysis of the model's complexity and opportunities for optimisation and improvement.

4.4 Cluster computing

A single experiment on a CPU can take days, if not weeks, to train. Using the parallelisation technique available on mainstream deep learning frameworks like PyTorch or Tensorflow, the experiment's training can be run on popular and widely available GPUs with CUDA cores or, better designed for these tasks, TPUs. These allow us to

```
(classifier): DeepLabHeadV3Plus(
  2.437 M, 57.358% Params, 0.773 GMac, 51.939% MACs,
  (project): Sequential(
    0.001 M, 0.029% Params, 0.005 GMac, 0.357% MACs,
    (0): Conv2d(0.001 M, 0.027% Params, 0.005 GMac, 0.317% MACs, 24, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (1): BatchNorm2d(0.0 M, 0.002% Params, 0.0 GMac, 0.026% MACs, 48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(0.0 M, 0.000% Params, 0.0 GMac, 0.013% MACs, inplace=True)
  )
  (aspp): ASPP(
    2.393 M, 56.326% Params, 0.592 GMac, 39.789% MACs,
    (convs): ModuleList(
      2.065 M, 48.601% Params, 0.508 GMac, 34.138% MACs,
      (0): Sequential(
        0.082 M, 1.940% Params, 0.021 GMac, 1.423% MACs,
        (0): Conv2d(0.082 M, 1.928% Params, 0.021 GMac, 1.409% MACs, 320, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(0.001 M, 0.012% Params, 0.0 GMac, 0.009% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(0.0 M, 0.000% Params, 0.0 GMac, 0.004% MACs, inplace=True)
      )
      (1): ASPPConv(
```

Figure 4.3: *A cutting of the output of the `flops-counter.pytorch` library. The number of parameters and MAdds (MACs) is displayed per layer and its nested children.*

take advantage of the parallel nature of the computation in convolutional neural networks and speed up the training by a couple of magnitudes.

However, with Computer Vision experiments and large models like DeepLabV3+, it is still very cumbersome and time-consuming to run the experiments one by one, even on the fastest GPUs available. Using a single GTX 1060, available on the Informatics GPGPU Teaching Cluster[2], we can train a baseline DeepLabV3 model, although with a limited crop size to fit the model into the GPU's memory, in around 5 hours. Therefore, performing a single experiment with a range of parameters to compare and fine-tune them to achieve the best validation IoU, even using a powerful GPU, can take days to finish, assuming a non-stop access to the hardware.

However, the Informatics cluster allows us also to run a batch of experiments in parallel. Using the Slurm[3] scheduler installed on the cluster, we can request multiple nodes with a specified CPU, GPU and memory configuration, and the scheduler will put our tasks in the queue.

Therefore, a batch of 20 experiments, given a low utilisation of the cluster and, therefore, shorter queue times and more GPUs available to the user, can be accomplished under 10h.

4.4.1 Scheduling experiments

Multiple steps are required to streamline the process of running a batch of experiments on the Informatics cluster and ensure efficient resource utilisation and data safety. This part has been crucial for any progress of this project; many hours had been spent optimising this pipeline, ensuring a good quality of results can be achieved and adequately analysed.

Generate experiment configurations An experiment-generating script[4] has been adopted and extended to schedule a number of experiments efficiently. The script generates a list of commands in a `.txt` format from a dictionary where the keys are

[2] `https://computing.help.inf.ed.ac.uk/teaching-cluster`

[3] `https://slurm.schedmd.com/overview.html`

[4] `https://github.com/cdt-data-science/cluster-scripts`

parameter names, and the values are lists of the variants of the parameters for the experiment. Each command would be then launched on a separate task with a dedicated GPU by another script.

The code directory in a `.zip` format, along with the scrips mentioned above and the experiment results are added to a separate folder to efficiently track the changes of the code and be able to replicate the experiments easily.

Data preparation Before the training process, the dataset has to be downloaded and pre-processed on the cluster's head node to extract the target segmentation maps. With the existing cluster configuration, only the primary node has access to the Internet. Even though all of the nodes are connected by a distributed file system, the large quantities of data being loaded during each experiment can put excessive load on the servers and slow down the jobs. Therefore, it is recommended that the data is loaded from each node's local scratch disk. To achieve that, the dataset has to be transferred to the machine running the experiment before starting the job. On each node, the startup script confirms if the data is present on the scratch disk using a `rsync` command, copying only the missing data from the head node.

Checkpoints and intermediate results A standard procedure when training large models is creating checkpoints of the progress as a backup solution in case the training gets interrupted. This approach is critical on a cluster, where the process of resource allocation is outside of our control. In this project, checkpoints have also been used to store the teacher models - checkpoints can be loaded and used for distillation.

With a large number of experiments and the large size of the saved models, checkpoints have been stored on the scratch disks of the nodes and only transferred to the head node after the completion of the experiment, limiting the impact on other users of the excessive data transfer that would occur otherwise.

Along with checkpoints, each model outputs validation scores during the training phase. The validation mIoU has been scored in a `.csv` file after each epoch per experiment.

Chapter 5

Experiments

In this section, several teacher networks are trained and evaluated. A comparison between the standard ASPP module and the KSAC module is shown. After creating the baseline results, the best model is chosen as the teacher for further experiments. Student networks are distilled using Knowledge Distillation or Attention Transfer. For Attention Transfer, four different variants are used. For comparison, a compressed variation of the ASPP module, CASPP, is used to verify whether the compression can be achieved without compromising the model's performance. Additional bottleneck layers are added to the parallel atrous convolution in the CASPP modules to take advantage of the layers' reduced size.

To combat the stochasticity of the validation results, a common approach is to report an average of k experiments. However, performing the distillation in its standard format only takes a single teacher's output to calculate the loss. This poses the following question: which trained model should be used for distillation if an average is taken for comparison?

It was concluded that a more fair approach is to repeat each experiment and consider the better performing model. Therefore, all experiments have been performed twice, and instead of taking a mean of the mIoU, the model with higher mIoU has been used for distillation and further analysis.

