UNIVERSIDADE BEIRA INTERIOR

A Study on Efficient Semantic Segmentation

Luis Cavaca Pereira

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

(2º ciclo de estudos)

Orientador: Prof. Doutor Luís Filipe Barbosa de Almeida Alexandre

Covilhã, junho de 2022

Acknowledgements

I would like to thank: Prof Luís Alexandre for the infinite patience and support, my parents for the immense and continuous help and support, Mariana for always pushing and motivating me, and the extended list of friends and colleagues for all the time spent together during this academic process.

Resumo

O processo de segmentação semântica envolve uma enorme quantidade de recursos. Por consequência, este tipo de modelos são dificilmente ou, na grande parte dos casos, impossíveis de exportar para dispositivos eletrônicos de baixa capacidade computacional. Pequenos dispositivos, sendo alguns deles *robots*, não têm as capacidades computacionais necessárias para tornar o processo de inferência viável. Estes pequenos *robots* não têm muitas vezes memória RAM suficiente, ou noutros casos, bateria grande o suficiente para inferir de forma contínua durante curtos intervalos de tempo. Um outro aspecto consiste na impossibilidade de treinar os modelos nos próprios dispositivos, o que faz com que a sua aplicação seja ela mesma pouco prática. Por outro lado, as novas gerações de redes neuronais têm vindo a aumentar a escala dos recursos necessários, o que por um lado afasta ainda mais a possibilidade de usar estes pequenos dispositivos para tarefas de segmentação semântica. Com este problema em mente, o projecto foca-se em explorar métodos que tornem possível o uso deste modelos *on the edge*. Com este objectivo em mente, planeia-se explorar arquitecturas e camada convolucionais que fazem uso dos recursos de forma mais eficiente, métodos alternativos de segmentação e mecanismos de representação dos pesos para formatos mais leves.

Palavras-chave

Segmentação semântica, dispositivos de baixa capacidade computacional, complexidade de redes, optimização

Resumo alargado

Segmentação semântica pode ser vista como uma extensão da classificação de imagem. Em vez de atribuir uma única classe a uma dada imagem, este processo atribui uma classe a todos os pixels dessa mesma imagem. Segmentação semântica concede informação referente à classe dos vários itens, assim como a sua localização. Segmentação de objectos é um outro método semelhante. No entanto, o método estudado não distingue as várias instâncias para uma dada classe. Este método encontra várias utilidades práticas no dia-a-dia, como por exemplo, condução autónoma, auxílio na detecção de doenças em imagens médicas e reconhecimento de escrita humana. Embora existam vários cenários onde é possível aplicar segmentação semântica, este método é altamente dispendioso. Este custo encontra-se na quantidade de energia necessária, na necessidade de materiais altamente dispendiosos, como placas gráficas, na quantidade de memória necessária de memória ou até mesmo no tempo necessário para o treino dos modelos. O principal objectivo deste projecto consiste em explorar métodos alternativos que permitam a criação de redes mais leves, mas que consigam atingir níveis de precisão semelhantes. Atualmente, existem vários métodos para baixar a complexidade computacional. Como por exemplo, redes que usam vários caminhos paralelos para refinar diferentes tipos de informação ou que fazem uso de redes mais leves para a extração das características. Mecanismos semelhantes podem ser encontrados ao nível das camadas. Neste caso, computações que seriam feitas numa só etapa são divididas em várias. Quantização consiste em mudar o formato de representação dos pesos. Por exemplo, representar pesos no formato FP32 requer quatro vezes mais espaço que representar os mesmos em INT8. Dentro da quantização é possível encontrar abordagens diferentes, porém duas categorias podem ser descritas: pré-treino e pós-treino. As duas abordagens retornam modelos com necessidade energéticas iguais, no entanto quantização pré-treino leva a uma menor redução na precisão da rede. A poda de pesos e filtros permite a eliminação de componentes cujo impacto na rede é ínfimo. O primeiro, necessita de *frameworks* com optimizações no cálculo de matrizes esparsas. Já o segundo, pode ser aplicado em qualquer plataforma. A destilação, embora não reduza diretamente as redes, pode ser usada em paralelo com outros métodos para auxiliar a rede a recuperar os pontos de precisão perdidos. O foco do trabalho consistiu em testar os métodos referidos e analisar o respectivo impacto em redes neuronais de baixo consumo. Através da quantização foi possível concluir que métodos que usam o erro de quantização durante o treino conseguem obter níveis de precisão mais altos. É possível obter ganhos diretos com a poda de filtros, e usando mecanismo de destilação consegue-se guiar o treino de pequenos módulos dentro das redes ou as redes como uma só. Finalmente, com a troca de camadas é possível verificar poupanças diretas na memória RAM usada e no tempo de treino necessário para o treino da rede.

Abstract

Semantic segmentation extends classical image classification by attributing one class for each pixel in a given image. This approach requires a significant amount of resources to be performed. The majority of time, low-power resource devices are unable to deliver predictions on this task, because of its computational requirements. Some small robots lack inference speed, enough memory to inference a single instance at time or, even, battery life to delivery continuous predictions. Another aspect, is the incapability of training models on the edge, which can be a major limitation on the practicality of the solution. As if current networks were not big enough for this type of devices, novel architectures tend to be even more complex, which can be seen as a continuous divergence on the possibility of running this kind of models on low-power devices. With this in mind, the project has the goal of exploring efficient solutions to deploy segmentation models in the edge. To do so, the project aims at exploring efficient architectures and light convolutional layers, alternative segmentation methods and alternative methods of weight representation. In the end, by performing benchmarks on efficient networks with quantization, filter pruning along distillation and layer replacement, it is shown that these methods can be used to save computational resources, but to do so, they sacrifice precision points.

Keywords

Semantic segmentation, low-power devices, deep neural network complexity, optimization

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document describes the work carried out within the scope of the master's thesis. In this chapter, the problem is introduced in section 1.1, the goals are listed in section 1.2, the contributions and document's structure are detailed in sections 1.3 and 1.4, respectively.

1.1 Problem Statement

Semantic segmentation extends classical classification by giving a prediction for each pixel instead of predicting the class of the most relevant object in a image. This method allows the machine to get information on the class of the object and its location. Semantic segmentation follows a similar path as object segmentation, but both differ how they treat objects of the same class. The former, does not take it into account and gives each instance the same pixel value. The latter, tries to predict each instance and distribute pixel values to each instance.

Semantic segmentation is used in a wide array of scenarios, such as autonomous driving, day-to-day scene understanding, finding diseases in medical diagnostic and handwriting recognition.

Current object segmentation solutions are highly demanding in terms of resources. These resources can be in the form of energy consumption, expensive high-end hardware, such as graphic cards, memory or even in time needed for training a network. The main goal of the project focus on studying some possible approaches to reduce the usage of the described resources, while maintaining good results.

With this goal in mind, we are going to explore four mechanisms to deploy efficient models, which are the usage of low complexity deep neural networks, layer replacement, reduced weight representation and knowledge transfer between a baseline model and a similar version, but with modifications that make it more resource friendly.

Finally, the goal is to shed some light on which methods is better and present some guideline for a good usage.

1.2 Objectives

This project has four mains goals:

- Explore deep neural networks designed to use a low degree of computational resources;
- Study how layer replacement affects the speed and precision of a model;

- Analyse the possibility of transferring knowledge between two networks with a different level of resource usage;
- See the consequences of using a more compact weight representation and how it affects the precision of the model;

1.3 Contributions

The following list contains the contributions of this work:

1. A report with the current state-of-the-art on optimizing deep neural networks was written;
2. By using an efficient model to be the target to PyTorch's framework for Quantization, it was possible to conclude that smaller networks require an even more delicate approach to reduce the weight representation. Typically, these networks require methods that embody the quantization error inside the training phase. Since the redundancy of these models is low, they are fragile and tend to loose their accuracy with method that quantizes the model and only performs a single calibration phase;
3. By applying the crafted experimental setup for knowledge distillation, it was possible to conclude that filter reduction has a direct improvement on inference and training speed. Also, methods that perform distillation in a gradual way can deliver better results than others that perform the transfer in a single phase. This gain in accuracy requires a higher time in training phase, which can be considered as a negative side effect. Even though filter pruning and weight pruning can have similar fundamentals, the first method results in direct savings in required space and improves the time needed to train the model and perform inference. The second method, did not show in our experiments direct improvements in any of these parameters;
4. Finally, by applying depth-wise separable convolutions, it was possible to reduce the training time require for BiSeNet V2 [YGW+20]. Even though the network converged faster, there was no significant gain in inference time. On the other side, the required space for saving the network and its parameter count was reduced.

1.4 Dissertation Outline

This thesis' report is composed by the following chapters:

- **Introduction:** this chapter focus on introducing the problem that the thesis tries to solve, enumerate the thesis' goals and outline the work performed;
- **Related Work:** a review of a collection of researched papers that have core concepts or possible solutions that meet the thesis' needs. The chapter is composed by an analysis of papers related to datasets, architecture, layers or other alternative methods for complexity reduction;

- **Experimental Approach:** the chapter list the various aspects related with the experimental setup, such as choice of model, dataset and methods to latter implement;
- **Preliminary Experiment:** description of the initial experiment. This experimental trial was performed to fine-tune some aspects on the setup. The chapter highlights the process and concludes with the obtained results;
- **Limited Precision:** this chapter contains a description of the use of limited precision representations and how to deploy it. An analysis is also made that points to advantages and disadvantages of its usage;
- **Distillation:** this chapter is constituted by an explanation of how a smaller network can learn from a larger and more precise network. This explanation is then followed by set of experiments and some conclusion that highlight the efficiency of the approach;
- **Lighter Layers:** the third and last experimental chapter. Similarly to the other two, this chapter consists on the methods description, followed by the experimental setup and results' discussion;
- **Conclusions and Future Work:** final chapter of the document. A brief outline describing the major points focused in the thesis and what it is possible to conclude from the work presented in the document.

Chapter 2

Related Work

Along this chapter, the core concepts related to semantic segmentation are explained. First, an overview on what is semantic segmentation and where can it be applied is made in section 2.1. The overview is followed by an enumeration of datasets, in section 2.2, and novel layers that are able to perform convolutions in a efficient way, in section 2.3. Neural architecture search is explained in section 2.4. Methods to reduce the model's complexity, transfer knowledge and better handle information from feature maps are explained in detail respectively in sections 2.5, 2.6 and 2.7. Then, architectures are addressed in section 2.8. Finally, some guidelines to better deploy and benchmark deep neural networks are listed in section 2.9.

2.1 Overview

Semantic segmentation, also known as dense predictions, tries to estimate the respective class for each pixel in a given image. Semantic segmentation and instance segmentation can be seen as a very similar approaches. The key difference between them focus on how both treat different instances of the same class. While the former does not take then into account, the latter tries to differentiate them. This computer vision task takes the job of answering questions such as "what kind of objects are represented in the image?" and "where is x object located?".
This technique can be employed in a wide range of areas, such as scene understanding, autonomous vehicles, biomedical image analysis and autonomous medical diagnostics.
Semantic segmentation predictions tends to have similar resolution as the input data. These prediction are commonly referred to as segmentation maps and are obtain via one-hot encoding from the final feature map. One-hot encoding uses the network's final soft predictions and infers, for each pixel, what is the channel with the highest value. Since each output channel matches a distinct class, this process allows the passage from soft predictions to the segmentation map.
The building of deep neural networks started with the realization that stacking layers and keeping the original resolution would deem no advantages to the precision levels and was too costly. Upon several trials, scientists found that low level concepts are encoded in early stages, while high level concepts are encoded in later stages. Also, the number of filters per layer should increase as the distance to the first layer of the network.
Since semantic segmentation networks must be able to deliver a precise answer to the location of the object, they started to use an encoder-decoder structure. The first part, learns how to discriminate features, while down-sampling the feature map. The second, takes the small resolution feature maps and projects it to a full scale segmentation map.

As a side note, classification networks have the advantage of ignoring spatial location, which leads to being able to freely down-sample with mechanism that damage the spatial information capacity of the network.

Encoder and decoders paths are complementary. The former, is mainly composed by down-sampling operations (eg: average and max pooling) to reduce image resolution. This operation is performed by summarizing the values from a region of pixels into a single value. The latter, is comprised by up-sampling operations (eg: nearest neighborhood, bed of nails and transposed convolutions) to distribute the value of a single pixel to a region of pixels. From all the up-sampling operations, transposed convolutions are the only that can take advantage of training. On the other side, they are less efficient and can lead to lower precision predictions from region overlaps (check-board artifacts).

Even though these kind of convolutions are considered as one of the best, networks that use many transposed convolutions tend to be slow on training and inference stages. Considering that the reduction ratio is extreme, fine-grained predictions are hard or near impossible to obtain. To solve this problem, skipping connections were introduced as a mechanism to refresh later stages of the network with information.

Skipping connections helps the reconstruction of object's shapes and boundaries. To correctly used them it is recommended to slowly up-sample the feature maps, and then merge them with hidden feature maps previously computed. U-Net architecture, was able to deliver state-of-the-art results, upon its creation, by merging information from the encoding path with the symmetric expanding path.

Dilated convolutions, also known as atrous convolutions, can deliver richer feature maps. These kind of convolutions use a hyper-parameter called dilation rate to rule the wideness of the field of vision. The higher the value, the higher the amount of used parameters. Networks that make use of atrous convolutions can be extremely precise, but as a downside, they tend to require a large computational resources.

As seen above, networks considerably increased in precision, but this came with the downside of requiring more resources. From using up-sampling mechanism agnostic to training to create up-sampling convolutions that retain information from the training. From standard convolutions with stride to perform down-sampling to (dilated) convolutions, which can use all image's information, to obtain richer feature maps with the correspondent lower resolution. Also, by introducing re-freshening mechanisms, which make use of previously computed feature maps to re-introduce information on the networks flow, and avoid utilizing insufficient data. An obviously pattern started to show, the higher the amount of resources used, the more precise the network became. On the other hand, are all the deployed resources needed? Or does a degree of redundancy exist on the network that can and should be eliminated?

A balance must now be taken in account, since an *ad infinitum* stack of layers is not effective and networks should and must be deployed in real world scenarios. To do so, a novel set of considerations started to appear. Training and inference time, amount of memory needed to train and perform inference, the possibility to export the network into small devices... As an additional remark, when referring to small devices maintaining the level

of memory usage under a given threshold allows the model to be kept on-chip, which later reflects as battery and needed time saved.
Several approaches were taken to optimize the network's footprint. Some tried to optimize the way operations were made - efficient layers [IMA+16], [SHZ+18b] and [MHR19]. Other used controllers to navigate a given search space and find optimal layer and network configuration - neural architecture search [TCP+19], [PGZ+18], [LL20] and [YHC+18]. Pruning, quantization and encoding can be used singularly or in a complementary way [LDS90] and [HMD16]. These methods reduce the network complexity by eliminating weights, reducing weight precision and mapping the values into a lower space representation, respectively. Another approach uses pairs of baseline-smaller version and use information from the larger network to train the smaller one into following its inference behaviors - distillation [HVD15], [RDG+17], [GSL+19], [PKLC19] and [PPA18]. Networks can be deemed as efficient in the sense of how they organize their operation flow and how they minimize their computational complexity. For example, BiSeNet's family [YWP+18] and [YGW+20], DFANet [LXFS19] and transformers.

2.2 Dataset

The main mission of Cityscapes [COR+16] is to offer a novel dataset for understanding outdoor street scenes. The data within the dataset was collected during a wide span of time covering spring, summer and fall. All the images are original from 50 European cities. The authors stated that the dataset is missing scenes in adverse weather conditions, since these scenes require specialized techniques for acquisition and should be grouped in case specific datasets.
Upon the time of collecting data, the sensors were installed behind the windshield. Captured images were taken within a high dynamic-range (HDR) with 16 bits linear color depth.
From the collected images, 5000 were manually selected for dense pixel-level annotations. This group of images focus on providing high diversity in the scene composition. In other words, the captured scenes are rich in foreground and background objects.
Aside from the 5000 images set, a bigger group of images was taken to form a 20000 set of images with coarse annotations.
Instances from the dataset also include vehicle odometry, taken from sensors inside the vehicle, outside temperature and GPS tracks.
The two groups of images have different levels of annotations. The smaller, 5000 images, have a higher degree of labelling quality, where no object boundary was marked more than once. The second group traded object boundaries' accuracy for speed, where pixels under a polygon must belong to the same class.
The authors defined 30 visual classes under eight categories. This separation can be seen on table 2.1.
Even tough the dataset is composed of 30 classes, the authors recommend a reduction to 19 classes when using it. In this collection, some classes have an insignificant degree of

Group	Classes
flat	road, sidewalk, parking and rail track
human	person and rider
vehicle	car, truck, bus, on rails, motorcycle, bicycle, caravan and trailer
construction	building, wall, fence, guard rail, bridge and tunnel
object	pole, pole group, traffic sign and traffic light
nature	vegetation and terrain
sky	sky
void	ground, dynamic and static

Table 2.1: Cityscapes class composition.

appearance. This class discrepancy can lead to a higher degree of noise in the training process, which can lead to a direct impact on the model's ability to perform inference. This class reduction is made by ignoring all the pixels on the image outside the 19 classes subset.

