



**Politecnico
di Torino**

Master's Degree in Computer Engineering,
Artificial Intelligence and Data Analytics

Master's Degree Thesis

**Optimization and Quantization on
Hardware Accelerators of
Semantic Segmentation Neural
Networks**

Supervisors:

Ing. Giuseppe AVERTA
Prof. Carlo MASONE
Ing. Francesco PAPARIELLO

Candidate:

Leonardo ROLANDI

DECEMBER 2023

Abstract

Adapting Deep Neural Networks on edge devices, including hardware accelerators as done in this thesis, is in general very challenging due to very specific low-level constraints, like the lack of available memory, the maximum layer dimension allowed and the limited type of layers and high-level architectures actually implemented on hardware.

In the context of the Semantic Segmentation, given that a lot of parameters are needed to well classify every pixel of the image, the problem of the limited amount of memory is particularly emphasized.

But if the transposition to the edge of the network is done properly, the overall process is worthy, because it can offer benefits such as faster performance, decreased power usage, reduced latency and enhanced parallelism. Furthermore, differently from typical cloud paradigms, it can depend much less from data traffic bandwidth limits and can be more reliable to maintain security and privacy.

The initial objective of this study is to select a Semantic Segmentation architecture that is suitable for hardware adaptation, without being too large or complex, yet capable enough to perform its task effectively. After selecting the network, it is trained carefully and analyzed to identify properties that can be leveraged in the subsequent optimization and quantization phase.

The focus of this work is on compressing and adapting in the best possible way the selected architecture to edge devices. Both the retraining approach with quantization awareness and especially the post-training approach are tested, with the latter one involving a guided search using a custom Genetic Algorithm to find the near-optimum quantization configurations.

The results demonstrate that Deep Neural Networks contain redundant information and that, by carefully compressing and optimizing them, their effectiveness is not compromised.

Acknowledgements

I would like to express my sincere gratitude to Dr. Giuseppe Averta, Dr. Niccolò Cavagnero, Prof. Carlo Masone and Prof. Barbara Caputo of research team Vandal, for giving me the invaluable opportunity to undertake this challenging yet exciting journey.

And I desire to express my deepest appreciation to my STMicroelectronic internship referents Ing. Francesco Papariello, Ing. Giuseppe Desoli, Ing. Fabien Castanier, Ing. Marco Grella, and all the other colleagues who followed me and made me find a serene and stimulating working environment. This experience allowed me to test myself, mature, and develop new knowledge and awareness that I will carry with me as priceless luggage throughout my life. For all this, I will always be grateful.

A special mention to Ing. Giovanni Callaris, another ST intern that supported me during almost all of these months and helped me, step by step, to get to the successful completion of this thesis.

To my father and my mother, who always supported and encouraged me in all my choices and have never stopped believing in me.

To my grandparents, who have always been close praying and rejoicing for every single exam I've done in these 5 years.

To my brother, my aunt and my second cousin, who have always been close to me.

To my girlfriend, who encouraged me day by day.

Contents

List of Figures	6
List of Tables	7
Acronyms	8
1 Introduction	9
2 Semantic Segmentation Models	12
2.1 Semantic segmentation	12
2.2 Pre-FCN era	13
2.3 Fully Convolutional Network Discovery	13
2.4 Post-FCN approaches	14
2.4.1 Encoder-Decoder	14
2.4.2 Atrous convolution	15
2.4.3 Spatial pyramid pooling	16
2.4.4 Relevant examples	16
2.4.5 Other Approaches	17
2.5 Challenges to adapt to the edge	17
2.6 The Q-Segs: the final custom models	18
2.6.1 MobileNetV2 backbone	18
2.6.2 Joint multi-scale upsampling	19
2.6.3 Final part	19
2.6.4 Q-Seg1	19
2.6.5 Q-Seg2	20
3 Training And Inference Set-Up	22
3.1 Metrics	22
3.1.1 Mean intersection over union	22
3.1.2 Memory usage	23
3.1.3 Execution time	23