5.1 Baseline experiments

A comprehensive set of experiments has been performed to demonstrate the capabilities of the teacher networks. Although these are mainly to provide a reference point when comparing students networks in distillation, the outcomes provide helpful information about the nature of compression when training smaller models.

The results of "cheaper convolution" blocks in the ASPP module are compared to the standard convolution, alongside the effect of using Kernel-Sharing Atrous Convolution or a smaller output stride of 32. The comparison is shown in the Table 5.1 for ResNet50 and Table A.1 for MobileNetV2.

Architecture					Metrics		
OS	KSAC	ST	DS	BTN	IoU	Params	MAdds
16	✓	✓			74.38	30.32M	14.94G
16		✓			73.99	39.63M	14.94G
16	✓		✓		73.81	25.53M	9.21G
16			✓		73.88	26.61M	9.21G
16	✓			✓	73.48	25.09M	8.72G
16				✓	73.21	25.39M	8.72G
32	✓	✓			73.14	30.32M	9.29G
32		✓			72.89	39.63M	9.29G
32	✓		✓		72.21	25.53M	5.95G
32			✓		72.26	26.61M	5.95G
32	✓			✓	71.75	25.09M	5.69G
32				✓	71.63	25.39M	5.69G

Table 5.1: Results of the DeepLabV3+ models with the ResNet50 backbone. Column names explained in Section 5.1.1.

5.1.1 Model parameter descriptions

- **OS** - Output stride of the model.
- **KSAC** - Kernel-Sharing Atrous Convolution in the ASPP module.
- **ST** - Standard convolution blocks used in the ASPP module.
- **DS** - Depthwise separable convolution blocks used in the ASPP module.
- **BTN** - Bottleneck blocks used in the ASPP module.

5.1.2 Baseline results

As can be derived from the tables, Kernel-Sharing Atrous Convolution achieves a substantial memory compression in all cases. However, the gain is smaller when a compressed block is used for each atrous rate. This finding aligns with the implementation details - kernel-sharing uses just one kernel for all rates, reducing the number of parameters for the atrous convolution by a factor of 3x in the case of 3 rates. With compressed kernels, the gain is lower in absolute terms.

In the case of ResNet50 backbone (Table 5.1), with KSAC, the IoU improves substantially for the standard ASPP encoder while staying approximately the same for compressed atrous convolutions, indicating that a smaller number of kernel-sharing parameters might be lacking some of the generalisation and representation power of the larger, non-compressed variants.

Surprisingly, with MobileNetV2 (Table A.1), the KSAC approach performs consistently worse than the standard counterpart. This finding could be attributed to the relatively small size of MobileNetV2. With a smaller number of parameters from the

start, the further reduction of the decoder can cause heavily undesirable effects on the accuracy.

The models with an output stride of 16 perform consistently better than their counterparts with output stride 32 while having substantially more MAdds and the same number of parameters. This result confirms the findings of [Chen et al., 2017] and [Chen et al., 2018], which treat the output stride as a variable that controls the amount of computation.

A promising result for compressed convolution blocks can be noticed when comparing a model with ResNet50 backbone, OS=32 and models with OS=16 and the separable convolution or bottleneck (Table 5.1). The variants with compressed ASPP and OS=16 show better performance while having fewer parameters and MAdds than a standard DeepLabV3+, with or without KSAC and OS=32. This important discovery can be approached in two ways:

- The necessity of more severe upscaling in the case of OS=32 is dwindling the performance more than using compressed convolution. More efficient convolution structures have similar generalisation capability, but with OS=16, they are not limited by the low resolution of the features.
- The naive method of controlling the model's output stride, introduced in the DeepLabV3 paper, is not a method of compression but rather an afterthought solution to control the excessive computation required by low output strides and, therefore, large resolutions. The original paper used OS=4 to achieve state-of-the-art mIoU at that time. However, even an OS=8 is not a viable option for most hardware and applications, with OS=8 using 2.9x (43.33G) while OS=4 using absurd 10.7x (160.94G) more MAdds when compared to a baseline model with OS=16.

Possibly, the introduction of "cheap convolutions" has allowed for a successful compression without unnecessarily impairing the prediction capability. Therefore, similarly to [Crowley et al., 2019a] it is concluded that the use of efficient configurations should be considered further, potentially incorporating deeper a structure of the convolutions with additional, "cheap" layers that could help recover from the accuracy drop.

Lastly, the accuracy of the depthwise separable convolution is generally better than the one of the bottleneck in all cases, at the cost of being minimally less compressed. This finding will be further studied in the next section.

5.2 Knowledge Distillation and Attention Transfer

The checkpoint from the best model from baseline, with the IoU of 74.38, has been chosen for distillation. During the distillation procedure, the teacher model was loaded from the checkpoint with original settings. A separate student model has been initialised with the student-specific settings. During training, each batch of images would be passed through both networks and their predictions combined to calculate the student's loss.

Architecture						AT			KD
OS	KSAC	ST	DS	BTN	None	Output	Atrous	All	KD
16	✓	✓			74.38	**74.45**	74.16	74.33	70.46
16		✓			73.99	**74.36**	74.18	74.28	70.39
16	✓		✓		73.81	**74.05**	73.90	73.96	68.25
16			✓		73.88	74.12	74.08	**74.22**	68.14
16	✓			✓	73.48	73.49	**73.63**	73.57	68.45
16				✓	73.21	73.49	73.71	**73.91**	68.15
32	✓	✓			**73.14**	72.04	72.78	72.69	70.46
32		✓			72.89	72.12	**73.08**	72.84	70.39
32	✓		✓		72.21	72.00	**72.57**	72.45	68.25
32			✓		72.26	72.05	**72.37**	72.36	68.14
32	✓			✓	71.75	71.81	**72.02**	71.72	68.45
32				✓	71.63	71.66	**72.25**	72.07	68.15

Table 5.2: The results of the DeepLabV3+ student models with the ResNet50 backbone. Column names explained in Section 5.2.1.