The authors state that coarse annotated images must be used only for training. On the other side, densely annotated images were split into three sets: training, validation and test. Data was not split randomly, but in a way that ensures a high degree of variability of different street scenes scenarios. To achieve this premise, a criteria that balances geographic location, population distribution, population size and time of the year was used. This results, in 2975 training images, 500 validation images and 1525 test images. The first two groups of the split have public annotations, while annotations for the third were retained for benchmarking purposes.

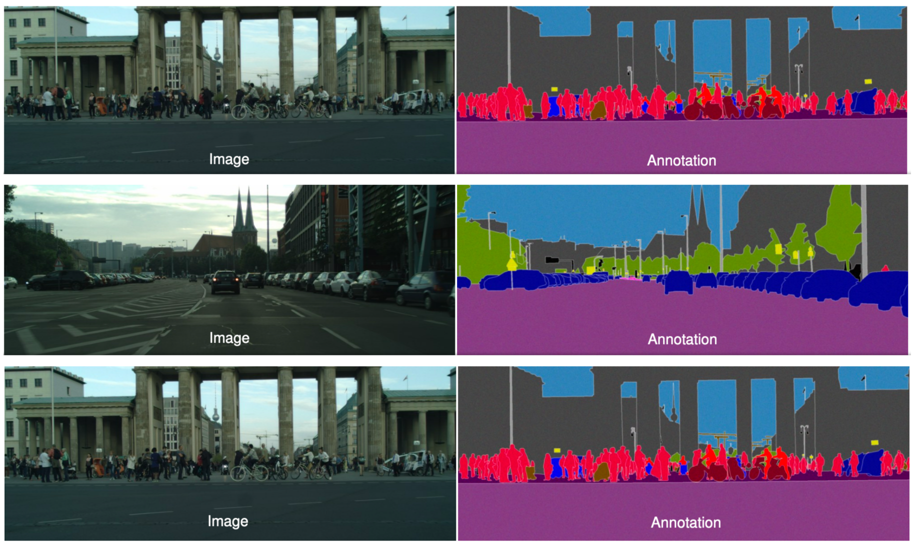

Figure 2.1: Three examples of cityscapes's raw images and their respective ground truth labels. Collage created by using pairs of image-annotation used on the dataset's paper [COR+16].

2.3 Efficient Layers

2.3.1 SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and < 0.5MB model size

The authors of the SqueezeNet paper [IMA+16] created a factorized mechanism for convolutional layers. This new mechanism was built on the following three premises:

- The size of the filters should be as small as possible. Therefore, replacing filter sizes of 3x3 by 1x1 is considered optimal, since they can lead to a saving of 9 times in terms of parameters number;
- The relationship between the size of the input channel and the number of parameters is directly proportional. So reducing the size of the first is extremely important to future savings;
- Down-sampling later in the network performs better than down-sampling earlier. This is based on the richness of the feature maps. Bigger feature maps contain more information. Applying convolutions on this kind of feature maps translate to better information refinement and more information inside the computational flow. If down-sampling is applied to earlier, information is lost and the quality of the information inside the architecture flow is lower.

The authors used the premises above to create a novel module, which is composed of two parts. The first part, performs a squeeze of the input channel, while the second part performs an expansion of the feature maps for feature extraction. The squeeze sub-module uses convolutions with smaller filter sizes - 1x1 - and strides bigger than one to reduce the dimensions of the hidden map. The expand sub-module extract features from the first part using a mixture of convolutions with filter sizes of 1x1 and 3x3. This novel module can be seen on figure 2.2.

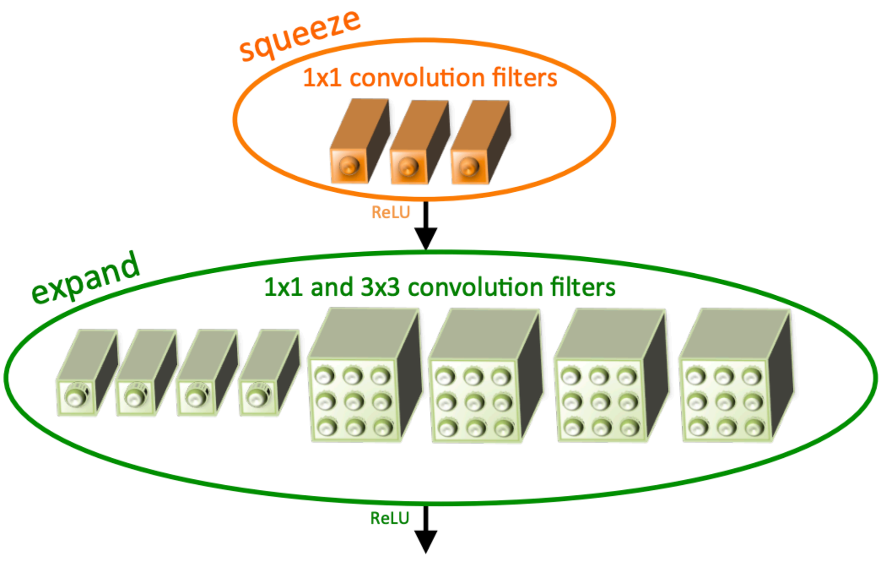

Figure 2.2: SqueezeNet's Fire Module composition. Image taken from [IMA+16]

One last remark on the expand sub-module is that balancing the ratio of filter sizes on the mixture can lead to optimal trade-offs between network precision and the number of parameters. In other words, it is possible to achieve precise networks, while keeping the size minimal.

2.3.2 MobileNetV2: Inverted Residuals and Linear Bottleneck

The authors of MobileNetV2 [SHZ$^+$18a] introduce the concepts of linear bottleneck and manifold of interest. The idea consists of the possibility to compress the information on a lower-dimensional block. This offers the possibility to save space, while maintaining the data properties.

A common approach to a residual block consists of applying element-wise addition between the input and separable convolutions followed by an expansion using point-wise convolutions.

From these points, the authors formulate a novel block, that takes an opposite approach. Instead of starting with a wide input, reduce, expand it and perform element-wise addition, we have a narrow input, expand it, narrow it followed by element-wise addition. This way it is possible to save important resources. This novel block can be seen in detail on figure 2.3.

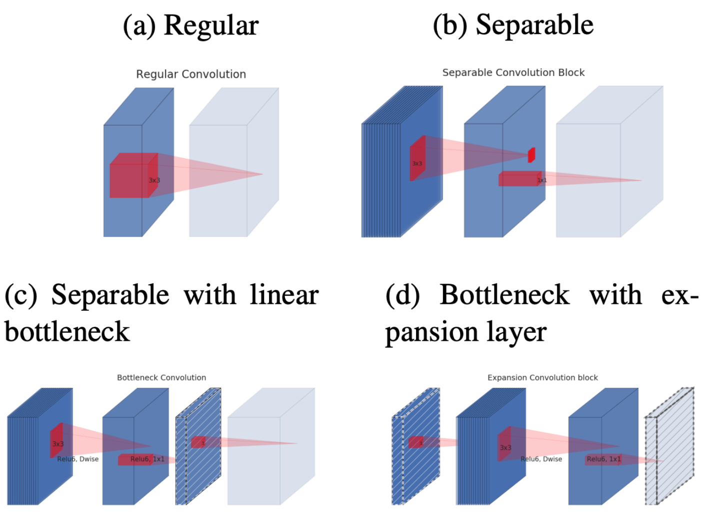
(a) Regular
(b) Separable
Regular Convolution
Separable Convolution Block
(c) Separable with linear bottleneck
(d) Bottleneck with expansion layer
Bottleneck Convolution
Expansion Convolution block
Relu6, Dwise
Relu6, 1x1

Figure 2.3: Evolution of convolutional block from regular to bottleneck with expansion layer. Image taken from [SHZ$^+$18a].

2.3.3 DiCENet: Dimension-Wise Convolutions for Efficient Networks

The paper [MHR19] starts by stating that standard convolutions encode both spatial and channel-wise informations simultaneously and separable convolutions, also knows as depth-wise separable convolutions, were introduced to isolate the encoding. The latter is a composition of a depth-wise convolution and a point-wise convolution, which respectively encode spatial and channel-wise information.

The main problem of separable convolutions consists on a heavy computational load on point-wise convolutions. To solve this problem, the authors formulated a novel unit called DiCE. This Unit merges as a factorization of standard convolutions and is composed of DimConv and DimFuse. The former applies a light-weight filtering to learn dimension-wise representations, while the latter combines the information. The architecture of the model can be seen in detail in figure 2.4.

DimConv is an extension of depth-wise convolutions, in the sense that it perform height, depth and width-wise convolutions. The output of the modules consists of a concatenation along the depth of the three convolutions.

The output from DimConv is then fed into DimFuse. This sub-module efficiently factorizes point-wise convolutions into two steps: local-fusion and global fusion. The first is obtained by performing group point-wise convolutions. Note that the idea of groups come from the concatenation along the way from the previous dimension-wise convolutions on DimConv. The result from local encoding is directed into two branches. The first, depth-wise convolution - encodes spatial representations. The second, squeezing channel-wise information via fully connected layers. Global information is then computed as an element-wise multiplication of these two branches.

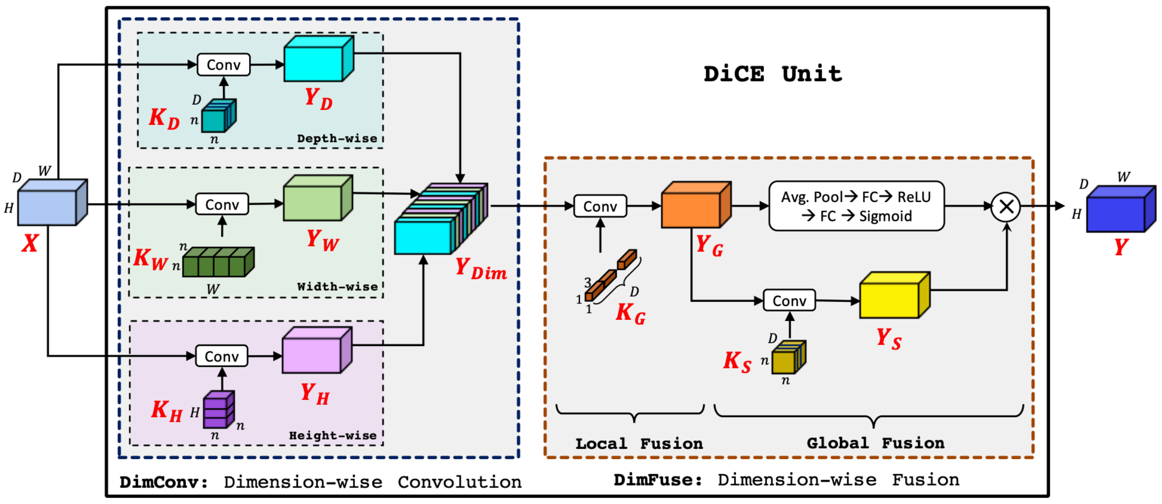

Figure 2.4: DiCE Unit's macroarchitecture. The image was taken from [MHR19].

2.4 NAS: Neural Architecture Search

2.4.1 MnasNet: Platform-Aware Neural Architecture Search for Mobile

The paper, MnasNet [TCP+19], consists on trying to find a Pareto optimal pair through reinforcement layer and a framework for automatically design of neural network architectures.

Initially the authors state that some resource metrics used in other papers are impractical, because they don't measure how well the model will perform under restricted resources platforms. Instead of FLOPS they used the latency of inference on a Google Pixel 1 phone.

The paper described a sample-eval-update type controller, which can be seen in figure 2.5. This controller first generates a neural network using a method called factorized hierarchical search. Then the model is trained, which leads to finding its accuracy. After training, the model is exported to a mobile phone and there the latency of inference is tested. After getting these two metrics, a reward function that balances accuracy and latency is calculated. Finally, depending on the value of the reward function, the parameters are updated.

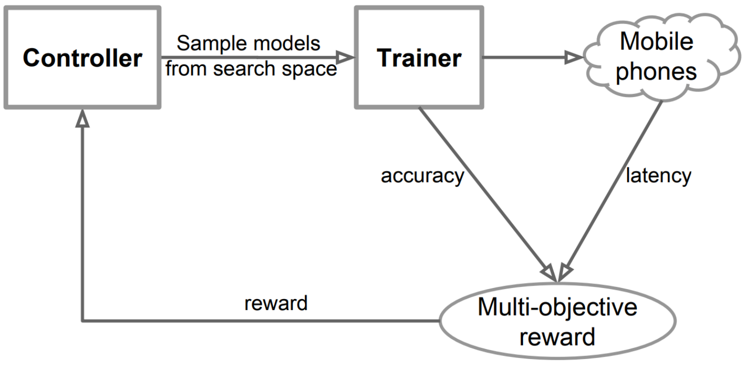

Figure 2.5: MnasNet's sample-eval-update mechanism. Image takan from [TCP+19].

A note on parameters and factorization. To generate neural networks the author implemented a model that generates blocks and then connects them into the networks. These blocks are made of a given number of the same layers. The morphology of these layers depends on the respective state parameter.
This implementation allows finding optimal trade-offs on accuracy-latency, while providing methods to generate models that take the maximum out of resources. For example, by stating that inference must be made under 100ms, it is possible to tweak the reward function and, consequently, search for the model with the best accuracy that perform under that constraint.

2.4.2 Efficient Neural Architecture Search via Parameter Sharing

As the degree of innovation in the fields of computer vision and neural networks grew up, the process of automatize the creation of neural networks became more important. This automation is followed by a heavier degree of parameter cost. This paper [PGZ+18] brings the idea of exploiting parameter sharing in order to reduce the memory footprint of the process.
The solution introduced in [PGZ+18] consists on a two-state trained framework that is able to automatically generate recurrent cells, convolutional cells or entire convolutional architectures.
The whole search space can be seen as an enormous acyclic graph, where nodes represent layers and the edges the flow of information. As mentioned earlier, the training of the framework is made in two smaller training phases. The first, consists on the training of the RNN controller, which is used to choose nodes from the search space graph. Second and lastly, the models previously generated by the controller are trained. The authors keep

the training of each component as an individual. While the controller is being trained, the generated models are set as static and vice versa. Also, the referred weight sharing comes from the state of the nodes in the graph. All the generated models uses the current setup from the search space graph. In other words, as the initial generated models converge, the weights from the correspondent layers are saved and used in the future generated models. This allows for a faster convergence on the future architectures.

In the end, the results obtained were competitive when compared with the original NAS, while achieving a 1000x reduction in time. As a final remark, in this method the generated models are only trained for a couple of epochs. While on [TCP+19] all the generated models are put on a complete training routine. This difference on the generated model's training leads to a higher gap in the necessity of needed resources, such as memory, time and used energy.

2.4.3 NeuralScale: Efficient Scaling of Neurons for Resource-Constrained Deep Neural Networks

Upon various iteration of neural networks crafting, scientist came to the conclusion that, as the network goes deeper, the number of filters needed is higher. These filters are required to capture embedded high level information in the feature maps and, also, to compensate the gradual reduction in the spatial dimensions.

The majority of designed networks blindly follows the previous statement. In other words, they blindly stack layers on top of layers to achieve a boost in accuracy. As a direct consequence, the level of redundancy increases.

To avoid having a higher degree of redundancy, a pruning mechanism can be used. Previously, pruning was used as a searching mechanism, instead of a one-step mechanism for obtaining a more efficient network. Merging this concept with fundamental mechanics from EfficientNet, a novel method for iterative pruning appears. As a side note, EfficientNet authors found that through the search for an optimal width, depth and resolution ratio the network can be scaled accordingly.

This new method, tries to scale the width of a CNN across several layers independently using global iterative pruning. The mechanism can be view as a two step pipeline that starts with parameters pruning - to reduce the levels of redundancy - followed by efficient filter scaling. This last part, tries to extend the number of filters in a given layer but maintaining a good ratio of size and redundancy. This pipeline can be seen in figure 2.6.

NeuralScale can be viewed as a NAS mechanism. It is a evolution from mechanisms that tried to scale the network in a agnostic way. While NeuralScales learns how to scale the deep neural network, previous works would apply all the scales in equal ways for every epoch.

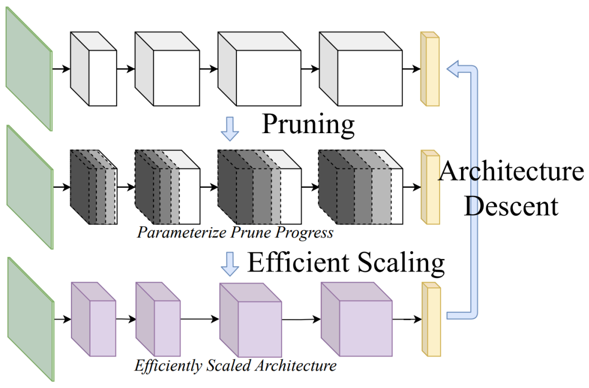

Figure 2.6: NeuralScale two step pipeline. Image taken from [LL20].