3.2	Datasets	24
3.2.1	Cityscapes	24
3.2.2	Coco-Stuff	25
3.2.3	Data augmentation	25
3.3	Optimizer and scheduler	26
3.4	Batch Normalization Fusion	26
4	Quantization Model Statistics	27
4.1	Target device	27
4.2	Fixed-point notation	27
4.3	Main goal	29
4.4	Qmn stats	29
4.5	Metric stats	30
4.6	Heuristic criterion and retraining	30
5	Genetic Algorithm Search	32
5.1	Post-training quantization	32
5.2	Genetic algorithm	32
5.3	Fitness	34
5.3.1	Sub-metrics	34
5.3.2	Pareto front	35
5.4	Search space dimension	36
5.4.1	Naive approach	37
5.4.2	Search division	37
5.4.3	Weight search space cut	37
5.4.4	Biased probability intervals	38
5.5	Final GA algorithm	39
5.5.1	Implementation	39
5.5.2	Search results	40
5.5.3	Estimated search time	42
6	Results	43
7	Conclusions And Future Works	50
	Bibliography	52

List of Figures

2.1	Semantic Segmentation	12
2.2	Fully Convolutional Network, from slides of Advance Machine Learning course at Polito	14
2.3	Encoder-Decoder SS structure, from https://it.mathworks.com .	15
2.4	Dilated convolution differences, from [5]. From left to right: a normal 3x3 convolution kernel; a 3x3 kernel with dilation rate=2; a 3x3 kernel with dilation rate=4.	15
2.5	Example of a Multiscale Spatial Pyramid Pooling, obtained upsampling and concatenating three initial features of different resolutions	16
2.6	Custom upsampling, obtained with a combination of Concatenate and Reshape layers	19
2.7	Q-Seg1. The NN inputs and outputs are indicated explicitly, the blue polygons enclose a sequence of layers, the grey one stands for the Concatenate Layer, and the coloured rectangles indicates NN features of different depths and resolutions that are used for the upsampling part.	20
2.8	Q-Seg2. The NN inputs and outputs are indicated explicitly, the blue polygons enclose a sequence of layers and the coloured rectangles stands for NN features of different depths and resolutions that are used for the upsampling part.	21
3.1	Intersection over Union (IoU)	22
3.2	Cityscapes dataset	24
3.3	CocoStuff dataset subset	25
4.1	QMN notation, 8 bit case. The violet square stands for the optional sign bit. The green squared indicates the bits reserved for the integer part (m), while the blue ones are the bits for the fractional part (n)	28
5.1	Genetic Algorithm Search	34

5.2	Pareto front graph, from [19]. In this example it is assumed that both metrics $f1$ and $f2$ must be minimized. The circles are all the feasible solutions, while the green ones indicates the non-dominate solutions (the best ones). In this case, the chosen optimal solution is the one with the minimum distance from the intersection of the lines, perpendicular to the axes, drawn from the two most external non-dominated points.	36
5.3	From random to biased intervals	38
5.4	Biased probability intervals (of a single NN parameter). The X axis indicates the number of bits of the quantization, while the Y measures the mIoU obtained quantizing only that parameter in that way. In this example 60 is the mIoU of the not quantized NN. The red lines indicates all the distances from which each average is obtained. Then, confronting the different averages, those who are bigger will have smaller intervals of probability.	39
5.5	Weight search (Q-Seg1, Cityscapes)	41
5.6	Output search (Q-Seg1, Cityscapes) <i>The estimates about the outputs size are an upper bound, as already said in Section 3.1.2, since they do not take into account the possibility that intermediate buffers may be reused</i>	41
5.7	All search (Q-Seg1, Cityscapes)	41
6.1	Cityscapes legend	45
6.2	Cityscapes image	45
6.3	Cityscapes target	45
6.4	Cityscapes Q-Seg1 inference (without quantization)	46
6.5	Cityscapes Q-Seg1 inference (Ga search quantization)	46
6.6	Cityscapes Q-Seg1 inference (Heuristic quantization and retraining)	46
6.7	Cityscapes Q-Seg2 inference (without quantization)	46
6.8	Cityscapes Q-Seg2 inference (Ga search quantization)	47
6.9	Cityscapes Q-Seg2 inference (Heuristic quantization and retraining)	47
6.10	CocoStuff legend	47
6.11	CocoStuff image	48
6.12	CocoStuff target	48
6.13	CocoStuff Q-Seg1 inference (without quantization)	48
6.14	CocoStuff Q-Seg1 inference (Ga search quantization)	48
6.15	CocoStuff Q-Seg1 inference (Heuristic quantization and retraining)	48
6.16	CocoStuff Q-Seg2 inference (without quantization)	49
6.17	CocoStuff Q-Seg2 inference (Ga search quantization)	49
6.18	CocoStuff Q-Seg2 inference (Heuristic quantization and retraining)	49