To verify whether it is possible to improve the compressed models' accuracy from Section 5.1, only the best teacher model has been taken and distilled into the smaller architectures. The teacher model was a DeepLabV3+ with the ResNet50 in the backbone, with an output stride of 16, Kernel-Sharing Atrous Convolution and standard ASPP architecture.

The same models from Section 5.1 have been used for the student models to see the impact of distillation on them.

5.2.1 Distillation method descriptions

- **None** - Standard training with no distillation, results from Table 5.1.
- **Output** - Attention Transfer with the attention taken from the output of the ASPP module.
- **Atrous** - Attention Transfer with the attention taken from the outputs of each atrous convolution in the ASPP module.
- **All** - Attention Transfer with the attention taken from the outputs of each atrous convolution in the ASPP module and the output of the ASPP module itself.
- **KD** - Knowledge Distillation with the second loss from the soft targets of the teacher (Equation 4.6).

The description of the architecture details is the same as in Section 5.1.1.

Architecture						AT			KD
OS	KSAC	ST	DS	BTN	None	Output	Atrous	All	KD
16	✓	✓			69.17	69.19	69.38	**69.55**	65.84
16		✓			**69.34**	69.32	69.24	69.28	65.83
16	✓		✓		67.73	67.65	**68.32**	67.82	64.34
16			✓		68.12	67.79	**68.20**	67.77	64.12
16	✓			✓	67.64	67.94	68.42	**68.74**	63.91
16				✓	68.15	67.65	**68.23**	68.08	64.02
32	✓	✓			**68.00**	66.99	66.92	67.26	64.89
32		✓			**68.20**	66.47	67.29	67.36	64.99
32	✓		✓		**66.38**	65.60	66.37	66.20	63.12
32			✓		66.44	**65.45**	66.04	66.17	62.92
32	✓			✓	**66.31**	65.64	66.12	66.27	62.59
32				✓	66.41	65.97	66.32	**66.55**	62.78

Table 5.3: The results of the DeepLabV3+ student models with the MobileNetV2 backbone. Column names explained in Section 5.2.1.

5.2.2 Distillation results

Table 5.2 and 5.3 compare the different approaches of distillation for student networks. From the start, it is easy to notice that the Knowledge Distillation performs significantly worse than any other approach. The results confirm the initial assumptions that KD does not work for semantic segmentation even though it achieves good results in image classification [Crowley et al., 2019a, Hinton et al., 2015]. The change of the domain from standard classification to pixel-level classification questions the underlying reasoning for the benefits and applications of Knowledge Distillation.

Possibly, with the teacher having mIoU that is far from the state-of-the-art, the students are trying to optimise for a mediocre prediction that ultimately might be confusing them instead of directing towards a good solution. Surprisingly, even the student with the same architecture as the teacher performs noticeably worse. This result goes against the findings of [Kim et al., 2020, Zhang et al., 2019], which show that Self-Distillation can improve a model's performance.

The story of Attention Transfer is more complicated. The larger, ResNet50-based model can take advantage of the attention and generally performs better with it than without it. However, smaller models with MobileNetV2 cannot make any substantial improvements over the model trained from scratch. Interestingly, with a lower output stride of 32, there are also adverse effects of the distillation. Acknowledging the results from the previous section, the smaller backbone of the MobileNetV2 combined with downsampling the attention maps to match student's resolution has possibly prevented the model from learning an optimal representation of the images.

Three different types of Attention Transfer have been tested in this work. All of them are concerning the layers of the ASPP decoder module, unique to the segmentation problem. The single `output` map, taken from the ASPP module's output, performs