2.4.4 NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications

When bench-marking deep neural networks, there are several possible metrics that can be used. These metrics can be split into two main groups: direct and indirect. The first group is composed by real-world benchmarks such as energy consumption, inference time and latency. The latter has benchmarks achieved by formulas, such as mAdds. In this case, the number of additions and multiplications are taken into account to measure the complexity of the model.

Indirect methods can be used to search for more efficient networks, but as the authors state, their usage is not linked with direct gains. Since correlation between number of parameters and effective training and inference time isn't directly linear and it is even possible to have models with fewer mAdds that can be slower their counterpart.

The paper [YHC+18] tries to solve three fundamental problems.

First, building models for specific hardware is expensive. When crafting dedicated implementations for a given hardware, there is a need to know deeply how it behaves and considering the large amount of devices that already exists, it is infeasible to deploy such projects.

Second, heterogeneous hardware can behave differently and have a different set of optimized operations.

Third, pruning isn't optimized in several deep learning frameworks, which is a direct effect of the lack of optimization for sparse matrix operations. Therefore, it is more efficient to remove complete filters instead of a given set of weights. As a side note, when removing filters, a special care is needed, since their removal can lead to the inability of the network to learn.

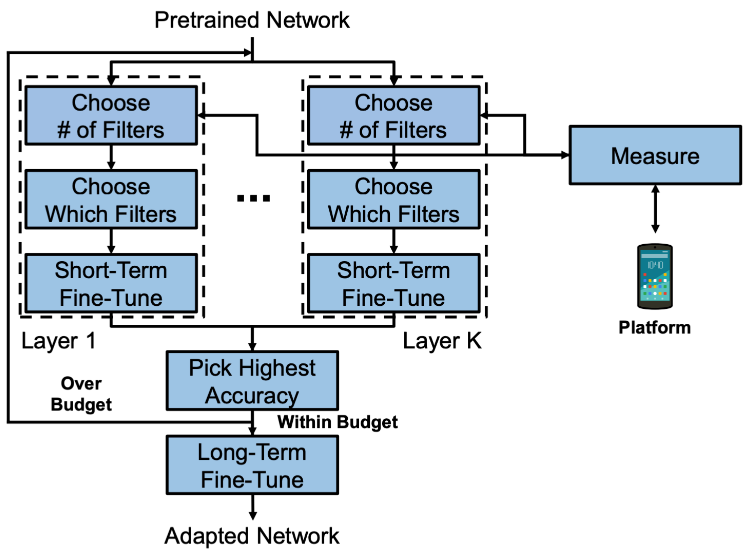

Figure 2.7: NetAdapt's algorithm flowchart. Image taken from [YHC+18].

The resultant framework, which can be seen in figure 2.7 takes a given model and adapts it to fit into a set of resources restrictions. Typically, the model is destined to run on mobile devices and uses indirect metrics on them to guide the optimization of the network. Even thought the framework was designed for mobile devices, it is agnostic and can be deployed into any kind of platform.
The NetAdapt algorithm follows a reduction schedule and runs interactively. By saving the best resultant model after each iteration, it is possible to achieve various degrees of reductions and different trade-offs pars. For each iteration, a given number k of filters is reduced and the model is fine-tuned. In the end, the more precise model that fits the constraints is chosen and a longer fine-tuning phase is performed. This final phase is needed so the resultant model is able to recovery any degree of precision lost in the optimization process.
Experiments on MobileNet v1 and v2 shows improvement on inference time while losing few on precision points.

2.5 Pruning, Quantization and Encoding

2.5.1 Optimal Brain Damage

The paper [LDS90] takes into account the minimum descriptive length, which states that smaller networks are better to represent information, since they learn how to truly generalized on the given data.
The authors state that metrics as magnitude are not optimal, since they don't incorporate the real impact of that weight in the network. So they propose a new approach. First, a network is trained, then the hessian matrix of the loss function with respect to the parameters is calculated. The third step consists of finding the value of saliency, a novel metric that balances the values on the matrix with the respective weights. Finally, the values

for saliency are sorted, a given number of the parameters with the smallest saliency are eliminated and the network is re-trained for calculating the final weights.

2.5.2 Deep Compression

Deep Compression [HMD16] is a compression pipeline composed of three steps, which are weight pruning, quantization and weight sharing and Huffman encoding.
The first step, weight pruning, is performed by training a given network with the standard procedure and then all the weights under a given threshold are eliminated. Finally, a second training procedure is performed to re-adjust the network.
The second step, quantization and weight sharing, consists on putting all the weights under K-means clustering. This method, groups all the weights by similarity under a given number of clusters. The id of the respective cluster is used to create a weight sharing table. Finally, a standard training procedure is done, to adjust the weights of the shared table.
The third step consists of using Huffman coding to compress the weights. This algorithm is optimal for the context because it allows a size reduction, since it encodes information in a variable-length form. In order other words, the most common words are encoded using a smaller number of bits.

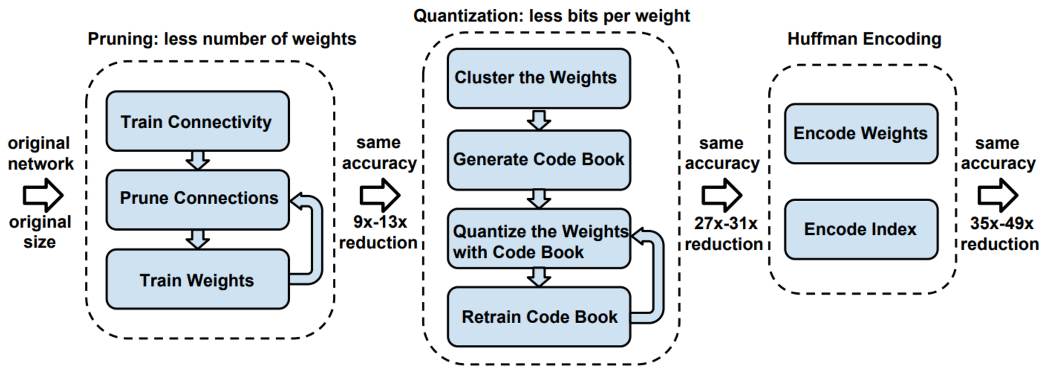

Figure 2.8: Deep Compression's three stage pipeline flowchart. Image taken from [HMD16].

This approach takes a state of the art network and applies this three step pipeline, which can be seen in figure 2.10 to achieve removal of redundancy and the possibility to represent the model on a smaller space.

2.6 Distillation

2.6.1 Distilling the Knowledge in a Neural Network

The work performed in [HVD15] was pioneer in the field of transfer learning. The core concept consists on using the predictions from one or more trained models to guide the learning process of the in-training model. Also, the method is agnostic to the architecture of the model and which task the model is pre-set to work on.
Initially, the authors talk about choosing a pair of models. One, the teacher, usually with higher complexity and already trained on the specific task. The second, the student, which

possesses a lower degree of complexity. Note that in both networks the task head is removed and both soft targets are used to guide the learning, where the student soft target is trained to match the professor's ones. A architecture's task head refers to its final part, where final computations are performed to deliver predictions. On semantic segmentation cases, the task head typically up-samples the hidden feature map to the desired dimensions and performs one-hot encoding.
When using only the professor predictions, the student is being trained to learn the classes distributions. Another way to use knowledge distillation involves the balancing of professor's soft targets and dataset's ground truth. By using this mechanism, the student is rewarded upon choosing the right class, instead of when mimicking the professor's class distribution.

2.6.2 Data Distillation: Towards Omni-Supervised Learning

The novelty from [RDG+17] comes from bringing distillation and its advantages from knowledge sharing between different models to the realm of data and datasets.
The proposed method uses a baseline model as a labeling mechanism. For a given dataset and given baseline, where the last is consider as robust and able to extract information efficiently, several transformation on the data are used. After the transformations, the baseline model is used to infer on all images of the dataset. This inference step can be seen as an auto-labelling mechanism on the dataset. When using more than one model for auto-labelling, a final step is needed, where all predictions are merged into one instance.

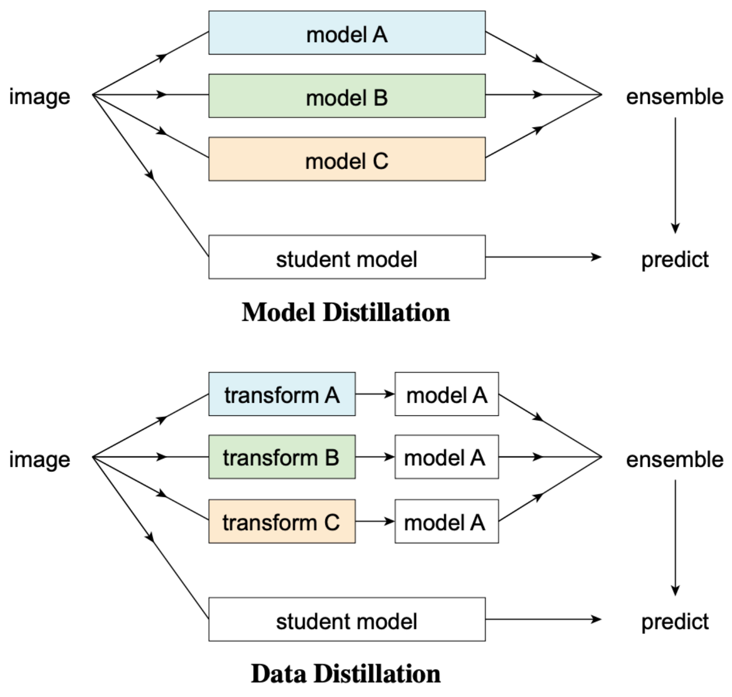

Figure 2.9: Model distillation *versus* data distillation. The former, is used to perform inference, while the latter is used as labelling and data augmentation mechanism. Image taken from [RDG+17].

A new dataset, originated from the mixing of original and auto-labelled images, is cre-

ated. Upon training models with this new dataset, the authors advise to keep a balanced ratio between original and auto-generated images. This healthy ratio gives room to take advantage from the mechanism, otherwise it would be useless.
Experiments using the methods, show that this new kind of data allows for more information and, as a consequence, model trained with this data-augmented method perform better.

2.6.3 An Embarrassingly Simple Approach for Knowledge Distillation

The paper [GSL+19] describes a factorized method for distillation. The method breaks a single model into several sequential parts, where the number of training stages is equivalent to the number of separated parts plus one.
When training the model for a given state, the authors consider the previous iteractions as trained and the future ones as non-existent. As an illustration, consider a model broken into three parts. In the second training phase, the first part as already completed the learning process and is only used as inference mechanism to feed information into the model's second part. The second component takes information from the first one and updates the weights along the training. The third part is discarded, since the information flow doesn't pass in the pipeline. This process can be seen in the figure 2.10.
The number of stages is directly proportional to the number of created break points plus one. This plus one stage is the same as a final fine-tune of the whole model.
The transfer of knowledge is guided through the hidden maps. Since the models learn from it, it is possible to add the possibility of heterogeneity between models. In other words, the student model can learn from professors, where the last ones can have a additional number of layers between the matching breakpoints.
As a side note, the task head of the model is treated as non-existent. This model's tail, is separately trained after the stage-by-stage distillation on the task specific dataset.

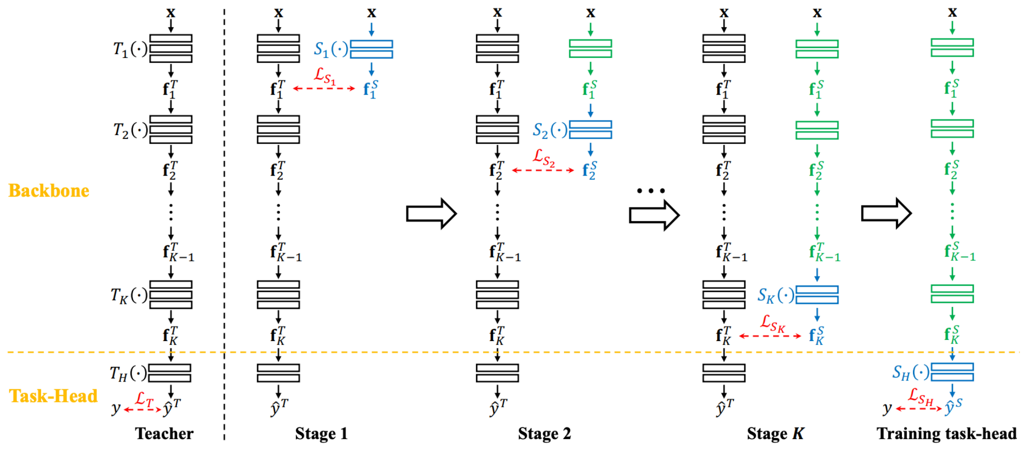

Figure 2.10: Stage-by-stage distillation method applied on a network broken into two stages. Image taken from [GSL+19].

The authors found that the optimal approach would be to create break-points on downsampling layers. It is possible to factorize the model into smaller parts, but this separations would not carry greater improvements. Also, when dealing with different shapes for the hidden maps, a single convolution can be used to match the student-professor hidden maps.

2.6.4 Relational Knowledge Distillation

The work on [PKLC19] starts with the premise that distillation focus only on singular predictions from individual instances, whereas it should take advantage on the relationship between different instances' predictions.
When distilling knowledge from the professor to the student, soft predictions from singular instances are used. They allow the student to mimic the behavior of the professor upon data with a close degree of similarity.
The core concept of relational knowledge is to avoid what was mentioned earlier and use a bigger flow of information. In order to do so, the authors used images from the same batch and infer how they correlate with each other. This process can be observed on the right side of the image 2.11. The degree of their relationship is calculated using a new set of losses, distance-wise and angle-wise loss. The first, checks how two images correlates with each other, while the second, follows a similar method, but for three images.

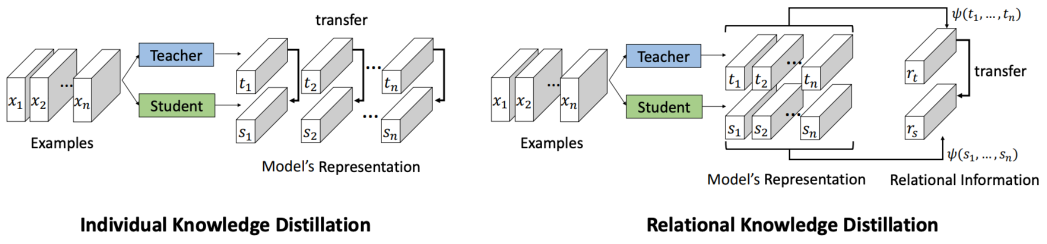

Figure 2.11: Distillation performed by use of a singular prediction *versus* distillation performed by usage of two or more predictions. Image taken from the [PKLC19].

One final remark is that distance-wise and angle-wise losses should be used along traditional losses, since they aren't strong enough for doing the job alone. This new set of losses must be leveraged using a balance factor, otherwise the training becomes unbalanced.

2.6.5 Model Compression via Distillation and Quantization

The paper [PPA18] has two main goals: prove that model can have a good level of accuracy even though it is restricted to integers and show that it is possible to distill knowledge from a full precision model to its distilled version.
Previous work has segregated compression and distillation. When compressing a network, a three-stage pipeline would be designed. These pipelines would begin with quantization, then weight sharing, and finally, would encode the weights using encoding algorithms (such as Huffman). Such kind of pipelines try to reduce the number of used weights and/ or its representations, which would lead to a loss in the ability to retaing knowledge of the network. As a rule of thumb, as the size of the representation of weights goes down, the higher the inability of the model to learn.

The paper [PPA18] introduces methods, quantized distillation and differentiable quantization, which were built on these two premises: full precision models perform better than their quantized versions and quantization is able to achieve a higher degree of compression. Both methods combine compression and distillation, where knowledge is shared between a full precision professor and a shallower compressed student. To enable this sharing, a loss functions combines soft targets between feature maps of the professor and the student. Also another loss that combines information from the student's predictions and ground truths is used. As a side note, the two methods differ in terms of the quantization mechanism. While the first is static, the second focus on non-uniform quantization, which is achievable using stochastic gradient descent. The method can be visualized in figure 2.12.

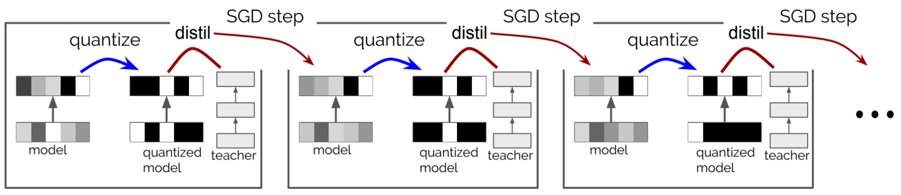

Figure 2.12: Model compression's pipeline behavior. Image taken from [PPA18].