List of Tables

- 6.1 Cityscapes dataset, comparison between different models/quantizations. 44
- 6.2 CocoStuff dataset, comparison between different models/quantizations. 44

Acronyms

AI	Artificial Intelligence
CNN	Convolutional Neural Network
DCNN	Deep Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
ED	Encoder-Decoder
FPS	Frames Per Second
FCN	Fully Convolutional Network
GA	Genetic Algorithm
HW	Hardware
LR	Learning Rate
ML	Machine Learning
MCU	Microcontroller
NN	Neural Network
NPU	Neural Processing Unit
RNN	Recurrent Neural Network
SS	Semantic Segmentation
SW	Software

Chapter 1

Introduction

Artificial intelligence (AI) , whose first studies started in the 50s of last century, is a rapidly evolving field of computer science that focuses on creating intelligent machines that can perform tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation. Deep learning (DL) is a subfield of AI that focuses on creating neural networks capable of learning and making decisions on their own. It is inspired by the structure and function of the human brain and is designed to mimic the way humans learn and process information. Deep Neural Networks (DNN) are capable of analyzing vast amounts of data, identifying patterns, and making predictions with high accuracy.

In the last decade, these algorithms have rapidly advanced, emerging as promising techniques that outperform previous machine learning methods. However, DL's effectiveness relies heavily on high-performance computing platforms with extensive storage capacities necessary for training these intricate models.

Cloud computing has been the go-to model for running machine learning algorithms, utilizing data centers equipped with substantial processing power and storage capabilities.

Yet, the escalating data traffic and the low-latency demands of many deep learning services are now challenging this centralized computing approach, making it difficult to ensure the necessary quality of service. The considerable bandwidth required by communication networks to handle such vast data volumes presents yet another significant hurdle. Recent research [1] indicates that by 2030, approximately 30 billion Internet of Things (IoT) connected devices will further compound these challenges. Moreover, ensuring security and privacy is crucial when handling user data across a wide range of applications. Finally, in a world dealing with an energy crisis, the escalating energy demands associated with this approach are increasingly problematic.

These challenges have driven to the adoption of the edge computing paradigm,

a decentralized approach that places computing resources, memory, and services in close proximity to where data is generated. This strategy accelerates response times, lessens reliance on communication bandwidth availability and when the targeted hardware is optimized, decrease the power usage and enhance parallelism. Although it's a promising concept, this emerging deep learning service faces a significant challenge: it demands substantial computational power and memory resources. This poses a problem, especially for edge servers and end devices, which typically have much lower computing and memory capabilities compared to large-scale cloud data centers.

In Semantic Segmentation(SS), a popular image vision task that is the objective of this study, the challenge of limited memory becomes prominent due to the extensive parameters required to accurately classify each pixel in the image.

Researchers have explored various avenues to address these constraints:

- Developing new designs and calculators' architectures that integrate edge servers with cloud servers aiming to achieve efficient computational coordination.
- Creating optimized DL models that are mindful of computing, memory, and energy limitations, ensuring more efficient usage of resources.
- Exploring innovative optimizations at the hardware level to enhance both performance and energy efficiency.

More in details, the target platform for this thesis is an hardware accelerated AI microcontroller(MCU) developed by STMicroelectronics. This device is already capable of supporting complex AI algorithms with high power efficiency and performance, however, being an MCU, it's desirable in embedded deployment scenarios, to develop optimized algorithms to further reduce both memory footprint and energy consumption without impacting accuracy significantly, also by leveraging network quantization.

So the core of this work is to create optimized semantic segmentation DL models, that can do well their task but at the same time are suitable for edge devices like STM microcontoller (MCU) with AI hardware(HW) acceleration, and to find optima or nearly optima quantization configurations of every network parameter. The used hardware platform is the one previously mentioned, but this wants to be a generic study that make general considerations, and that can be useful and can be easily extended for adapting DNNs (and not strictly SS ones) to almost all kind of edge devices.

In particular this analysis initially aims to select a suitable Semantic Segmentation architecture to be adapted, not too large in size and without complicated layers, but at the same time good enough to do its task.

Then, given a chosen dataset and an objective, the chosen network is trained and analyzed to find out properties that helps afterwards.

And the core of this study is the subsequent optimization and quantization of the selected architecture, in order to compress and make it suited for the edge devices. Both the retraining approach with quantization awareness, and the post-training approach, are tested. The latter one has its focus on finding the near-optimum configuration doing a guided search to juggle all the many possibilities that the huge search space permits, and a custom Genetic Algorithm (GA) is utilized to perform it.