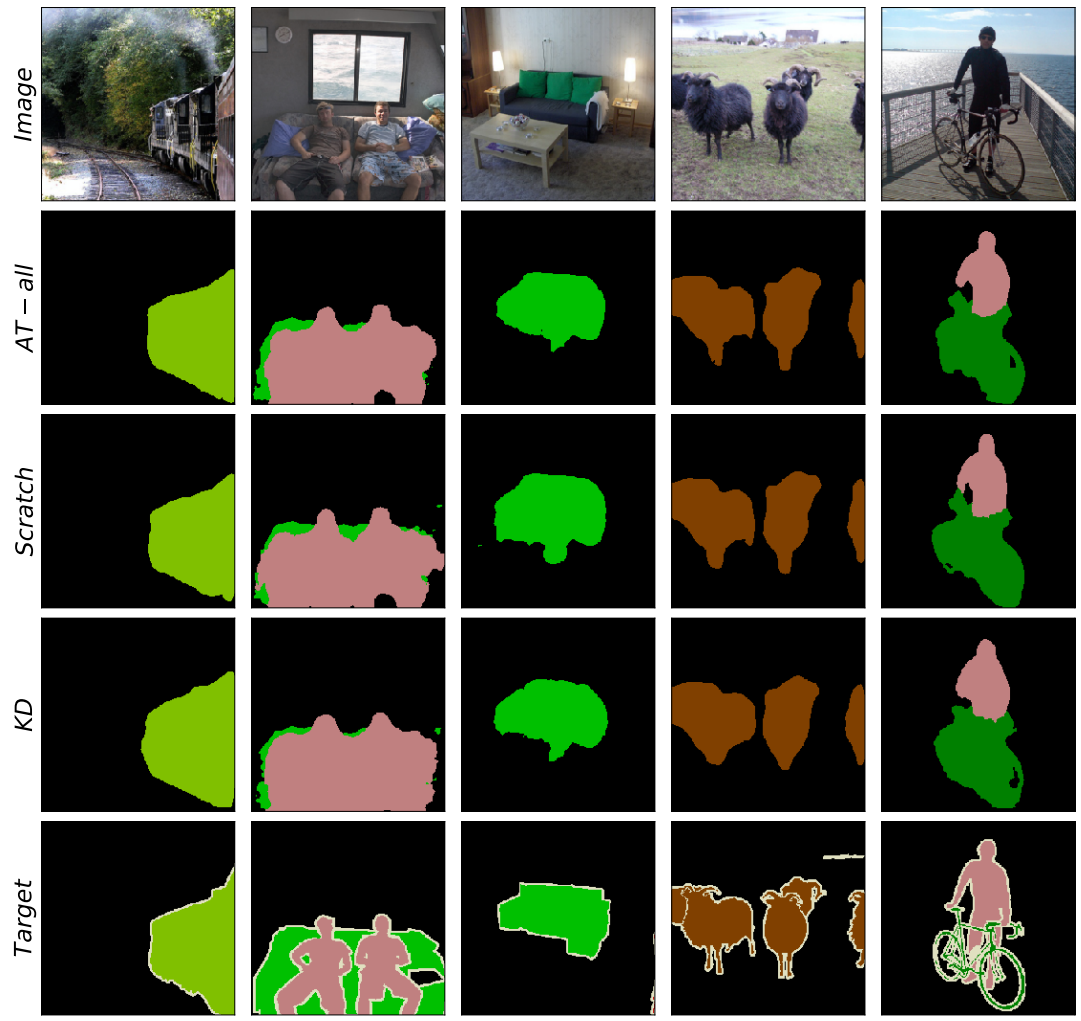

Figure 5.1: *Example predictions of the DeepLabV3+ model with the ResNet50 backbone. All models use non-shared kernels and depthwise separable convolution. Row 2 shows predictions of the AT-all student, Row 3 the teacher model without distillation, and Row 4 shows the least accurate student trained with KD.*

worse than other solutions. This could be explained by the limited number of layers (only one) and the fact that this is one of the last layers, which combines the outputs of atrous convolution. Theoretically, each rate of the dilated convolutions is responsible for retrieving features of a different scale. Thus, the attention of the combined layers is less detailed than the one of the atrous layers, which can be noticed on the visualisation of the attention maps in Figure 5.2.

The `atrous` and `all` variants are similar, with an almost equal split of best mIoU results. It is impossible to tell from the collected data which performs better. Thus, more experiments have to be performed to make a more probable conclusion. However, the two approaches share the attention maps of the atrous convolution layers, which seem to help the ASPP module learn optimal weights.

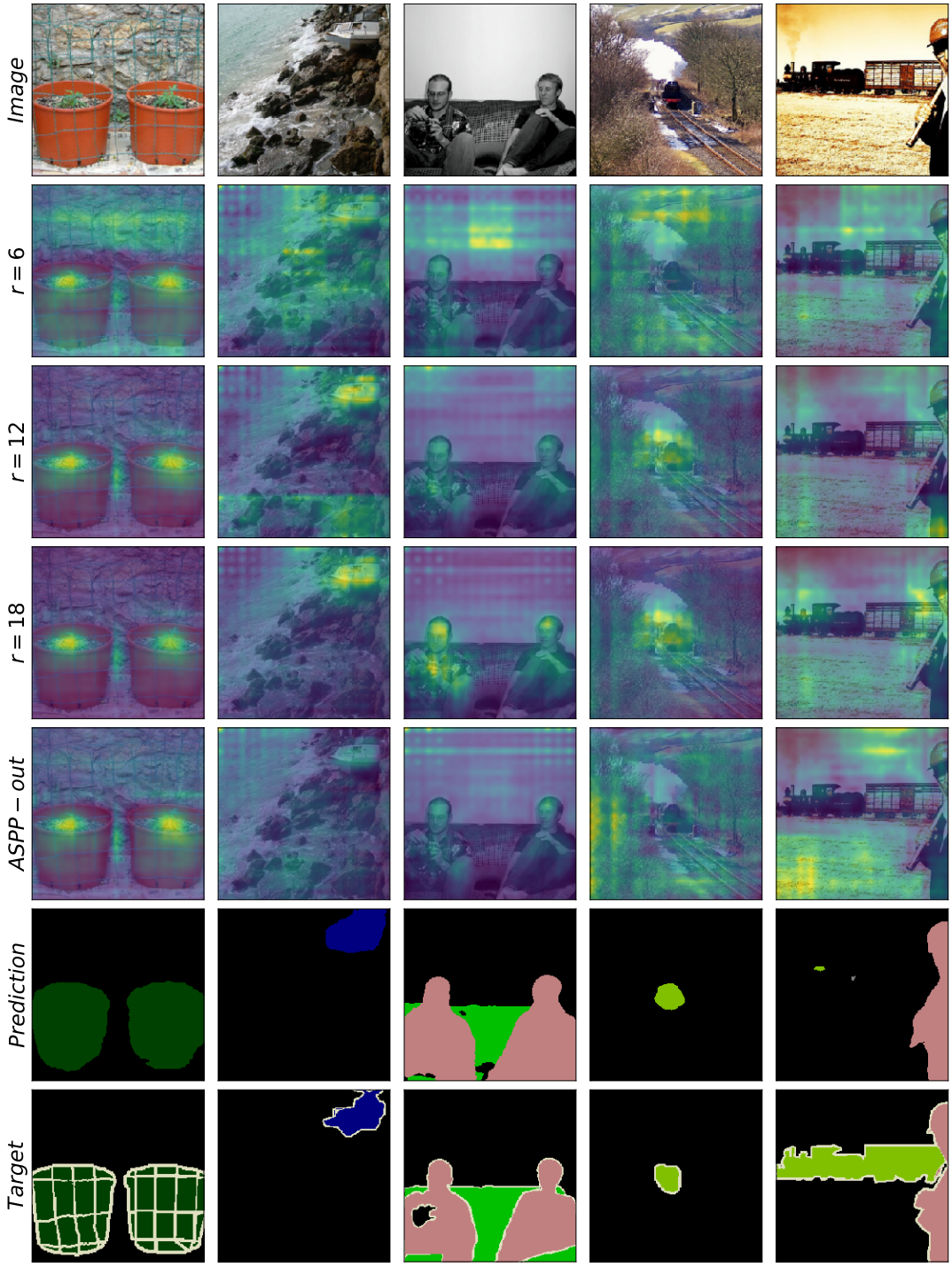

Figure 5.2: *Spatial maps of the attention of the best baseline DeepLabV3+ model with ResNet50 backbone. Row 2, 3 and 4 represent the attention of the atrous convolution with corresponding rate. ASPP-out represents the attention of the ASPP module output. Attention of atrous convolutions tends to activate near the correct parts of the image while the combined ASPP output can lose the focus.*

5.3 Compressed ASPP

Section 5.1 focused on compression methods by swapping the standard convolutional blocks for their cheaper counterparts. The swapped blocks were in the DeepLabV3+ ASPP encoder module, taking advantage of the unchanged, pre-trained backbone. In this section, with the encouraging results of the cheaper convolutions in ASPP, a crucial element of the DeepLabV3+, three augmented variants are evaluated further.