By using this mechanism, the authors were able to find a 4-bit quantization version of ResNet18 with higher degree of precision than the correspondent full precision version. The authors state that they could go further by using Huffman coding in the final stage. Since this mechanism uses variable-weight encoding, it is possible to have more common words using less bits.

2.7 Attention

The paper [WJQ+17] implements the merge of two concepts, residual learning and attention mechanisms.

First, residual learning solves the problem of gradient explosion and gradient vanishing using residual blocks. Typically, these blocks consist of element-wise operations, such as addition or multiplication, via a shortcut connection and the output of a series of convolution blocks.

Second, attention mechanisms focus on giving a special consideration to key parts of the input.

When merging these concepts, we get a Residual Attention Block that is able to learn soft masks with the capability of removing noise or redundant parts of the data.

A key characteristic of the new module consist on the ability to scale the number of stacking components. Previous approaches to attention modules would reach a point where stacking would lead to precision degradation. This component has the ability to improve precision as the number of modules goes up.

As a final remark, the residual network architecture is made upon the stacking of these sub-modules. Figure 2.13 shows how the stack of modules gave birth to the residual attention network architecture. By using modules, the network breaths modularity in its core, which can be translated as new architectures, that vary in degree of complexity and can be crafted by stacking additional modules.

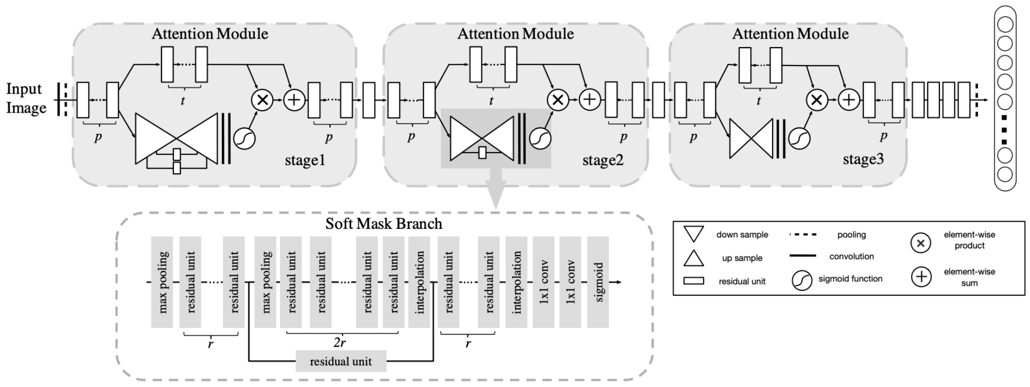

Figure 2.13: Residual Attention Network, designed for the task of image classification, macro-architecture. Figure taken from [WJQ+17].

2.8 Baseline Architectures

2.8.1 Deep Residual Learning for Image Recognition

The paper [HZRS15] introduces one of the most ground-breaking architecture in the field of computer vision and image classification. ResNet's architecture brought to life the concept of residual connections and even today it is used as baseline, for architectures benchmarking, and as a backbone for feature extraction.

Architectures prior to ResNet applied to much focus on stacking layer after layer to increase precision. When stacking too many layers, problems such as gradient explosion and gradient vanishing started to appear and made impossible the learning of the network.

Residual connections works as information re-fresheners. This connections irrigates later stages in the network with information and allows fusion of multi-stage information. Typically, they follow a path of information de-multiplexing, where one of the branches passes through some computations. After one or more stages, both paths are fused. This kind of connections can be seen in the figure 2.14.

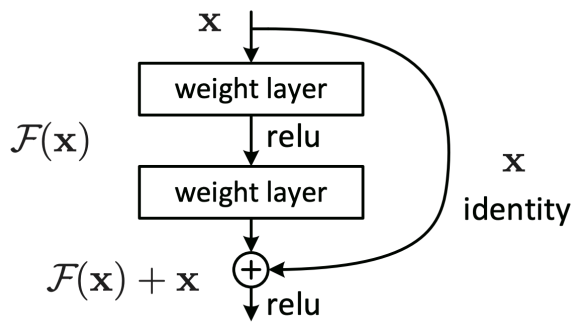
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.14: Fundamental theoretical concept for building residual blocks and skipping connections. Image taken from [HZRS15].

The concept of knowledge re-utilization became standard and lately several architectures took inspiration on it. These networks use residual connections, residual blocks or follow a de-multiplexing-multiplexing architecture.

2.8.2 BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation

The bilateral segmentation network [YWP+18] was born from the mixture of the five following concepts: spatial and context information, U-shape architecture, attention mechanism and real time segmentation. The first, spatial information consists of high level details relative to the position in the image. Where, on the other side, context information is in parallel with low level details and refers to details more difficult to see at naked high, such as variations in color. U-Shape architecture tries to solve the problem of losing spatial information. These architectures, first, halve the size of the feature maps to learn context information and at latter stages, they up-sample the feature maps to the original - or similar - resolution. This pipeline is prone to the degradation of information at the border, which can be recovered with the addition of skip-connections. The fourth concept, attention mechanism, is employed for learning the global context and to highlight more important features in the image. Finally, real time segmentation, translates to the speed constraint of the problem. The crafted network needs to be efficient and able to segment information fast and on the fly.

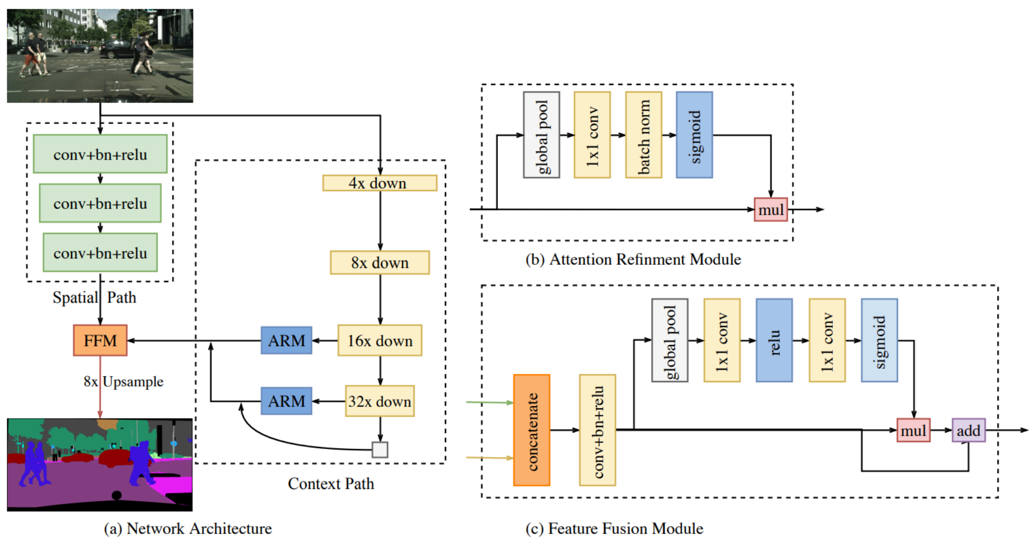

Figure 2.15: BiSeNetV1 and sub-modules' macro-architecture. Image taken from the original paper.

Bilateral Segmentation Network is composed by two information paths, one sub-module for refinement, and a final aggregation module that fuses the information from the two paths to deliver the final prediction. These two paths are called Spatial Path and Context Path, the sub-module is Attention Refinement Module (ARM) and, the last one, is Feature Fusion Module (FFM). The architecture can be seen in detail on figure 2.15.

- **Spatial Path:** Solves the problems introduced by the usage of previous methods (image resizing, other architectures,...), such as less available or degradation of information. This path is build with three convolutional layers with stride 2, batch normalization and ReLU. In the end, is possible to have feature maps with large spatial size and rich spatial information. By employing this method, the down-sample is fast and lightweight;
- **Context Path:** Inspired by methods such as pyramid pooling. This path tries to solve the problem of its ancestor, which is the high amount of used memory. This problem leads to low inference speed, or sometimes, even impossible to use the model on small mobile devices. The path was built using a light weight backbone model, Xception, with a tail of global average pooling. By using a light weight backbone it is possible to down-sample the map fast, while having a large receptive field that encodes high level semantic context information. Global average pooling as tail provides a receptive field with maximized global context information. As a final note, the resultant architecture can be seen as an incomplete U-Shape;
- **Attention Refinement Module:** This sub-module can be found inside the context path. It is used to refine features and capture global information by the usage of an Attention Vector. Also, by keeping the feature map always with the same size, its computational cost is almost negligible;
- **Feature Fusion Module:** This module serves as final computational unit before delivering the prediction. It fuses high level features from spatial path and low level

feature from context path. Previous iterations used sum of the feature maps. This method would lead to poor performance in terms of accuracy. To avoid this, the authors design the module to concatenate the outputs from the two paths, passed the new tensor under batch normalization (to scale the features) and, finally, compute a average max pooling, This final step allows selection and combinations of features.

The datasets used targeted urban understanding scene and the understanding of images composed by common day to day objects. For the first task, Cityscapes and CamVid datasets were used, while for the latter Coco-Stuff was deployed. As a side note, the authors used 19 classes for Cityscapes and only performed training with the sub-dataset composed by images with fine annotations. The other two datasets, CamVid and Coco-Stuff, were mainly used to perform benchmarks.
As a conclusion, the authors were able to craft a lightweight architecture that can perform better than DeepLab v2 in terms of accuracy and inference speed. By the parallel information processing the architecture tends to be faster. Also it is possible to segregate high level features - spatial path - from low level features - context path - and fuse them latter to deliver a more precise prediction.

2.8.3 BiSeNet v2: Bilateral Network with Guided Aggregation for Real-Time Semantic Segmentation

BiSeNet v2 [YGW+20] is a direct improvement of BiSeNet v1. BiSeNet architecture family segregates the computation of low and high level details into two separated paths. The output from these two paths is fused, using a layer or sub-module, to delivery the networks' prediction. In the second version, both paths were upgraded and the sub-module where the information is fused was re-designed. The network targets scene understanding, autonomous driving, human-machine interaction and video surveillance. Additionally, differently from the previous iteration, a booster trainning strategy was depployed to allow a faster convergence.
When creating BiSeNet v2, three main aspects were taken in attention. Generic semantic segmentation methods, real-time semantic segmentation methods and lightweight architectures.

- **Generic Semantic Segmentation Methods:** composed by architectures based on dillatation backbones (Deeplab v3, PSPNet,...) or encoder-decoder skeletons (RefineNet, LRR; GCN, HRNet,...). Typicaly, these backbones are highly accurate, but lack in inference speed;
- **Real-Time Semantic Segmentation Methods:** (SegNet, E-Net, ESPNet,...) can deliver predictions fast, but with the cost of a significant margin of error;
- **Lightweight Architectures:** (such as Xception, MobileNet, ShuffleNet, ...) use group or depth wise convolutions and separable convolutions. These networks have a reduced computational complexity, optimized memory access cost and can deliver results in real time.

The core of this novel architecture is found at three sub-modules, detail branch, semantic branch and aggregation layer. Detail branch treats spatial details, which encodes low level information, and is built using a shallow structure with small stride values. The authors avoided any kind of residual connections to keep module's complexity at the minimum. The second sub-module, semantic branch, treats high level semantics. In this path, a fast down-sampling mechanism is applied to promote the level of feature representation and enlarge the receptive fields efficiently. The final sub-module, Aggregation layer, fuses complementary information from the previous sub-modules. In this layer, a pre-fusion calculation is needed, since the data from different paths have different resolutions when entering this sub-module. The fusion of information is performed using bidirectional aggregation method, which can lead to higher levels of precision and inference speed when compared to standard fusion mechanism, such as summation or concatenation. Figure 2.16 illustrates how this components are structured.

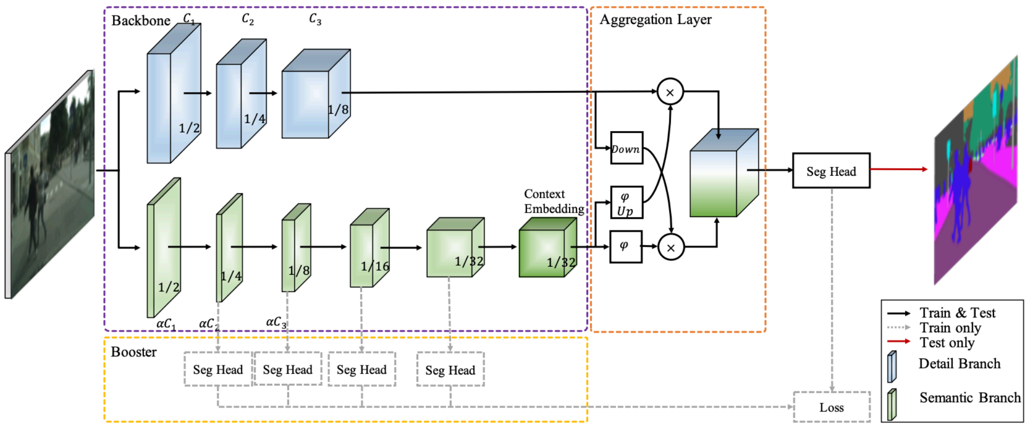

Figure 2.16: BiSeNetV2's macro-architecture and micro-architecture. Image taken from BiSeNet-V2 paper.

In the following list, it is possible to see some implementation details and sub-module compositions:

- **Detail Branch:** pipeline composed of three stages. Each stage consists on a convolution, with stride value of two, batch normalization and an activation functions. Second and third convolutions have the same number of filters and output size;
- **Semantic Branch:** Allies lightweight backbones (such as Xception [Cho16], MobileNet [HZC+17], [SHZ+18b] and [HSC+19] and ShuffleNet [ZZLS17]) with efficients blocks to enlarge the richness of the fields;
  - **Stem Block:** initial architecture's block. Performs down-sampling with two different methods in order to shrink the feature representation. The outputs from these two methods are concatenated to delivery the block's output;
  - **Context Embedding Block:** global average pooling and residual connections. Contextual information is merged efficiently in this block.
  - **Gather-and-Expansion Layer:** in this paper a lighter mechanism was used.

Instead of using separable convolutions with 5x5 kernel, the authors used two separable convolutions with 3x3 kernel. This block enriches the receptive field.

- **Bilateral Guided Aggregation:** performs the encoding of multi-scale information using element-wise summation and concatenation.

In the training phase, the rocket booster strategy was employed. This method enchances feature representation during the training, which latter translates in a faster training. The method employes auxiliary heads in the middle of the networks. When the train is completed, these heads are discarded.
In this second iteration of BiSeNet, the used datasets were the same as in the first one. The datasets were Cityscapes, Cambridge-driving Labeled Video Database, also known as CamVid, ann COCO-Stuff. In the first one, only 19 classes were used and on the second one, authors ignored any pixels that didn't belong to the desired classes.
The resultant architecture improved the two most important aspects of light models, accuracy and inference speed. This novel architecture improved BiSeNet family's architecture and the state of the art in lightweight semantic segmentation.

2.8.4 DFANet: Deep Feature Aggregation for Real-Time Semantic Segmentation

This paper [LXFS19] introduces a novel architecture that uses multi-scale feature propagation through sub-networks and sub-stages arranged in cascade. The network uses semantic information and structure detail from the re-use of backbone's high level extracted features. These features are combined at different stages so enhance feature representation inside the network.
The network formulation is based on 5 key points, which are:

- **Real-Time Segmentation:** perform semantic segmentation fast and precisely on demand. In this sub-task, networks such as SegNet [BKC15], ENet [PCKC16] and BiSeNet [YWP$^+$18] were commonly used;
- **Layer Factorization.** where a convolutional operation is divided into several parts to reduce the footprint. In this case, depth-wise separable convolution;
- **High level features.** Multi-scale representation is the most used mechanism. Two main architectures approaches, encoder-decoder and pyramid pooling. PSPNet [ZSQ$^+$16] and DeepLab [CZP$^+$18];
- **Context Encoding** via channel-wise attention (such as in SE-Net [HSS17]);
- **Feature Aggregation** by sub-modules that perform middle computations to extract multi-scale features between the encoder and decoder, as in RefineNet [LMSR16].

DFANet possesses three core parts, which are the lightweight backbones, the sub-networks aggregation modules and sub-stages aggregation modules.

The authors selected the Xception network to be DFANet's backbone. They replaced conventional convolutions by depth-wise convolutions and appended a fully connected attention module in the end. This new tail allows for a maximum receptive field.
Sub-networks' aggregation modules connect different backbones through stages. They up-sample the high level feature maps derived from the last backbone and feed them to the next one. Sub-networks aggregation modules refine prediction results from a coarse to a fine level. This type of structure is commonly named stack of encoder-decoder "hourglass".
The final key points is the sub-state aggregation module. Inside the module, features from different stages are combined to deliver feature maps with good receptive fields and high dimension structure details. Inside the stage pipeline, feature maps encode low level and spatial information. Data refinement is obtained through layer concatenation. By using data from different stages, the information is richer and possess different context properties.