The results of all this process and optimization, showed in the final results part, are very good compared to the initial software(SW) inferences, proving that DNNs store a lot of redundant information and that by compressing them in a clever way the overall effectiveness does not change, as it's said in the conclusions.

So, within this thesis you can find the following main contributions:

- The search of a custom Semantic Segmentation DNN that performs well even if it's lightweight, and most importantly even if it is made only by simple layers easily implementable at the edge. In particular it is composed by a **MobilNetV2 backbone** and a custom **joint multi-scale upsampling**.
- The implementation of a sequential, effective and complete methodology to deeply analyze a NN from a **fixed-point quantization** perspective. And then searching and finding, with a custom targeted **GA search** which takes into account the previous statistics, some near-optimal quantization configurations, thanks to a multi-objective optimization that considers prediction quality, model size and inference speed.

Chapter 2

Semantic Segmentation Models

2.1 Semantic segmentation

Semantic segmentation is a deep learning technique that assigns a label or category to each individual pixel in an image. So it doesn't distinguish object instances, but it is utilized to identify a set of pixels that represent unique categories (Figure 2.1).

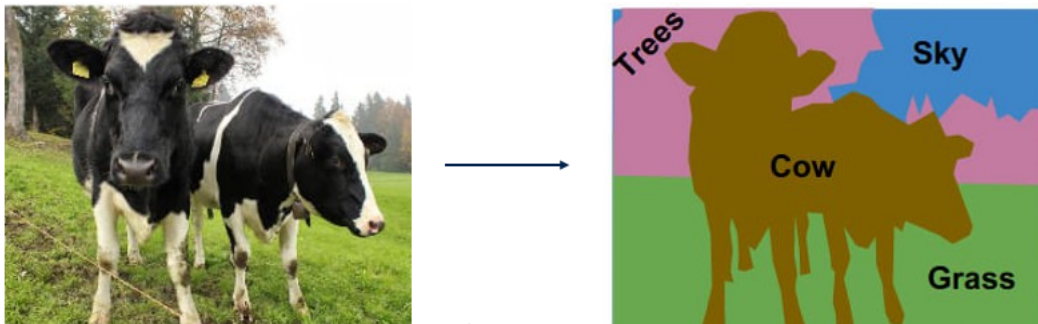


Figure 2.1: Semantic Segmentation

It has many applications, like:

- **Autonomous Vehicles:** Used to identify pedestrians, vehicles, road signs, and obstacles, aiding real-time driving decisions.
- **Augmented Reality:** Separates foreground objects from the background, enhancing realistic interactions in augmented reality applications.
- **Medical Image Analysis:** Assists in identifying and delineating structures and organs in medical images for tasks like tumor detection and organ segmentation.

- **Scene Understanding:** Helps analyze complex scenes in images or videos, essential in applications like video surveillance.
- **Robotics:** Enables robots to identify objects, supporting tasks such as accurate object manipulation and grasping.
- **Image Editing and Special Effects:** Utilized in background removal, special effects and video editing, allowing precise editing in the entertainment industry.

There have been many models in the literature that have implemented it and now there will be an overview of all the most famous ones, dividing them between before and after an event that marked the way to build semantic segmentation NNs, the birth of the Fully Convolutional Networks (FCNs).

All this to arrive, in the end, to a final custom model, chosen after having evaluated pros and cons of every architecture while also thinking about edge hardware constraints.

2.2 Pre-FCN era

In the past there have been various and heterogeneous attempts to face this task, using directly semantic segmentation features or using other data representations, like image segmentation techniques.

Those methods, such as thresholding, clustering and region growing, rely on low-level features like edges and blobs to identify object boundaries in images, making them less effective in scenarios requiring semantic information, especially when similar objects overlap.

Instead methods like Markov Random Fields, Conditional Random Fields, and forest-based techniques were used precisely for SS, utilizing graphical models to understand pixel interdependence and infer scene labels.

Another set of studies, known as Layered models, combined pre-trained and distinct object detectors to extract semantic information.

Then, in the early days of deep convolutional neural networks (DCNNs), attempts were made to adapt classification networks (e.g., AlexNet, VGG) for segmentation, by fine-tuning fully connected layers and then using often a refinement process, since their results were quite unsatisfactory.