5.3.1 ASPP module descriptions

- **ASPP** - Standard atrous convolutions with just one layer, used in Section 5.1.
- **CASPP-1** - Compressed atrous convolutions with just one bottleneck layer, used in Section 5.1.
- **CASPP-2** - Compressed atrous convolution with *two* convolutional layers. The first bottleneck reduces the number of output channels to $2 \times N_{out}$. The second reduces the output channels further to N_{out}.
- **CASPP-3** - Compressed atrous convolutions with *three* convolutional layers. The first bottleneck reduces the number of output channels to $4 \times N_{out}$. The second reduces the output channels further to $2 \times N_{out}$. The last layer reduces the output channels to the final N_{out}.
- **CASPP-4** - Compressed atrous convolutions with *four* convolutional layers. The first bottleneck reduces the number of output channels to $4 \times N_{out}$. The second reduces the output channels further to $2 \times N_{out}$. The third layer reduces the output channels to the final N_{out} while the last layer is an additional bottleneck with N_{out} output channels.

5.3.2 Compressed ASPP results

As shown in Table 5.4 and Table A.2, all models with Compressed ASPP have significantly fewer parameters and MAdds than a traditional ASPP, despite potentially having three extra layers per atrous convolution.

CASPP-1, the model with just one bottleneck, described in Section 4.2, in all configurations, performs undoubtedly worse than the original DeepLabV3+ model but achieves the best compression ratio.

The CASPP-2, CASPP-3 and CASPP-4 models, with 2, 3 and 4 bottleneck layers per atrous convolution, respectively, perform better than their counterpart with just one layer. As initially expected, the additional bottleneck layers help the model achieve improved accuracy. This could be attributed to the additional transformations that the model can apply to translate the high-level features from the backbone into segmentation maps.

Counterintuitively, the best performing model is the CASPP-2 without KSAC. It is possible that the kernel-sharing method does not work well with an increased number of layers. With two bottleneck blocks per atrous convolution and three distinct atrous

Architecture					
OS	KSAC	Decoder	IoU	Params	MAdds
16	✓	ASPP	74.38	30.32M	14.94G
16		ASPP	73.99	39.76M	14.94G
16	✓	CASPP-1	73.48	25.09M	8.72G
16		CASPP-1	73.21	25.39M	8.72G
16	✓	CASPP-2	73.42	25.43M	8.89G
16		CASPP-2	**74.25**	26.08M	8.89G
16	✓	CASPP-3	73.49	26.49M	9.4G
16		CASPP-3	73.48	28.07M	9.4G
16	✓	CASPP-4	73.79	26.59M	9.43G
16		CASPP-4	73.64	28.17M	9.43G
32	✓	ASPP	73.14	30.32M	9.29G
32		ASPP	72.89	39.76M	9.29G
32	✓	CASPP-1	71.75	25.09M	5.69G
32		CASPP-1	71.63	25.39M	5.69G
32	✓	CASPP-2	72.13	25.43M	5.74G
32		CASPP-2	**72.29**	26.08M	5.74G
32	✓	CASPP-3	72.16	26.49M	5.86G
32		CASPP-3	71.85	28.07M	5.86G
32	✓	CASPP-4	71.52	26.59M	5.87G
32		CASPP-4	71.57	28.17M	5.87G

Table 5.4: The results of the DeepLabV3+ models with the ResNet50 backbone and variable ASPP modules. Decoder names explained in Section 5.3.1.

rates in the CASPP, the number of shared kernels can be burdensome to the prediction ability. However, with the results from Section 5.1 showing that KSAC does indeed yield improvement, the optimal selection of the bottleneck layers with kernel-sharing in the CASPP structure could alleviate this problem and, therefore, should be studied further. Presumably, sharing only the first layer can be the best of both worlds: more reliable and resilient shared kernel, as in the original paper, and the distinct transformation applied to the output of each dilation rate.

Ultimately, adding more than two layers seems to decrease the performance slightly while unnecessarily increasing the number of parameters and MAdds. This decrease can be attributed to the disproportion in the Compressed ASPP module - only the atrous convolution blocks have their depth increased. In the meantime, the image level features extracted from the backbone by the encoder in DeepLabV3+ (Figure 4.1) have not been extended, creating a disparity between the two signals. This uneven combination could potentially prevent the model from extracting the final output features in a balanced manner.