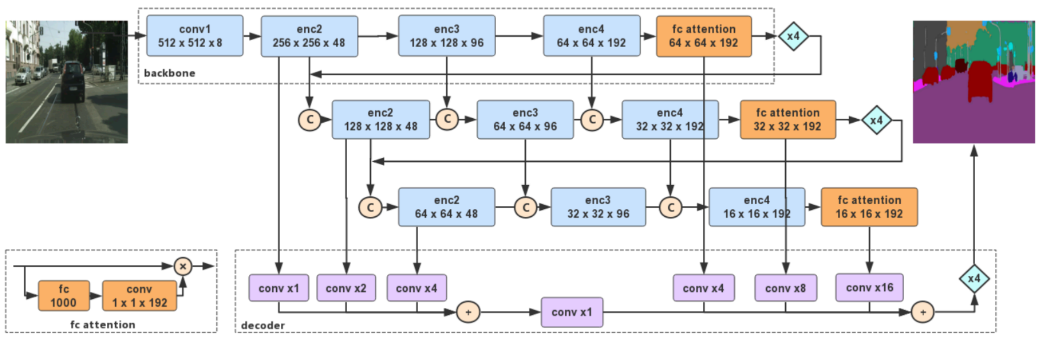

Figure 2.17: DFANet's micro and macro-architecture. Image taken from the original paper.

DFANet architecture follows encoder-decoder architecture. The former, incorporates three Xception backbones allied with sub-network aggregation and sub-stage aggregation modules. All the backbones were previously pre-trained on image classification tasks, in this case, ImageNet. In recent works, all the backbones are pre-trained in image classifications tasks, since it started to be a de facto that backbones pre-trained in classification are quite efficient at feature extraction. The latter, the decoder, was built using a low level of complexity. This is caused by the fusion of different feature levels inside the encoder part. To deliver final predictions, outputs from all the backbones are fused, via concatenation, refined by a set of convolutions and, finally, projected to higher dimensions by bi-linear up-sampling operations.
The produced network, which can be seen in detail on figure 2.17 was able to achieve state-of-art speed, but lacking by some points in terms of accuracy, on CityScapes and CamVid.

2.8.5 Searching for MobileNetV3

The paper [HSC+19] introduces the third model of the MobileNet neural networks' family. When optimizing the network, so that it can be deployed into mobile devices, the authors targeted four main points. The creation of complementary search techniques, crafting a novel non-linearity optimized for mobile platforms, designing the network more efficient and, finally, designing a lighter segmentation decoder.

The creation of this novel model was built upon previous efficient networks, such as SqueezeNet, MobileNet v1 and v2, Shufflenets and MnasNets. These kind of networks focus on having optimal accuracy and efficiency trade-offs and, while some are handcrafted, others were automatically designed. SqueezeNet uses 1x1 convolutions along with squeeze and expand modules. MobileNet v1 introduced depth-wise separable convolutions, which factorize spatial filtering and feature generation mechanism. The former, by deploying light weight depth-wise convolutions, and, the latter, by using a heavier 1x1 point-wise convolutions. The second version of MobileNet was designed around a resource-efficient block with inverted residuals and linear bottlenecks. They use compact representations at the input and the output, while expanding to a higher dimensional feature maps internally. MnasNet takes the MobileNet v2 and introduces lightweight attention modules into the bottleneck structure based on the squeeze and excitation mechanism.

With these architectures as foundation, the authors found that layers at earlier and final stages of the network were more expensive than layers on the middle of it. To optimize the computational cost of the network and incorporate more efficient blocks into the search space, the authors modified the latter layers, where the final features are produced. There, the authors introduced max pooling before the last 1x1 convolutional layer. After this change, the final set of features was computed using 1x1 spatial convolutions instead of 7x7. After this replacement, the final convolution became almost cost free. One side effect of the change, consisted on the possibility to remove the bottleneck projection layer, which latter reflects in a complexity and latency improvement.

One final change on the layer behavior focused on reducing the conventional first layer, where filters were projects to a 32 filter bank by a 3x3 convolution. With the usage of hard swish non-linearity, it was possible to use a 16 filter bank instead of the previous 32 filter bank without considerable dropping the accuracy.

MobileNetV3's authors used two NAS methods to automatically craft and, latter, optimized the network. First, they change the behavior the optimization criteria of platform-aware NAS. This change on the reward, was deployed to enable the RNN controller to take in account the needs of smaller mobile devices. After that, MnasNet-A1 model was used as seed and previously layer changes were added into the search space. The resultant model consists on a derivative architecture with a global architecture more robust and that can take better advantage of its resources. The novel architecture is then put under NetAdapt algorithms. The method is explained above, but as a refresher, it takes a given network and tries to completely eliminate filters inside convolutional layers to fit the model into latency-accuracy constrains. The final step of this two-NAS pipeline, consists on choosing the best architecture and re-train it from scratch.

The final key change of the method was the re-design of the non-linearities. Since the sigmoid function is not optimized in small devices, its use translates to higher latency. ReLU6 was fit as a replacement. The change was based on two reasons, which were the high degree of availability of ReLU in almost all devices and the capability of ReLU to

avoid potential numerical precision loss on quantized settings. Since the cost of using non-linearities goes down as the reduction of feature maps goes up, swish was used in the latter half of the network.
One final change was the replacement of the ratio of the expansion layer inside the bottleneck. The authors used 25% of the filter number. This changes, modestly increases the number of parameters, but without aggravation on the latency cost.
In the end, two versions of MobileNet v3 were created. One small and one large. The first, targets mobile devices with hard constrains. While the second, targets mobile devices with higher computation capacity.

2.9 Benchmarks

The paper [BCCN18] introduces several concepts and guidelines on how deep neural networks work, how to benchmark them and what kind of requirements are needed to deploy them on embedded devices.
When bench-marking deep neural networks, some useful metrics are:

- Accuracy: measures how precise the networks predictions' are when compared with the dataset's ground truth;
- Computational Complexity: qualitative metric and is relative to network's computational power requirements;
- Inference Time: average time spent on performing the network's task;
- Memory Footprint/ Usage: metric that benchmarks the allocated memory during the use of the network;
- Model Complexity: quantifies the total number of network's learnable parameters. As a note, learnable parameters and static parameters are different in the sense that learnable parameters are mutable and are the target of the training;
- Number of Parameters: quantitative metric that tells how many parameters the networks. Typically parameters are values represented as floating points.
- Operations Count: counts the number of adding and multiplying operations needed to infer a single instance;
- Power Consumption: metric used to calculate the electric energy needed to deploy the network. Also the higher the power consumption, the more difficult it is to deploy the model into embedded devices.

The following list presents some of the benchmarks found in the paper:

- **Accuracy Rate vs Computational Complexity vs Model Complexity:** model complexity and image classification accuracy aren't diretly related. In a similar way, model size and operations made aren't directly correlated. Model which use less parameters and tend to focus more on efficient operations (such as fusion) can achieve better accuracy results;

- **Accuracy Rate vs Learning Power:** density is a metric that relates the level of accuracy and the network's number of parameters. When deploying DNNs into embedded systems, models such as NasNet-A-Mobile and MobileNet-v2 are the ones with best density. The reason focus on their good enough precision levels, while maintaining a low number of parameters;
- **Inference Time:** super real-time, 60FPs, is easily achievable on desktop, while being extremely difficult on small devices, such as Jestons. As a note, delivery real-time inference with 60FPS is quite difficult for tiny models on low-resource devices, even thought the batch size used was 1;
- **Accuracy Rate vs Inference Time:** From the explored networks, the authors found that as the precision of the network goes down the inference speed goes up;
- **Memory Usage vs Model Complexity:** while different architectures families perform under different levels of memory consumption, the complexity of the model and memory usage is directly related. The trick for minimizing model's complexity and maintain a healthy use of memory consists on exploiting the maximum amount of RAM possible;

It is common to use neural networks for classification tasks as semantic segmentation backbones. When deploying these networks, it is possible to group them into different categories. The first, groups them by memory usage into three different groups. High, medium and low, where their respective usage is more than 1.4GB, 1GB and 0.7GB. The second, segregates them by computational capacity into three categories: half real-time, real-time and super real-time. The respective bounds are throughput under 15FPS, 30FPS and 60FPS.
The authors concluded that ResNets 18 and 50, and MobileNet networks are good for real-time throughput, while achieving good accuracies. This conclusion is reinforced by inspecting the field of semantic segmentation, where this kind of networks are employed as the backbone for feature extraction.

2.10 Conclusion

Several conclusion can be draw about semantic segmentation and model complexity optimizations. When deploying semantic segmentation, datasets that treat urban scene and day-to-day scene understanding are predominant. Networks can be build using human hand or through automatic mechanisms. The former, consists on using pre-determined conventions to optimize the micro and macro-architecture aspect of the network. In the latter, a controller navigates in a given search space and searchs for the ideal configuration. Those configurations can try to optimize direct or indirect aspects. Complexity reduction methods usually take a pre-trained network and employ mechanisms such as pruning, quantisation or encoding. In addition to those mechanisms, distillation can be used. Since it takes a bigger network to guide the training phase of a smaller one, the time needed tends to be bigger than those previously referred. Finally, attention focus on macro aspect of deep neural networks. It arranges the structure to better process information, which can lead to richer feature maps. These feature maps can be so much more fertile that the amount of layers and operations can be reduced. This leads to optimizations in time, memory and space. Efficient networks, employ multi-scale information coming from different paths or through skip connections. Since it is already known that architectures pre-trained on image classification tasks can perform better feature extraction, multi-scaling is obtained from the use of backbones pre-trained on classification tasks. With this information in mind, the next chapter is going to introduce the proposed method. To do so, the chapter contains which networks are going to be studied, which methods are going to be implemented and their respective chronological flow in the project.

Chapter 3

Experimental Setup

In this chapter, we describe each stage of the experimental approach followed in the thesis. To do so, we divide the work into the following stages:

1. **Overview:** small introduction where an outline for the experimental trial is presented;
2. **Baseline Model:** the specified model is going to be used as a reference for precision levels and as target when doing modifications using the chosen methods;
3. **Methods to benchmark:** list of the selected complexity reduction methods. Enunciation of the method by explaining how it works, what to expect and how the experimental setup is going to be composed.
4. **Preliminary Experiments:** description of early stage experiments. These experiments can be seen as initial trial step. The experimental setup is going to be backbone to further and tougher setups;
5. **Common Settings:**specification of common components to the experiments detailed on chapters 5, 6 and 7.

3.1 Overview

The experimental setup was designed to test several smaller versions of a network. The setup takes a full-precision network, which ideally is already pretrained, performs several optimization and outputs a fully optimized version. This setup optimizes several key points in the network, as an example, parameter count, memory usage and time needed to perform inference. Initially, a good deep neural network should be chosen from the plethora of existent ones. This model is going to be used as baseline and target to future modification via the found methods. After having a baseline model, various mechanisms will modify the network and create newer versions. These versions are going to be benchmarked.

3.2 Baseline Model

The first step, consists on choosing one or two models to be the project's baselines. Since the nature of the project lies on efficiency and resource savings, our baselines should be lightweight in terms of inference time and space needed to be save and be efficient in terms of operations and respective memory needed to deploy them.

During the researching phase, three architectures that tackle the refereed requirements were found: BiSeNet v1, BiSeNet v2 and DFANet.
DFANet is the fastest architecture. For images with resolutions of 2048x1024 it was able to deliver a throughput of 160 FPS, which consists on 4 more FPS when compared with the fastest version of BiSeNet v1 and v2.
On the other hand, all networks from BiSeNet's family can deliver more accurate predictions (around 1.3-4 percentual points on Cityscapes test set). The only found exception was BiSeNet v1 with XCeption39.
Authors of BiSeNet created two different architecture for each version of the deep neural network. These architecture can be differentiated in large and small using constraints as time needed for inference and number of parameters count. The version with XCeption39 as backbone can be seen as the small version, while the version with ResNet18 can be seen as the large version. When benchmarking all the version on Cityscapes dataset some conclusion were drawn:

- **Accuracy:**
  - **Small:** v2 dominates v1 in accuracy levels. On val split, a difference of 4.4% points were found (v1's 69% versus v2's 73.4%). And on test split, a difference of 4.2% points was measured (v1's 68.4% *versus* 72.6%);
  - In both val and test split, v2 reached a additional 4% accuracy points (69%,68.4% versus 73.4%,72.6%);
  - **Large:** v1 and v2 deliver similar levels of accuracy. On val split, the difference is a minimal 0.1% point (v1's 74.8% and v2's 74.7%). In the test split, the difference is higher, but not close enough to reach a percentual point (v1's 74.7% *versus* 75.3%);
- **Throughput:**
  - **Small:** v2 outperforms v1 (v1's 105.8 vs v2's 156 FPS) by a significant margin. This difference consists in a 50% gain in speed;
  - **Large:** v1 can deliver a higher level of throughput than v2. v1 and v2 were able to achieve, respectively, 65.5 and 47.3 FPS on images with a resolution of 2048x1024.

When comparing the small versions of the networks, v2 completely outperforms the v1 version. The key aspect of the benchmark was the degree of dominance in term of throughput, where v2 performs 50% faster than the v1 version. On the bigger versions, both networks are similar in terms of accuracy, but the first version can deliver predictions faster.
Another important remark focus on the existence of backbone. As a reminder, BiSeNet v1 uses XCeption39 and ResNet18 networks as backbones for feature extraction.
Upon looking at those aspects, it possible to conclude that large v1 is a good network for the project's experimental setups. Since large v1 benchmarks similar accuracy levels, when compared with the respective v2 version. Also, on inference speed, v1 is considerable

faster than v2. In sum, large v1 is the best deep neural network for our experimental setup.

3.3 Select methods to benchmark

This project will test three methods for reducing model's complexity: quantization, distillation and lightweight convolutions.

3.3.1 Weight Quantization

Limited precision techniques allow saving at inference and saving times. Since weights are represented using a fewer number of bits, they need a smaller memory space in order to be saved. The same happen at inference time. The only downside of computing using a reduced number of bits is the aggravated margin of error and, the consequent, loss of precision.
PyTorch framework comes with three different methods to quantize deep neural networks. Post-training dynamic (also known as weight-only quantization), post-training static quantization and quantization aware training.
The first, post-training dynamic, pre-quantizes all the network's weights. Activations are dynamically quantized during each inference. The positive side of the method consists on the dynamic clipping. This clipping consists on at each novel input, the ranges are calculated and adapted. In the end, the resultant predictions has a higher confidence degree since the clipping is more efficient.
Post-training static quantization is in the middle of the first method and fine-tuning mechanism. The first step, consists on fusing adjacent convolutional and batch normalization layers and activations, such as ReLU. Fusion allows only using only one operational kernel, which then leads to savings in memory usage, time needed to perform inference and a smaller quantization error. After fusion, observers are inserted at network's key points and a fine-tuning phase is deploy to calibrate them. After 100 mini-batches, the previously inserted observer are re-calibrated.
The third, and final method, is quantization aware-training (QAT). In this method, weights are stored and back-propagation is performed in FP32. Feed-forward passes are also performed in FP32, but simulating INT8 operations. By allowing the final loss to account for expected quantization errors, the model is able to achieve a higher degree of precision. In other words, it learns how to avoid those quantizations error.
Before setting up experiments, we can take some *a priori* guidelines.

- In terms of precision preservation capability, the order from less to highest is: post-training dynamic, post-training static and quantization aware training;
- As the upper-bond, from the method, goes up, the time needed to employ it also goes up;
- The less complex a network is, the more robust the quantization algorithm needs to be so that precision is affected the least.

Unfortunately, the first method is not available for convolutional layers. The authors stated that, at the time, it was only developed for LSTMs and RNNs. Therefore, the project's trials on quantization will be based on post-training static and quantization aware training.

3.3.2 Distillation

Distillation is a techniques where two networks are coupled together to create a student-professor pair. The student network learns from the professor predictions. During the training phase, the professor network is used to predict on the training set and feed the student network with information of how the bigger model behaves.
Student can rely singularly on the professor network, where it learns only how to mimic the bigger network's behavior. Or can learn from the dataset's ground-truth and, at the same time, from
Student can rely singularly on the professor network or can, also, take advantage from the datasets' ground-truth. The first mechanism, teaches the student how the professor behaves and inactivates it to mimic the professor. The second mechanism, uses the predictions to guide the student, but at the same time making room for it learning how to perform inference along and learn from its mistakes.
The method is typically deployed using a smaller student. This student can be a smaller network or tweaked version of the professor. The one-directional information flow can lead to the smaller student learn how the bigger version behaves and mimics. In other words, it is possible to see the student as a mini-professor. This means that the methods can optimize the model by reducing it. At the end, the novel architecture will have close enough precision levels, but with the additional advantage of lower usage of memory and spent time.
In the project, distillation is going to be deploy by performing a study on how the shrinking of layers can affect the precision levels of the model. Using a pair of student-professor networks, the student is going to have a reduction ratio where each layer filter number is upper-bounded to the respective professor's filter number.