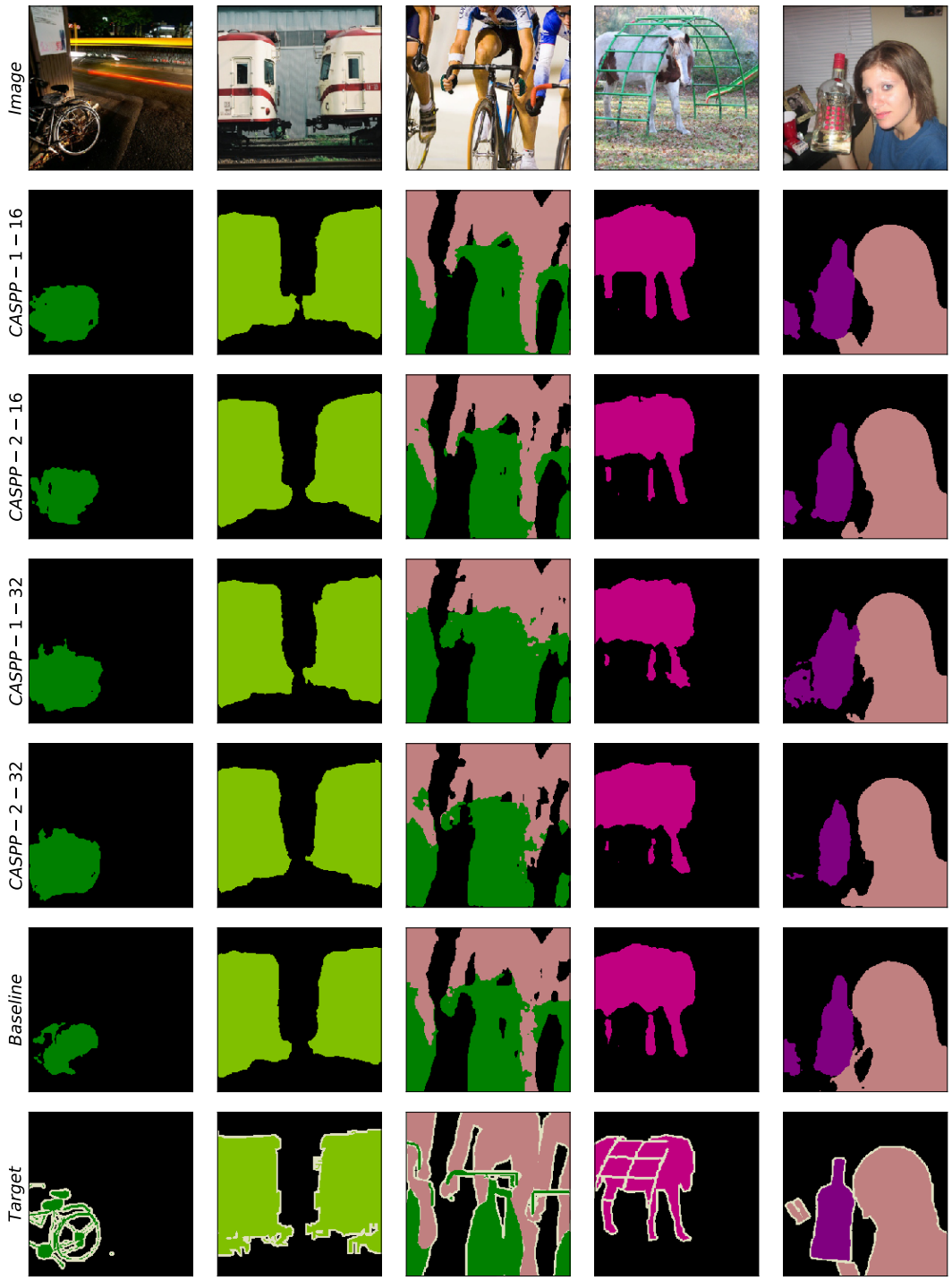

Figure 5.3: *Example predictions of the DeepLabV3+ model with the ResNet50 backbone. All models use non-shared kernels. Rows 2-5 show predictions for CASPP-1 and CASPP2 with an output stride 16 an 32. Row 6 shows the baseline from Section 5.1.*

Chapter 6

Conclusion

This project's objective was to determine whether there are some intrinsic differences in compression approaches of the semantic segmentation networks compared to image classification. The majority of the literature on efficient neural network designs for Computer Vision focuses primarily on image classification, assuming that the outcomes will be directly transferable to other tasks since extraction of the high-level features is a common task for most applications. This transferability can be true for some semantic segmentation since most models use a slightly modified version of an image classification model as a feature extractor.

Conversely, semantic segmentation has its particular, domain-specific problems like having the same input and output resolution. These problems encourage original approaches to classifier modules like ASPP that produce the output from the abstract, high-level features.

Several teacher models, based on DeepLabV3+, have been compared, focusing on the accuracy and the compression ability when modifying the network's classifier module to support Kernel-Sharing Atrous Convolution, depthwise separable convolution and bottleneck block.

The best model has been distilled to a large number of smaller, more efficient models using Knowledge Distillation or Attention Transfer to verify the applicability of distillation in semantic segmentation. For AT, three different approaches have been studied, focusing on the layers that the Attention Transfer should be applied to.

As predicted, Knowledge Distillation with the soft targets performs considerably worse than any other model. Attention Transfer, if executed on correct layers, can bring slight improvement to the models. Specific settings of AT for segmentation should be studied further to verify its usefulness. However, the amount of preparation work required to train the teacher models and then train the students afterwards does not work in favour of the distillation. In general, current distillation approaches, designed for image classification tasks, do not transfer well to pixel-wise classification domains.

Lastly, having noticed the reassuring compression when changing the standard convolution blocks to "Cheaper Convolution", a Compressed ASPP variant of the ASPP

module has been investigated. The addition of bottleneck layers to the convolutions with atrous rates improved accuracy while the model has remained more efficient than the naive compression approach. Additionally, to our best knowledge, efficient layers in the ASPP have not been studied before and shed new light on the atrous convolution and the incorporation of the extracted features with the object contours.

These discoveries expose us to the next steps that should be undertaken when working on semantic segmentation compression. Some of them are outlined in the next section.

6.1 Future work

The compressed ASPP module shows that it is possible to achieve similar accuracy with a more efficient design. A simple addition of the efficient bottleneck layers can have a significant impact on performance. However, only the addition of bottleneck layers to the atrous convolution layers has been studied. Therefore, a more comprehensive set of experiments should be performed with the Compressed ASPP to verify the module's optimal structure in the future.

Precisely, the addition of bottleneck layers to the low-level features should be studied, as well as the overall structure of the encoder. Residual connections could play a significant role in the better propagation of the signal in the deeper layers. Possibly, inverted residual connections, implemented by MobileNetV2, could offer a further boost. Moreover, since a predefined number of input and output channels has been used for the bottlenecks, the additional layers' specific sizes should be verified.