3.3.3 Lightweight Convolutions

When designing a deep neural network, there are two key aspects that it is needed to take in account. Macro and micro-architectures. The former, takes focus on network's layer types and how they are arranged in the network's skeleton. The last, is referent to characteristics such stride value and kernel sizes.
In the current ecosystem, exists a enormous plethora of convolutional layers. They can range from light to heavy in terms of computational complexity.
To infer how layer types can lead to saving in memory usage during training and inference time, one step of the project will take layers that factorize the convolution process. Then, train the model and compare its performance with the original one.

3.4 Preliminary Experiments

Before benchmarking the chosen methods, a initial trial phase is performed to infer some key aspects for the realization of the project. This initial phase, will consist on taking a small outdated semantic segmentation model and perform changes, which in this case are going to be layer replacement. Then, train the network and benchmark it.
Even though the complexity of this part is lower, when compared with the rest of the method, this step deem its usefulness in the way that allow the tweaking of small aspects in the experimental setups and enables the possible to take some early stage conclusions.

3.5 Common Settings

The dataset and experimental environment used on the experiments in the chapters 5, 6 and 7 are detailed.

3.5.1 Dataset

In this experimental trial, the dataset used was Cityscapes. More information about it can be found in the datasets' section 2.2. Additionally, two key notes must be added.
The first remark consists on the class usage. In this work, only 19 classes were used. This was done to avoid class imbalance. In this dataset, several classes are note well represented. In other words, their appearance is close to nonexistent. This imbalance leads to inefficient learning. Additionally other scientific studies always use 19 classes, instead of the whole dataset's classes set.
The second, and final remark, takes emphasis on similar scientific experimental setup. BiSeNet v1 and v2 used Cityscapes dataset in their original training. If the network is not efficient in its nature, its study would not be possible, since the use of this data set with these networks require substantial computational resources. Therefore, if the setup is doable and the results are good, then its scalability to other case scenarios is easily performed.
When working with the dataset, only data in the format of images were used. Also, from the two different sets, coarse and fine, only images with fine annotations were used. As a side note, in BiSeNet V1 and V2 papers, only this set was used.
To discard labels outside the desired classes, the value 255 was attributed to their respective pixels in the ground-truth masks. On evaluation, these pixels were ignored.

3.5.2 Components

All the trials during this experiments were performed in a desktop with a NVIDIA® GeForce® RTX 2070 Super with 8GB and a Intel® Core™ i7-10700.
Experiments were designed and deployed using PyTorch.

Chapter 4

Preliminary Experiments

4.1 Replacement of Normal Convolutions with Depth-Wise Separable Convolutions

4.1.1 Introduction

Before benchmarking the chosen methods, a initial trial phase is performed to infer some key aspects for the realization of the project. This initial phase, will consist on taking a small and outdated semantic segmentation network, U-Net [RPB15] and perform changes on its macro-architecture structure. In this specific case, it will be the replacement of layers, where separable convolution will be employed instead of regular ones. After a standard training phase, the deep neural network is going to be evaluated. Even though the complexity of the experiment is lower when compared with the rest of the project's experiments, this step will be fundamental, since it will allow to find some key aspects for future tweaking in the experimental setups and, also, will enable the possibility to find some early stage experiments about precision-complexity trade-offs.

4.1.2 The architecture

The chosen architecture has a shape similar to the letter U, which is the main reason to its name (U-Net). This can be seen in figure 4.1. The original network has four contracting points and four expanding points. For each expanding point there is a skipping connections that delivers to it the output from its respective contracting point.
The contracting path is composed by down-sampling operations. The goal is to refine the hidden feature maps and then extract the most salient feature. On the other side of the network, expanding path, the extracted features are then interpolated to their correspondent space. This path is mainly composed by up-sampling operations. It is also possible to think of the architecture as a filter, where important features are highlighted and the superfluous ones are discarded.

4.1.3 Modified Architecture

In this experiment, two different architectures were created. The first deep neural network consists on a perfect clone, where standard convolutions were replaced by depth-wise separable convolutions. The second architecture follows a similar path as the first one, but with an increment in the number of utilised convolutions.
The second version of U-Net light, as seen in figure 4.2, was created given the poor results from the first light version. Since the replacement of standard convolutions by two depth-

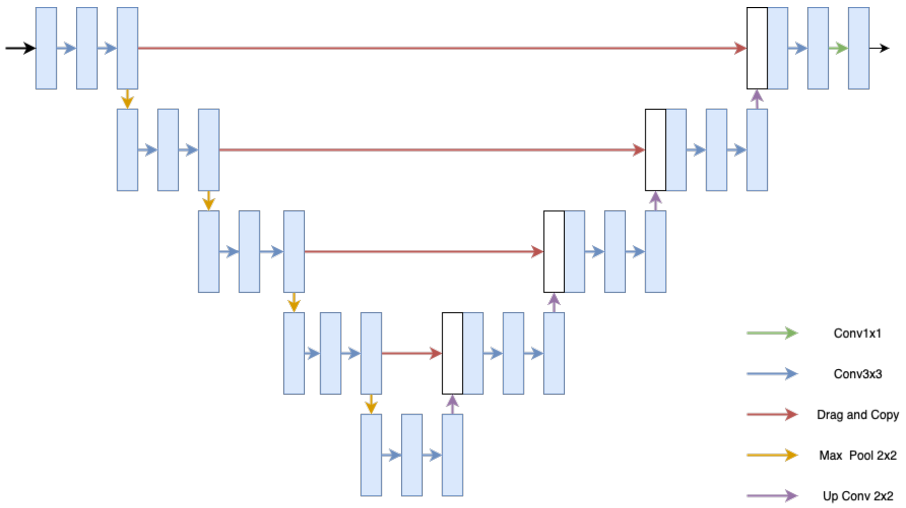

Figure 4.1: Original U-Net architecture. Four down-sampling operations on the contracting path and four up-sampling operations on the expanding path.

wise separable convolutions marked lower levels in complexity, the second version was crafted and lately evaluated.

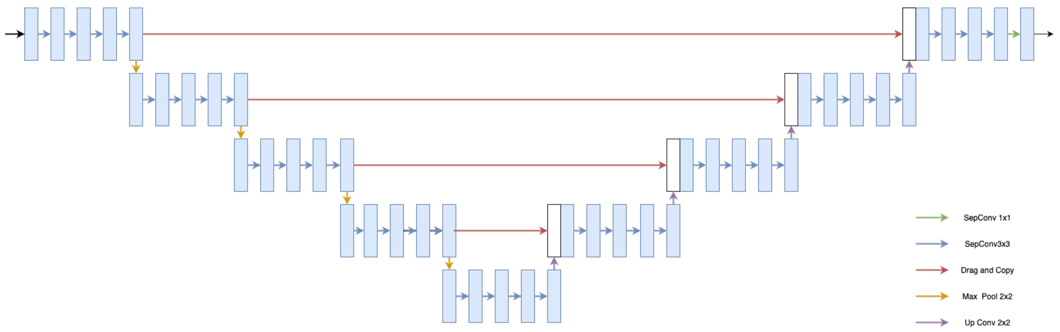

Figure 4.2: U-Net Light Version Two. The network has a similar architecture as the original. For every standard convolution in the original, this clone has two depth-wise separable convolutions.

4.1.4 Dataset

In this preliminary experiment, the dataset used was the same as in the SuperParsing paper [TL13]. It is composed by 2688 images with resolution of 256x256. The dataset focus outdoors scene understanding as in the Cityscapes dataset.
The main reason for the dataset's selection was the lower resolutions and focusing a similar context as Cityscapes. Since the image's quantity is lower and the resolutions smaller, when compared with the second dataset, it is deemed as ideal to deploying fast experiments and study some minor aspects for future scaling.

4.1.5 Training

The experimental details can be found in the table 4.1. When taking an attentive look some heterogeneity can be found. The reason for it is based on the ad-hoc approach when tweaking the finer aspects. In particular, two key details:

- **Number of Epochs:** initially, network were trained for 100 epochs. After finalizing the training for U-Net Light 1, it was concluded that training for more than 50 epochs was near to useless. Therefore, when training U-Net Light 2, the number of epochs was set to 50;
- **Used Batch Size:** different architecture need different levels of memory to run, which leads to a tuning of the batch size.

Criteria	**Description**
Epochs	100 epochs (U-Net & U-Net Light 1)
	50 epochs (U-Net Light 2)
Learning Rate	Starting at 0.001 and reducing it by a factor of 10 at 30th, 60th and 90th epoch
Weight Decay	0.001
Batch Size	4 (U-Net)
	3 (U-Net Light 1)
	1 (U-Net Light 2)
Optimizer	SGD Optimizer with Nesterov Momentum
Momentum	0.9
Loss	Cross-Entropy Loss

Table 4.1: Training parameters for the three version of U-Net architecture. Heterogeneity can be seen in the number of epochs and batch size.

4.1.6 Results

Upon taking a look at table 4.2, it is possible to conclude that from U-Net to U-Net Light One a reduction on the network's complexity was achieved. While having a smaller batch size, U-Net Light One was able to finishing the training phase in 76.9% of U-Net's training time.
For U-Net Light Two, conclusions on training time cannot be taken, since the heterogeneity is too excessive.
U-Net Light Two requires a higher amount of resources, since the achieved training time was close to U-Net One, while the training setup being considerably less complex. In this case, half of the epochs and a batch size of one, instead of three.

Architecture	Batch Size	Epochs	Time Spent on Training (s)
U-Net	4	100	54010
U-Net Light 1	3	100	41545
U-Net Light 2	1	50	45644

Table 4.2: Time spent on training each model.

The results presented in table 4.3, lead to the conclusion that the original U-Net model has better precision, around 10% points, when compared with the best light version. This precision levels carry a toll in the model's complexity, since the original model is around 4 and 3 times bigger than Light One and Light Two, respectively.
From the addition of one more depth-wise separable convolution for each replaced layer, the network was able to delivery an improvement of 14% points. While increasing in complexity, the model was able to be have similar throughput in GPU. On CPU, the toll was enormous.

Architecture	Best mIoU (%)	Params (x 10^6)	Size (MBs)	Inference on CPU (s)	Inference on GPU (s)
U-Net	0.491	31.0	124.2	0.25	0.032
U-Net Light 1	0.252	7.4	29.9	0.22	0.011
U-Net Light 2	0.396	12.4	49.9	0.38	0.013

Table 4.3: Results for the performed experimental benchmarks.

4.1.7 Conclusion

At the end of this experiment, some key points can be taken. Speed, space and needed memory can be optimized, but with a trade in the model's precision levels. Having a smaller number of parameters is not indicative of the model's performance in terms of inference time. In the present time, PyTorch does not provide an efficient implementation for this kind of layers, which leads to having a higher spent time in Light Two.
In the next chapter, experiments on the weight representation will be performed.

Chapter 5

Weight Quantization

5.1 Introduction

The first experiment touches the mechanism used to represent the weights of a given network. The main goal is to infer how reducing the number of bits to represent them, affects the network. To figure how the network is impacted, some metrics will be measured and exhibited here.
First, theoretical foundations 5.2 will introduce the concepts and methods. Additionally, it will have a list of reason explaining why the used method was chosen. Then, the experimental setup 5.3 will add fundamental information on which neural network was used. As a reminder, details on the dataset and other components can be found in section 3.5. The next section is implementation overview 5.4, which consists on the description and some minor aspects referent to the implementation of the method. After that, experiments will be showed and analysed in section 5.5. The last chapter conclusion 5.6, will deliver an outline about the whole chapter and the method.

5.2 Theoretical Foundations

Weight quantization consists on reducing the number of bits used in the weight representation. In this experiment, PyTorch quantization framework is going to be used.
PyTorch's quantization modules offer the opportunity to represent the model's weights in INT8 base. When comparing with standard 32 floating point precision, INT8 occupies four times less space. *A priori*, it is possible to assume that the improvements in model speed and reduction in model size and memory bandwidth should be around that ratio.
This method can be seen as a form of clustering, since different weight values are mapped into the same value. For example, 2.114 and 2.139 are mapped to 2, when passing to integer representation.
The framework offers three types of quantization methods. Two for conversion post training (dynamic and static) and one to convert during the training phase (training-aware).
Post-training dynamic quantization converts the model's weights at the start. Activations are converted during the run-time as the model is requested. The input is scaled and distributed into bins. The scaling weights are fine-tuned as the model is requested In other words, as the predictions are requested, the weights are calibrated and, therefore, converge to the optimal configuration. This method is the one that degrades the precision the most. Also, it does not gain any saving space, because weights are still saved in its original representation and is not (yet) deployed for convolutional layers. In sum, this method is not going to be tested, because the chosen baseline models are mainly composed

by convolutional networks. Large networks, which possess a high degree of redundancy are the main target for dynamic quantization.

Post-training static quantization is the second method present in the framework and targets medium-to-large size models. It works as the method above, but in this case the calibration is made in amortized time after the training phase. Since calibration happens in a single phase, the scaling vector are more optimize and precision tends to be higher. The model needs to take in account quantization opening and closing points, respectively named stub and destub. The former, takes FP32 or FP16 weights and converts them into INT8 weight. The latter, performs the inverse operation. These points allow the existence of hybrid models, which in turn allows for more refined models. Inserting these points is not a trivial tasks to perform and needs special attention.

Calibration must be performed on a platform with the same backbone as the target device. They can be either x86 CPU or ARM CPU. The reason centers around the reduced precision tensor matrix libraries. If they do not match, the model cannot be deployed.

Fusion takes tuples or triplets of convolutional layers or linear layer, batch normalization and ReLU activations and ”fuses” their kernels. This operation avoids the usage of middle level kernels between operations, which in turns leads to savings in memory bandwidth and computing time.

The third and final method, Quantization Aware Training, merges the training phase into conversion phase. This method provides the best results of the three tested quantization approaches with the downside of being the more complex in terms of implementation. Instead of using standard stubs, fakeQuantize stubs are inserted inside the network. Layer inside these stubs and destubs mimic the operational process of INT8, while, in reality, the weights are being represented using FP32. The game changing aspect of this method is the introduction of the quantization error on the training phase. By allowing the optimizer to take in account this error, the model is able to converge and find solutions closer to the optimal configuration. While delivering the best precision results, the method requires a longer training phase.

Quantization Aware Training provides the best accuracy results, while needing a longer deployment time. Static Quantization finds itself in the middle ground, in terms of accuracy and time needed. On the other side, dynamic quantization results in a big degradation and cannot be implemented for many architectures.

Some guidelines should be taken in account. Heterogeneous models have different levels of quantization tolerance. The bigger the model, the higher level of tolerance. Hybrid models can be the optimal configuration for quantization. Balancing which layer should be and which should not be quantized is a delicate tasks. Finally, layers are not implemented in equal form. It is possible to add singleton tensors to other tensors in regular PyTorch, while in the quantization models is impossible. In this last case, a new tensor with matching dimensions must be created.

5.3 Experimental Setup

In terms of deep neural network, BiSeNet v1 and v2 were chosen. For each version of the networks, the larger versions were selected. This follows the principles that larger networks tends to be more robust to reduction modifications.
Recalling the information on section 3.2 about these convolutional neural networks, V1-Large offered similar accuracy levels when compared to V2-Large. In terms of throughput, the version V1 was faster by close to fifty perceptual points. Therefore, it is the main network for the experimental setup.
In conclusion, for the experimental setup, V1-Large and V2-Large are the networks that offers the best trade-offs in terms of accuracy-resource consumption and better *a priori* robustness to modifications.

5.4 Method Overview

The first step consists on verifying which methods are available for the chosen architectures and for what kind of device they can be deployed.
As a reminder, PyTorch Quantization module offers three methods. Post-training dynamic, post-training static and quantization aware-training. For network composed by convolutional layers only the last two are applicable. Also, the module only is available for CPU architectures. If the goal was to deploy quantization on GPU, TensorRT software development kit would be the best method.
The goal for this experimental trial is to benchmark the model using inference on CPU upon the usage of post-training static and quantization aware-training.
Since the benchmark will be performed on the desktop where the network is going to be calibrated/ trained, attention on compatibility of CPU is not needed.
For each method, there is a need to perform some changes in the network and its workflow. The first change consists in the insertion of the quantization stubs and de-stubs.
Initially, a stub was inserted in the beginning of the network allowing the data to be quantized upon the entrance of the network. The destub was inserted at the end of the network, which delivers the prediction map in the original representation. This kind of method is named one-shot quantization, since the whole deep neural network is quantized.
After deploying this initial solution, a problem arose, which would deem impossible to one-shot quantize the whole model. The problem was found on layers that could not be quantized. Two solutions were found. First, replace these kind of layers by similar ones that would perform the same kind of task, but were available for quantization. Second, maintain the model hybrid and inserting stubs and destubs after each layers. Layers that could be replaced were replaced and where layers that could not be replaced were encapsulated inside quantization stubs and destubs.
The next step consisted on translating the model. The novel models were similar to the original ones with the except of the existent stubs and destubs. From the original to the novel models, layers that were found in their respective correct spot, would be copied. The

others would be freshly initialized.
After the translation, a fine-tuning phase must be deployed to allow the recovery of precision in the model.
This process is performed in the two quantization mechanisms. The only difference is the kind of stubs that are inserted inside the network.
In the post-training static, a calibration phase is performed using instances from the training phase. This calibration step is similar to a fine-tuning phase. After calibration is made, the model is converted to INT8 representation. From that model, all inferences are performed using quantized weights.
In quantization aware-training, it was preferred to perform a mix of fine-tuning and training, instead of following a train from scratch. One thing to take in consideration is that the training epochs using these methods are much more time consuming, which leads to eliminating the scenario of a train made from scratch. The goal is to take the translated fine-tuned network and perform a training plus calibration using the training set and finally convert it.