Kernel-Sharing Atrous Convolution achieves the promised benefits with standard convolution blocks, but its performance diminishes with the change of ASPP architecture. Possibly, a further study should be performed on the specific configuration of the kernel-sharing, especially that there has not been any follow-up work on the original paper.

Lastly, the recommended Attention Transfer settings by [Crowley et al., 2019a] yield only a negligible improvement in mIoU but confirm that distillation is possible for semantic segmentation. Therefore, other, more sophisticated attention mechanisms could be verified along with an ablation study on the parameter values.

6.2 Plan for the next year

Since this is the first part of a two-year-long project, next year's focus will be on compression methods for neural networks. Optimisation of the existing architectures requires a good comprehension of the authors' design decisions. Therefore, the focus will shift to the compression of more recent architectures that have improved upon the DeepLabV3+.

The overall topic of compression in Computer Vision is broad. The disproportionate number of techniques focuses on Convolutional Neural Networks. Although, with the recent rise of popularity of the large Transformer models in CV, there has also been

even more demand in making them more approachable. These sequence-to-sequence approaches displace the encoder-decoder structure, invalidating a large portion of the literature. Therefore, there is a large void that is only starting to be filled with ideas.

Consequently, next year's project will focus on finding new, attention-based equivalents of "cheap convolutions" in the Transformer-based network.

Bibliography

[Arnab et al., 2018] Arnab, A., Zheng, S., Jayasumana, S., Romera-Paredes, B., Larsson, M., Kirillov, A., Savchynskyy, B., Rother, C., Kahl, F., and Torr, P. (2018). Conditional random fields meet deep neural networks for semantic segmentation: Combining probabilistic graphical models with deep learning for structured prediction.

[Bai, 2019] Bai, K. (2019). A comprehensive introduction to different types of convolutions in deep learning.

[Brown et al., 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698.

[Chen et al., 2017] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs.

[Chen et al., 2018] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation.

[Chen et al., 2020] Chen, X., Wang, Y., Zhang, Y., Du, P., Xu, C., and Xu, C. (2020). Multi-task pruning for semantic segmentation networks.

[Cordts et al., 2016] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding.

[Crowley et al., 2019a] Crowley, E. J., Gray, G., and Storkey, A. (2019a). Moonshine: Distilling with cheap convolutions.

[Crowley et al., 2019b] Crowley, E. J., Turner, J., Storkey, A., and O'Boyle, M. (2019b). A closer look at structured pruning for neural network compression.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

[Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale.

[Everingham et al., 2015] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136.

[Fang, 2019] Fang, G. (2019). Deeplabv3plus-pytorch. `https://github.com/VainF/DeepLabV3Plus-Pytorch`.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Han et al., 2016] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. (2016). Eie: Efficient inference engine on compressed deep neural network.

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks.

[Hariharan et al., 2011] Hariharan, B., Arbeláez, P., Bourdev, L., Maji, S., and Malik, J. (2011). Semantic contours from inverse detectors. In *2011 International Conference on Computer Vision*.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151.

[He et al., 2018] He, K., Girshick, R., and Dollár, P. (2018). Rethinking imagenet pre-training.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[Heffels and Vanschoren, 2020] Heffels, M. R. and Vanschoren, J. (2020). Aerial imagery pixel-level segmentation.

[Hernandez and Brown, 2020] Hernandez, D. and Brown, T. B. (2020). Measuring the algorithmic efficiency of neural networks.

[Hinton et al., 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network.

[Huang et al., 2019] Huang, Y., Wang, Q., Jia, W., and He, X. (2019). See more than once – kernel-sharing atrous convolution for semantic segmentation.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[Kim et al., 2020] Kim, K., Ji, B., Yoon, D., and Hwang, S. (2020). Self-knowledge distillation: A simple way for better generalization.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA. Curran Associates Inc.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, page 91–110.

[Luo et al., 2017] Luo, J.-H., Wu, J., and Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines.

[Ren et al., 2016] Ren, S., He, K., Girshick, R., and Sun, J. (2016). Faster r-cnn: Towards real-time object detection with region proposal networks.

[Sandler et al., 2019] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2019). Mobilenetv2: Inverted residuals and linear bottlenecks.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.

[Sovrasov, 2020] Sovrasov, V. (2020). Flops counter for convolutional networks in pytorch framework. `https://github.com/sovrasov/flops-counter.pytorch`.

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions.

[Tan and Le, 2020] Tan, M. and Le, Q. V. (2020). Efficientnet: Rethinking model scaling for convolutional neural networks.

[Wild, 2018] Wild, M. (2018). Turning up the heat: The mechanics of model distillation. `https://towardsdatascience.com/turning-up-the-heat-the-mechanics-of-model-distillation-25ca337b5c7c`.

[Wu and Lee, 2018] Wu, Y. and Lee, T. (2018). Reducing model complexity for dnn based large-scale audio classification.

[Wu et al., 2019] Wu, Y., Liu, L., Bae, J., Chow, K.-H., Iyengar, A., Pu, C., Wei, W., Yu, L., and Zhang, Q. (2019). Demystifying learning rate policies for high accuracy training of deep neural networks.

[Zagoruyko and Komodakis, 2017] Zagoruyko, S. and Komodakis, N. (2017). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer.

[Zhang et al., 2019] Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., and Ma, K. (2019). Be your own teacher: Improve the performance of convolutional neural networks via self distillation.

Appendix A

Additional experiments results

Architecture					Metrics		
OS	KSAC	ST	DS	BTN	IoU	Params	MAdds
16	✓	✓			69.17	3.75M	4.29G
16		✓			69.34	5.23M	4.29G
16	✓		✓		67.73	2.48M	1.25G
16			✓		68.12	2.65M	1.25G
16	✓			✓	67.64	2.39M	1.03G
16				✓	68.15	2.46M	1.03G
32	✓	✓			68.00	3.75M	3.50G
32		✓			68.20	5.23M	3.50G
32	✓		✓		66.38	2.48M	0.83G
32			✓		66.44	2.65M	0.83G
32	✓			✓	66.31	2.39M	0.64G
32				✓	66.41	2.46M	0.64G

Table A.1: Results of the DeepLabV3+ models with the MobileNetV2 backbone. Column names explained in Section 5.1.1.