5.5 Results

Upon taking an attentive look at table 5.1 it is possible make some conclusions about the method and how it impacted the network. As a reminder, only benchmarks for CPU are used, since the PyTorch Quantization framework does not offer the option for deployment using the GPU.

BiSeNet V1 Version	mIoU (%)	Size (MBs)	Training Time Per Epoch (h)	Inference CPU (s)
Original	61.6	56.6	-	2.31
Post-Training Static	20.4	13.8	-	1.11
Quantization Aware-Training	36.6	13.8	3.84	1.16

Table 5.1: Bench-marked results on accuracy, required saving space, training time per epoch and inference on CPU.

First of all, for post-training static and quantization aware-training, inference levels are close. The former method improved the inference time by 2.08 times, while the latter upgraded it by 1.99 times.
In terms of required saving space, both methods need the same amount of memory, which was 13.8 MBs. This value consists on models being 4.09 times lighter than their counterpart.
In terms of resources, the resultant models are faster at performing inference and lighter in terms of required saving space. On the down-side, their accuracy levels dropped by a significant margin. The original model scored 61.6 percentual points on mIoU, while the new model marked 20.4 and 36.6 percentual points on post-training static and training-aware, respectively.
As expected, by the benchmarks it is possible to conclude that calibration and training at the same time (QAT) is able to deliver higher precision levels than first train, then per-

form calibration (Post-Training Static). Additionally, the time required for one epoch of training was around 3.84 hours, which is considered as excessive.
In resume, inference time and required saving space was highly improved, while the training time and accuracy levels took a significant blow. As a remark, the QAT's model was trained in a fine-tuning alike mechanism. If training the whole model from scratch was to be considered, then the require time would be higher.

5.6 Conclusion

From the three quantization methods offered by the PyTorch Quantization framework only two could be deployed - Post-Training Static and Quantization Aware-Training. The other method was designed for small RNNs networks, which was out of the project's scope. These two methods take a model and convert it from its original weight representation to a lower representation level. Calibration during training time offers architectures with higher accuracy levels than their counterpart. This is due to the impact of the quantization error on the training convergence. In terms of improvements, both methods deliver the same levels in terms of inference speed and required saving space.
In the next chapter, distillation mechanisms using a pair of professor-student networks are introduced and a novel experiment is going to be detailed and analysed.

Chapter 6

Distillation

6.1 Introduction

The second scientific experiments tries to distil knowledge from a deep neural network into a smaller version of it. The goal is to study the computational complexity, speed of inference and accuracy of the smaller version. To do so, experiments using standard knowledge distillation and stage-by-stage are performed.
The next section consists of theoretical foundations 6.2, where concepts and methods are illustrated. The additional aspects of the experimental setup are described in 6.2. An overview of the implementation is seen in section 6.4. After explaining the implementation, results are shown and analysed in section 6.5. Finally, overall conclusions about the methods are taken in section 6.6.

6.2 Theoretical Foundations

While searching on the topic of distillation, four main methods were found: knowledge distillation, stage-by-stage distillation, relational distillation and quantized distillation.
The first method on distillation was named Knowledge Distillation. Here, two networks were used. One bigger and previously pre-trained, the professor, and a smaller one, which would be put into a training phase with the guidance of the professor. The smaller network emulates how the professor performs its inference by trying to match the bigger network's logits after each instance. Additionally, the professor can instead only leverage the training of the student. By using this mechanism, the student is able to learn how to deliver precise predictions, while taking "advice" from the professor. Ideally, the precision levels from student networks will be better than precision from copy of the student, which did not receive guidance from the professor.
Stage-by-Stage extends the previous formulation by splitting the architecture into several break-points and apply, sequentially, knowledge distillation. The authors advise that ideal configurations would use down-sampling points to become the method break-points. This split enables the student to better mimic each part of the professor, which in turn transforms in a overall mimic process. The final key-point consists on the possibility of introducing heterogeneity. The only requirement is the matching of the logits. If the logits match, previous computation flows are useless to the student.
Relational Distillation takes inspiration on natural language processing such as the cosine distance. They take two or more instance from the same training batch and calculate how they correlate. Then the student tries to learn how to obtain the same type of correlation.

In this case, the optimizer search for configurations that minimize the difference between professor and student's correlation.
Quantized distillation merges the concept of knowledge distillation and quantization. Here, the final output, should be a copy of the professor model, where the different would be how the professor's weights are represented. These kind of pipelines, tend to quantize the model, then calculate the loss, back-propagate the error, and as last step de-quantize the model to the original representation. The advantage of this method consists on introducing the quantization error inside the training phase, which means that the optimizer takes it into account and tries to converge to the optimal configurations.
With this in mind, the focus of the experimental setup will be the comparison on knowledge distillation and stage-by-stage distillation. In the end, the goal is to see if by using more break-points it is possible to recover or beat the precision levels of the original network.
The other two methods were discarded by the following reasons. Relational distillation would take a enormous toll in resources, while delivering a marginal gain on precision levels. The method requires a batch size higher than 1 and as the size of the batch grows, the higher the resources dispensed. The reason centers on the combinatorial explosion, since the method utilizes every possible combination of instances in the current training batch. Quantized distillation requires a working pipeline of quantization. The previous experience was not able to deliver good enough results in terms of accuracy to enable the usage of quantization plus distillation. With the inability to insert quantization in this method, it is possible to better isolate the filter removal part and perform a more exhaustive trial on its impact.

6.3 Experimental Setup

Similarly to the experiment done in the previous chapter 5, the networks used were BiSeNet V1-Large and V2-Large.
As a reminder, Large versions were preferred because of being efficient, while more computationally demanding than their lighter version. This leads to a better robustness to modifications, such as reducing the number of filters.
In addition, V1-Large is more efficient than V2-Large, since it reaches a higher accuracy score, while computing faster and with less memory consumption.

6.4 Method Overview

Considering the information above, the comparative study is going to use the first version of BiSeNet to perform two trials on the two parallel paths. The first trial, is going to benchmark the gap in knowledge and stage-by-stage distillation. The second, will see how the increase of professor impact the distillation professor.
As a reminder, neither of the distillation points can use the final prediction maps. Therefore, hidden feature maps will be used to guide the training process by being the quality

Training Method	Filter Ratio (%)	Training Time (s)	Accuracy (%)
Original	1	-	75.56
KD	0.8	2593	74.71
SBS		13036	74.70
KD	0.6	2556	74.71
SBS		12990	74.66
KD	0.5	2532	74.66
SBS		12678	74.70
KD	0.4	2523	74.71
SBS		12767	74.70
KD	0.3	2580	74.69
SBS		12741	74.74
KD	0.2	2615	74.72
SBS		12617	74.69
KD	0.1	2524	74.41
SBS		12574	74.65

Table 6.1: Score on training time per epoch and accuracy, when performing distillation on reduced filter number networks. KD refers to knowledge distillation method and SBS refers to stage-by-stage distillation mechanism. Experimental was deployed using the network's spatial path.

measures.
In detail, the first will benchmark the gap between two distillation methods, knowledge distillation and stage-by-stage distillation. In order to do so, the filter ratio between original and newly crafted going to decrease and distillation will be used in the training. After training, benchmarks will be performed.
The second, is going to compare the quality of the models, when using two professors, instead of one. In this case, the backbone on the context path is going to be the model to use. In this case, a ratio of 75% was chosen, and two version of ResNet was used. One was the version inside the trained BiSeNet v1 version, while the other was a bigger version that can be found inside the PyTorch model zoo. To merge the information from this two models, a weight loss was used. This weighted loss allows versatility, in the sense that, by changing the weighting factor, the student learning process leans to the one with bigger influence.
Next, the results for this two trials are going to be presented.

6.5 Results

The tables 6.1, 6.2 and 6.3 are related to the study on decreasing of filter size employing two distillation methods, while table 6.4 refers to the experiment on the number of professors.
When looking at 6.1, it is possible to conclude that the training time is considerably higher using the stage-by-stage distillation. The reason assents on the total number of epochs performed. Since the method trains each stage independently, the requirement amount of time to train is higher. In terms of accuracy, both methods achieved a similar level for all the filter ratios.
The bigger difference, can be seen when going from the original model to our first version,

ratio of 0.8, for both methods.

From table 6.2, it is possible to see the direct improvement on CPU, when looking at isolated inference speed, by reducing the filter sizes. From the original to the version with 0.1 as filter ratio, an improvement of 3.2 times was achieved. On GPU, the isolated benchmarked were equal for all the filter ratios.

Similarly, when testing the models in a whole, a toll on inference time can be seen. The new models score similar speeds on CPU. In terms of GPU, the require time for benchmarking are equal.

Training Method	Filter Ratio (%)	Inference Modified Part (s)		Inference Whole Architecture (s)	
		CPU	GPU	CPU	GPU
Original	1	0.1419	0.0007	1.2035	0.0052
KD	0.8	0.2435	0.0007	2.2734	0.0060
SBS		0.2519	0.0007	2.3212	0.0060
KD	0.6	0.1924	0.0007	2.1957	0.0060
SBS		0.1952	0.0007	2.2488	0.0060
KD	0.5	0.1850	0.0007	2.2032	0.0060
SBS		0.1893	0.0007	2.2325	0.0061
KD	0.4	0.1332	0.0007	2.1595	0.0060
SBS		0.1460	0.0007	2.2542	0.0061
KD	0.3	0.1288	0.0007	2.2860	0.0060
SBS		0.1088	0.0007	2.1827	0.0060
KD	0.2	0.0897	0.0007	2.3060	0.0060
SBS		0.0798	0.0007	2.1311	0.0059
KD	0.1	0.0431	0.0007	2.0580	0.0060
SBS		0.0435	0.0007	2.0949	0.0059

Table 6.2: Inference times both on CPU and GPU. Modified part and whole network focus on measuring the sub-module where changes were applied and testing the network as a single unit, respectively. KD refers to knowledge distillation method and SBS refers to stage-by-stage distillation mechanism. The experiments targeted the network's spatial path.

When looking at the size and parameters on table 6.3, it is possible to see the impact of reducing the number of filters. On the isolated part, when the sub-module uses 10 times less filter, the parameter ratio is close to 2.8%. In this case, the reduction of filter is extreme.

When seeing the network as a whole, even though the ratio is extreme, the impact on the network's footprint is close to minimal. All the filter ratios impact he network as an whole in less than one percentual point.

A direct conclusion from this table, is that the spatial path has a minimal footprint on the whole training and inference steps.

For the second trail, a filter ratio of 0.75 was chosen. Results can be seen in table 6.4. The time required per epoch increased considerably. When distilling with one professor and with two professors the network required 2356 and 4554 seconds, respectively.

When isolated, scores on inference, size and parameters count are equal between models originated from the two methods. From the original to the reduced model, inference on CPU is 1.4 times faster and occupies less 43.75% space in disk to be saved. On GPU, the required time for inference is equal.

Training Method	Filter Ratio (%)	Modified Part (s)			Whole Architecture (s)		
		Size (KBs)	Parameter Count	Parameter Ratio (%)	Size (MBs)	Parameter Count	Parameter Ratio (%)
Original	1	378.4	65904	100	56.500	14092800	100
KD	0.8	255.6	61405	66.77	56.379	14062237	99.78
SBS							
KD	0.6	157.2	36926	40.15	56.281	14037758	99.61
SBS							
KD	0.5	120.5	27689	30.10	56.244	14028512	99.54
SBS							
KD	0.4	83.1	18531	20.15	56.207	14019363	99.48
SBS							
KD	0.3	57.6	12093	13.15	56.181	14012925	99.43
SBS							
KD	0.2	33.5	6220	6.76	56.157	14007052	99.39
SBS							
KD	0.1	19.1	2590	2.82	56.143	14003422	99.37
SBS							

Table 6.3: Benchmarks on the size of the resultant models. Modified parts refers to the sub-module where the changes where employed. Whole network treats the network as a single unit. KD refers to knowledge distillation method and SBS refers to stage-by-stage distillation mechanism. Spatial path was used as target module.

When testing the network in a whole, CPU's inference is 1.26 times faster. On GPU, inference is close to the original. In this case, the toll is close to minimal. Here, the ratio between novel and original is 67.25 percentual points.

The only downside, focus on the loss of precision. By observing the table, it is possible to see that the network lost is ability to perform inference. Since the network's precision was completely lost, the different between these two methods is circumstantial.

			Context Path Original	Context Path Single Professor SBS	Context Path Dual Professor SBS
Parameter Ratio (%)			100	75	75
Training Time (s)			-	2356.42	4554.37
Modified Part	Inference	CPU (s)	0.630	0.4528	0.4453
		GPU (s)	0.0028	0.0030	0.0030
	Size (MBs)		44.8	25.2	25.2
	Parameter Count		11176512	6549296	6549296
	Parameter Ratio (%)		100	56.25	56.25
Whole Model	Inference	CPU (s)	1.2035	1.0383	1.0383
		GPU (s)	0.0052	0.0054	0.0053
	Size (MBs)		56.5	38.0	38.0
	Parameter Count		14092800	9465584	9465584
	Parameter Ratio (%)		100	67.25	67.25
Accuracy (%)			75.5	0.293	2.061

Table 6.4: Complete benchmark's score for the modified BiSeNet's context path. KD refers to knowledge distillation method and SBS refers to stage-by-stage distillation mechanism.

6.6 Conclusion

Reducing the number of filters can be seen as a direct method to reduce model's footprint. On the other hand, some deep learning frameworks, offer the possibility to deploy pruning. Even though pruning is performed, PyTorch still requires the same amount of resources. Therefore, cutting down the number of filters, which is similar to filter pruning, is better than pruning in terms of efficiency.
From the experiments deployed, it is possible to conclude that, when tweaking the number of filters, the accuracy can be influenced in drastic manners. Even though the reduction ratio was considered minimal, the accuracy level dropped by a percentual point. When comparing knowledge and stage-by-stage distillation, it is possible to see that the required training time is considerably higher when using the latter method, while delivering similar results in terms of model accuracy.
When taking an attentive look at the second trial, it is possible to see the direct improvement from cutting the number of filters. On the table, it possible to see that inference, parameters and required disk space decreased. The downside was seen on the accuracy, which was almost lost.
On the next chapter, the last experimental trial, which tackles layer replacement, is going to be detailed and results shown.

Chapter 7

Lightweight Convolutions

7.1 Introduction

The third and last chapter about experiments extends the chapter about the preliminary experiment. The goal is to study how layer replacement can affect the behaviour of a given network. Typically, efficient layers tend to factorize the operation process. In other words, instead of employing convolutions a single time, they split it into several steps, which leads to having several smaller operations instead of a single layer.
The next section, analyses the fundamental concepts 7.2. Then the experimental setup is described in 7.3, where information about the deep neural network used is added. The employed method and obtained results are presented, respectively in sections 7.4 and 7.5. The last part, conclusions 7.6, analyses the experiment in a succinct way.

7.2 Theoretical Foundations

The last studied approached in this project consists on replacing standard convolutions by factorized convolutions. As the name says, factoring a convolution consists in separating the calculations in several steps. The calculations performed, in the end, will be equivalent to the complete one-step version. As an example, spatial separable convolution divides the classical 3x3 kernel from standard convolutions into two steps. Their kernels have shape of 3x1 and 1x3 respectively. By treating width and height individually, the number of performed multiplications decreases. This reduction in complexity leads to a direct speed up in the network.
When factorizing convolutions, special detail should be taken in the size of the kernels. Several kernels cannot be divided, which makes impossible to factorize the convolution. Depth-wise separable convolutions are not affected by this factor, which makes them extremely transversal and available to replace any kind convolution, independently of the kernel size.
As an example, standard convolutions with 3x3 kernel can be seen as a cube. This cube navigates from the top left to the right bottom, while driving through the front to the back of the feature map. This type of navigation takes an enormous quantity of resources, because a lot more of multiplications need to be performed. On depth-wise convolutions, the kernel navigates through top left to right left to all the filter individually. Then, group of filter are merged in the x-axis. This combination takes less resources, since the navigation is made more efficiently and redundant operations are avoided.
Depth-wise separable convolutions splits the classical convolutional layer into two parts, depth-wise and point-wise. The former, applies convolutional filter for each input chan-

nel. The last, performs a linear combination of the previous output. This process can be seen as, first, operating on each image interpretation (channel), then combine information from those interpretations to create a newer and condensed version.