Architecture					
OS	KSAC	Encoder	IoU	Params	MAdds
16	✓	ASPP	69.17	3.75M	4.29G
16		ASPP	69.34	5.23M	4.29G
16	✓	CASPP-1	67.64	2.39M	1.03G
16		CASPP-1	68.15	2.46M	1.03G
16	✓	CASPP-2	68.41	2.61M	1.12G
16		CASPP-2	**68.75**	2.83M	1.12G
16	✓	CASPP-3	68.73	3.45M	1.46G
16		CASPP-3	68.63	4.15M	1.46G
16	✓	CASPP-4	67.65	3.55M	1.49G
16		CASPP-4	68.09	4.25M	1.49G
32	✓	ASPP	68.00	3.75M	3.50G
32		ASPP	68.20	5.23M	3.50G
32	✓	CASPP-1	66.31	2.39M	0.64G
32		CASPP-1	66.41	2.46M	0.64G
32	✓	CASPP-2	67.39	2.61M	0.67G
32		CASPP-2	**67.68**	2.83M	0.67G
32	✓	CASPP-3	67.59	3.45M	0.75G
32		CASPP-3	67.65	4.15M	0.75G
32	✓	CASPP-4	66.10	3.55M	0.76G
32		CASPP-4	66.06	4.25M	0.76G

Table A.2: The results of the DeepLabV3+ models with the MobileNetV2 backbone and variable ASPP modules. Decoder names explained in Section 5.3.1.

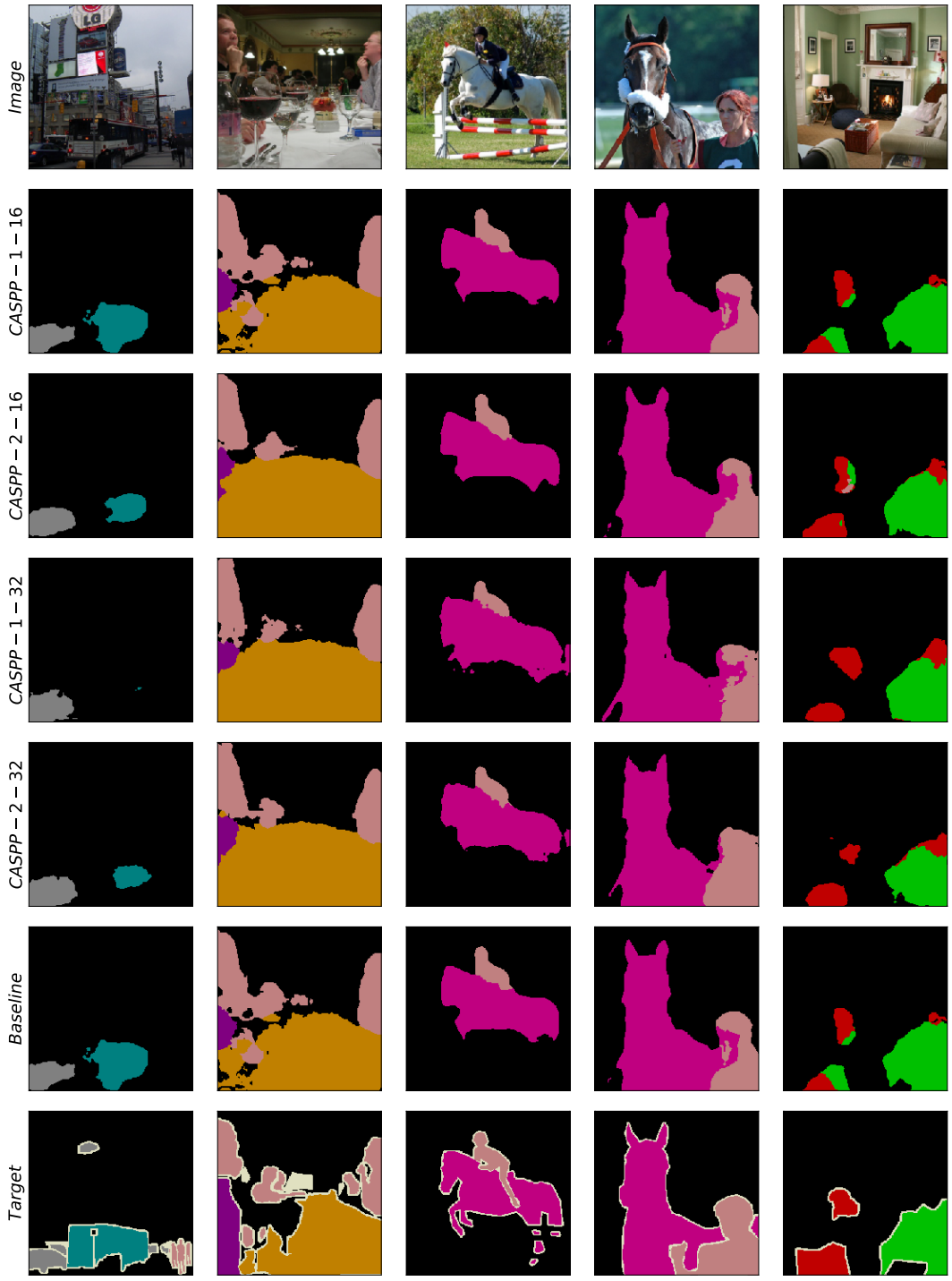

Figure A.1: *Additional predictions of the DeepLabV3+ model with the ResNet50 backbone. All models use non-shared kernels and. Rows 2-5 show predictions for CASPP-1 and CASPP2 with an output stride 16 an 32. Row 6 shows the baseline from Section 5.1.*