These kind of convolutions allow for delicate trade-offs. The reduction in the number of parameters is high, therefore networks can become extremely compact. Unfortunately this can lead to cases where the network is unable to learn. On the other side, if the model can be replicated and the correspondent layers switched, while maintaining precision levels, the resultant architecture is extremely efficient.

As a side note, point-wise convolutions can be used to increase or decrease the depth of a feature map. Also, it is common in deep neural networks to apply activations, such as ReLU after each point-wise convolutions. In other words, this layer can be used to increase the number of times activations are used, while maintaining the number of parameters close to the original number.

7.3 Experimental Setup

Following the experiments illustrated in chapters 5 and 6, the preferred networks for this last trial were the same.

V1 and V2 Large were preferred instead of their respective light version, because of the previously mentioned degree of robustness when lowering their computational complexity.

Additionally, V1-Large takes a more important role, since this architecture is able to score a similar precision level, while being considerably more efficient than its counterpart.

7.4 Method Overview

The method will be divided into three different steps.

The first part, a study on parameters count will be performed. In that experiment, kernels with size of 1x1 and 3x3 will be used and for each kernel, while incrementing the number of filters per layers (3 to 64, 64 to 128, 128 to 256, 256 to 512 and 512 to 1024). From this study, conclusion should be draw about layer replacement and how they will affect the network.

The second part, will take the original network and replace layers inside pre-determined breaking points. Those breaking points will encapsulate each path of the bilateral network, all layers from the start until the end of the paths merging point and, finally, the whole network. From this study, it will be possible to infer which sections of the network contain more parameters and subsequently are more robust to layer replacement.

Finally, the third part, will use the final version of the network, with whole layer replacement, to performed a more detailed study. To conduct this study, benchmarks on training time, required saving space, accuracy and inference (both on CPU and GPU) will be used.

7.5 Results

In table 7.1, it is possible to see the first part of the initial trial. The goal was to find how standard and depth-wise relates in terms of parameters count when using kernels of size 1x1. The first row, when initial and last channel number are, respectively, 3 and 64 is the only where a significant different can be seen. A ratio of 76.2%. For all other rows, the variation in the count is almost non-existent and as the size grows, the gaps between them decreases.

Convolution Type	In Channels	Out Channels	Number of Parameters	Ratio
Normal	3	64	256	0.7620
Depth-Wise Separable			195	
Normal	64	128	8320	0.9923
Depth-Wise Separable			8256	
Normal	128	256	33024	0.9961
Depth-Wise Separable			32896	
Normal	256	512	131584	0.9981
Depth-Wise Separable			131328	
Normal	512	1024	525312	0.9990
Depth-Wise Separable			524800	
Normal	1024	2048	2099200	0.9995
Depth-Wise Separable			2098176	

Table 7.1: Comparison between the computational footprint of normal *versus* depth-wise separable convolutions with kernel size of 1x1. The number of parameters for each layer was obtained via a small program.

The second part of the first trial, focus on finding a relation between the parameters count between standard and depth-wise convolutions with kernel size of 3x3. The results can be seen on table 7.2. As a reminder, in the previous trial, as the number of channels grow, their parameters count tend to approximate a relation of 1. Now, the opposite happens, even if the margin is small. As the number of channels per layers grow, the parameters relationship tends to dimishing. A crude estimate of 11 percentual points can be seen when relating the size of depth-wise and standard convolution. After this trial, it is possible to see a good optimization in saving space when performing layer replacement.

Convolution Type	In Channels	Out Channels	Number of Parameters	Ratio
Normal	3	64	1792	0.1222
Depth-Wise Separable			219	
Normal	64	128	73856	0.1187
Depth-Wise Separable			8768	
Normal	128	256	295168	0.1149
Depth-Wise Separable			33920	
Normal	256	512	1180160	0.1113
Depth-Wise Separable			133376	
Normal	512	1024	4719616	0.1121
Depth-Wise Separable			528896	
Normal	1024	2048	18876416	0.1116
Depth-Wise Separable			2106368	

Table 7.2: Comparison between the computational footprint of normal *versus* depth-wise separable convolutions with kernel size of 3x3. A small program was used to count the number of parameters per layer.

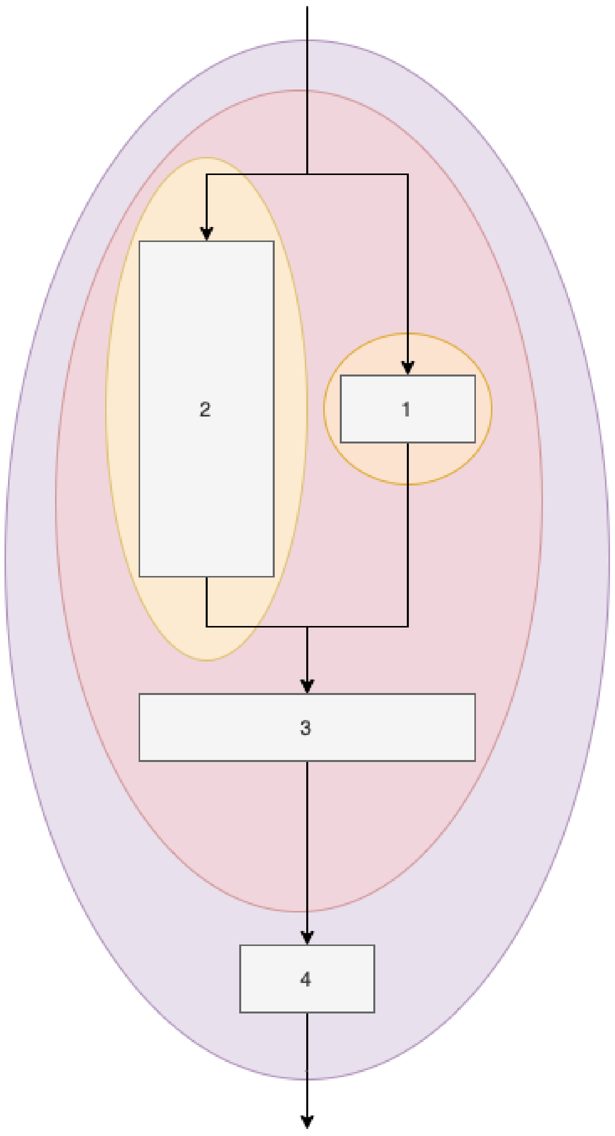

Figure 7.1: The rectangles can be interpreted as the sub-modules of the network. The ellipses represent the various areas where the replacement happened. Number one and two represent segmentation and detail path, respectively. Number three represent the fusion path and number four the final computation area. Rectangles inside the ellipse means that the module takes in part in a replacement trial.

The network can be seen as a intersection road. Inside the road, two different lanes exists, where different kind of information is being processed. The two roads intersect and a final refinement is performed to deliver the final predictions. In table 7.3 is possible to see the results for the second trial. The focus was to split the network into four different regions, where replacement would be performed. Each one of the parallel paths, the merging points and the whole network.

The savings of layer replacement inside the paths was 1.9 and 2.4 MBs. From the start to the fusion point was 12.3 MBs and, in the whole network, 16.5 MBs. With this information, it is possible to conclude that the merging point and final computations are the heaviest points in the architecture. The best version, has a savings of 16.5MBs, which consists on a 85% ratio.

Even though improvements can be seen in the saving space, these kind of improvements does not appear in the inference time. In CPU benchmarks, only one version was faster. This version replaces networks in the parallel paths and the merging point. When replacing layers inside the whole network, inference tends to be slower in CPU. The slowing factor resides in the range of 4.7 and 13 percentual points. GPU benchmarks shows similar values for all the crafted version, except for the one, where all layers were replaced. On this version, the inference time was slower in 14 percentual points when compared with the original version.

Changes	Size (MBs)	Inference Time (AVG 40 Images)	
		CPU	GPU
-	111.1	1.186	0.0070
Replacement on Detail Path	109.2	1.277	0.0069
Replacement on Segmentation Path	108.7	1.242	0.0070
Replacement on Paths and Fusion Point	98.8	1.161	0.0073
Complete Replacement	94.6	1.347	0.0079

Table 7.3: Study on the gradual replacement of standard by depth-wise convolutions on BiSeNet-V2 Large. The idea is to break the original network into several parts and perform benchmarks on the impact of isolated and global layer substitution. Inference time were measured as the average of the time needed to perform inference on 40 instances from the training set.

Tables 7.4 and 7.5 shows the complete benchmarks for the version where was performed a complete layer replacement. Inference and required saving space was previously detailed. In terms of training time, a significant reduction can be seen. The benchmarked training ratio was around 80 percentual points. In other words, for each 100 minutes dispensed on the original version, the project's version took 80 minutes. In terms of precision, the model was able to score a value of 50 percentual points in mIoU. This score is good, since the module's original precision was 15 percetual points above.

Changes	Training Time (s)	Size (MBs)	Ratio (%)	mIoU (%)
-	47181.82	111.1	-	65.95
Total Replacement	38583.57	94.6	85.15	50.01

Table 7.4: Final benchmarks for complete layer replacement on BiSeNet v2-Large.

Changes	Inference Time (AVG 40 Images)	
	CPU (s)	GPU (s)
-	1.186	0.007
Total Replacement	1.347	0.008

Table 7.5: Final benchmarks on inference time, CPU and GPU, for complete layer replacement on BiSeNet v2-Large. Inference time were measured as the average of the time needed to perform inference on 40 instances from the training set.

7.6 Conclusion

When taking an attentive look at the experiment benchmarks, it is possible to draw several pros and cons to this method. On the pros side, it makes faster training and requires less space for saving the models. On the cons side there are the loss of precision and slower inference both on CPU and GPU.
In terms of lost precision, the problem could be found on the replacement of convolutions with kernel size of 1x1. In the tables above, standard and depth-wise convolutions envolve a number of parameters. But standard convolutions can refine information better, since they do not segregate multi-dimensional information. In other words, they compute using channel-wise and depth-wise information simultaneous.
Since the number of parameters is similar, the degree of information refinement inside the layer should be the determining factor. As an example, for a kernel size of 1x1, standard convolutions should be chosen, instead of depth-wise separable convolutions.
In terms of inference time, it was latter found that other PyTorch users suffer the same toll when using separable convolutions. Here, two aspects where found. First, at the current time, PyTorch does not have optimizations for this type of convolutions. Second, the amount of IO operations is considerably higher, since an additional feature map must be created for each channel on the input map. These two reasons together add a tremendous amount of time, which consequently impacts the inference time.
A curious aspect can be seen in the improvement of training time *versus* the slower inference time. In this case, the key can be found in the optimizer and the backward propagation. Optimizations in those mechanisms lead to a faster training phase. This improvement is huge, since it compensates the slower feed-forward phase in the network.
When optimizing the network by layer replacement, the considerations above should be taken into account so the end user gets faster and lighter networks, while maintaining similar precision levels.
In the next section, an overview of the thesis work will be presented. Some aspects will be discussed and possible improvements for a future iteration will be detailed.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Deep neural networks encouter several problems when scalling. The creation of larger networks was the *de facto* method for the improvement of precision. While being able to acquire state-of-the-art results, this approach tends to encounter several problems, such as inability to perform on lower-power devices.
Several approaches were created for optimizing deep neural network's footprint. Efficient architectures try to process data in a wiser manner, i.e. by processing different types of information in parallel. Efficient layers perform convolutions differently, either by factorizing the convolution in several steps, or by using bottlenecks to refine information on lower representations. Neural Architecture Search methods can navigate the search space and find architectures that fit under multi-constraints, such as precision-efficiency pair. Pruning and quantization, eliminates a given number of weights and reduce their base representation, respectively. Distillation methods take a network, reduce it, and then perform a training phase where the smaller network learns from the guidance of the original.
Four experiments were performed. First, a preliminary one using the U-Net architecture, which replaced its convolutions. Second, deployment of PyTorch's Quantization framework to reduce weight's representation. Third, eliminating whole filter and training this smaller network using student-professor mechanisms. Finally, an extension of the first experiment, which took a heavier architecture and a larger dataset, then performed a deeper study on layer replacement.
By deploying this list of experiments, several conclusions were found:

- Quantization is able to reduce the footprint of any deep neural network. As the size of the model goes down, the model's redundancy becomes lower and the model is more prone to lose its accuracy;

- On smaller models, quantization mechanisms that merge training and weight calibration are preferred. These methods are able to deliver preciser models, while requiring the same amount of resources for running and storing;

- Pre-training filter reduction or post-training filter pruning is able to reduce the amount of required resources. Its counterpart, weight pruning, does not offer these perks;

- Our experiment with knowledge and stage-by-stage distillation did not offer the desired conclusions, since the second method should deliver models with high accuracy scores. The downside of using this method consists on the higher training time;

- Additionally, the number of stages affect the training time. As the number becomes higher, the training time grows;
- When taking in consideration replacing standard convolutions by depth-wise separable convolutions the size of the kernel is key. For a kernel with size of 1x1, this replacement will only degrade the precision of the network, while maintaing a similar network's footprint. For kernels bigger than 1x1, the required space in disk and memory using on computation become lower and the training time is reduced;
- On the real world, depth-wise separable convolutions did not brought the expected speed up on inference. The reason can be found on the lack of an efficient implementation.

8.2 Future Work

For each of the experimental scenarios studied, there exists a wide array of smaller details that could improve the results, such as a further refinement of the configuration parameters and exploring other architectures.
Also, additional methods to implement efficient networks were found and should be studied like neural architecture search and quantization on GPU. Since the answer to the project's problem could be found in them.
Finally it would be interesting to test the use of more than one of these approaches simultaneously and make evaluations on low-power devices.

Bibliography

[BCCN18] S. Bianco, R. Cadene, L. Celona, and P. Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018. 29

[BKC15] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015. Available from: `http://arxiv.org/abs/1511.00561`. 26

[Cho16] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. Available from: `http://arxiv.org/abs/1610.02357`. 25

[COR+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016. Available from: `http://arxiv.org/abs/1604.01685`. xv, 7, 8

[CZP+18] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018. Available from: `http://arxiv.org/abs/1802.02611`. 26

[GSL+19] Mengya Gao, Yujun Shen, Quanquan Li, Junjie Yan, Liang Wan, Dahua Lin, Chen Change Loy, and Xiaoou Tang. An embarrassingly simple approach for knowledge distillation, 2019. xv, 7, 18

[HMD16] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016. xv, 7, 16

[HSC+19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019. 25, 27

[HSS17] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. Available from: `http://arxiv.org/abs/1709.01507`. 26

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. 7, 16

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. Available from: `http://arxiv.org/abs/1704.04861`. 25

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Available from: `http://arxiv.org/abs/1512.03385`. xv, 21, 22

[IMA+16] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. xv, 7, 9

[LDS90] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990. Available from: `http://papers.nips.cc/paper/250-optimal-brain-damage.pdf`. 7, 15

[LL20] Eugene Lee and Chen-Yi Lee. Neuralscale: Efficient scaling of neurons for resource-constrained deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. xv, 7, 14

[LMSR16] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian D. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. *CoRR*, abs/1611.06612, 2016. Available from: `http://arxiv.org/abs/1611.06612`. 26

[LXFS19] Hanchao Li, Pengfei Xiong, Haoqiang Fan, and Jian Sun. Dfanet: Deep feature aggregation for real-time semantic segmentation, 2019. 7, 26

[MHR19] Sachin Mehta, Hannaneh Hajishirzi, and Mohammad Rastegari. Dicenet: Dimension-wise convolutions for efficient networks. *CoRR*, abs/1906.03516, 2019. Available from: `http://arxiv.org/abs/1906.03516`. xv, 7, 10, 11

[PCKC16] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *CoRR*, abs/1606.02147, 2016. Available from: `http://arxiv.org/abs/1606.02147`. 26

[PGZ+18] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. Available from: `http://arxiv.org/abs/1802.03268`. 7, 12

[PKLC19] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation, 2019. xv, 7, 19

[PPA18] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization, 2018. xv, 7, 19, 20

[RDG$^+$17] Ilija Radosavovic, Piotr Dollár, Ross Girshick, Georgia Gkioxari, and Kaiming He. Data distillation: Towards omni-supervised learning, 2017. xv, 7, 17

[RPB15] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]). Available from: `http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a`. 39

[SHZ$^+$18a] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. Available from: `http://arxiv.org/abs/1801.04381`. xv, 10

[SHZ$^+$18b] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. Available from: `http://arxiv.org/abs/1801.04381`. 7, 25

[TCP$^+$19] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. xv, 7, 11, 12, 13

[TL13] Joseph Tighe and Svetlana Lazebnik. Superparsing: Scalable nonparametric image parsing with superpixels. *International Journal of Computer Vision*, 101(2):329–349, January 2013. 40

[WJQ$^+$17] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. xv, 20, 21

[YGW$^+$20] Changqian Yu, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation, 2020. 2, 7, 24

[YHC$^+$18] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications, 2018. xv, 7, 14, 15

[YWP$^+$18] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation, 2018. 7, 22, 26

[ZSQ$^+$16] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016. Available from: `http://arxiv.org/abs/1612.01105`. 26

[ZZLS17] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. Available from: `http://arxiv.org/abs/1707.01083`. 25