2.3 Fully Convolutional Network Discovery

The paper by Shelhamer et al.[2] in 2017 introduced the concept of removing fully connected layers from DCNNs, and to underline this idea, they named the

proposed architecture Fully Convolutional Networks.

The use of FCNs marked a significant development in the field of semantic segmentation, demonstrating that DNNs can be trained for semantic segmentation in an end- to-end manner on variable sized images (Figure 2.2).

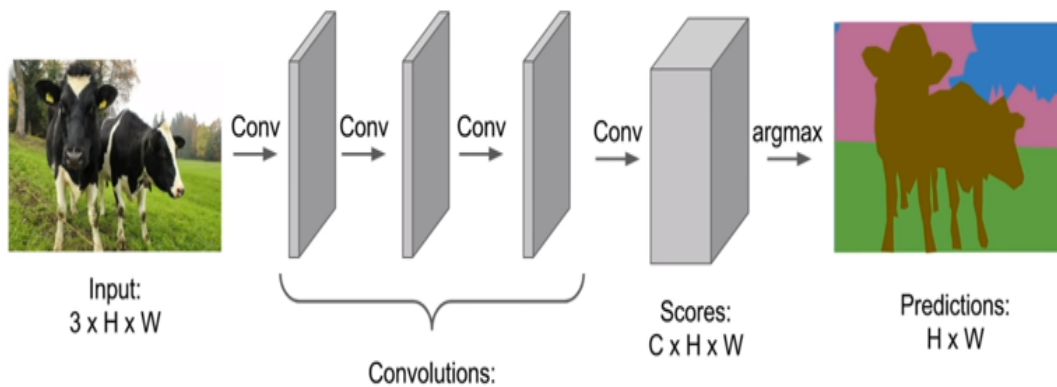


Figure 2.2: Fully Convolutional Network, from slides of Advance Machine Learning course at Polito

The approach of Shelhamer et al. was more clever than simply using a plain FCN, since they used primitive skip connections to prevent eventual loss of localized information due to pooling layer, and Deconvolutional layers that can upsample coarse deep Convolutional layer outputs to dense pixels of any desired resolution. But generally speaking, the FCNs have some drawbacks such as: inefficient loss of label localization within the feature hierarchy, struggle to process global context knowledge, and the lack of a mechanism for multiscale processing.

2.4 Post-FCN approaches

2.4.1 Encoder-Decoder

One way to try to address those problems are the Encoder-Decoder (ED) architectures, also known as U-nets (named after the pioneering study by Ronneberger et al. in 2015[3]) and they consist of two parts: the encoder and the decoder. The encoder reduces the spatial dimension, while the decoder recovers object details and spatial dimension.

In this way with less layers it is possible to have an effective receptive field size pretty big that covers all the input image, and so to maintain a quite good global context knowledge. And obviously also the computation performance is improved, since thanks to the downsampling, the size of the features are a lot smaller than a FCN without stride (Figure 2.3).

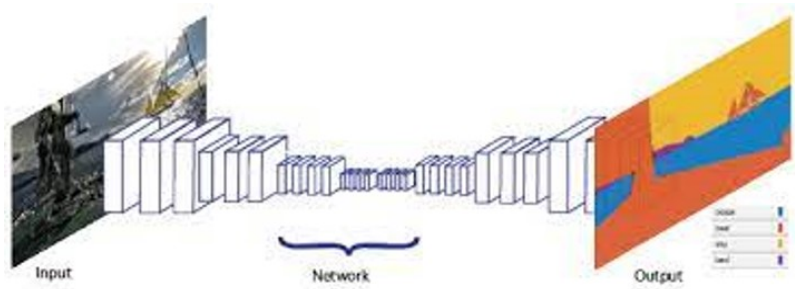


Figure 2.3: Encoder-Decoder SS structure, from <https://it.mathworks.com>

2.4.2 Atrous convolution

Another method to face even more directly this global context issue is employing dilated (atrous) convolutions, a pretty simple yet effective concept. When using standard contiguous convolutional filters, the effective receptive field of units can only increase linearly with layers. However, with dilated convolution, which has gaps in the filter, the effective receptive field can increase much more rapidly (Chen et al. 2018[4]). As a result, a rectangular prism of convolutional layers is created without any pooling or subsampling. Dilated convolution is an effective and powerful method for preserving feature map resolutions in detail. However, compared to other techniques, it requires more GPU storage and computation power since the feature map resolutions do not decrease within the feature hierarchy (Figure 2.4).

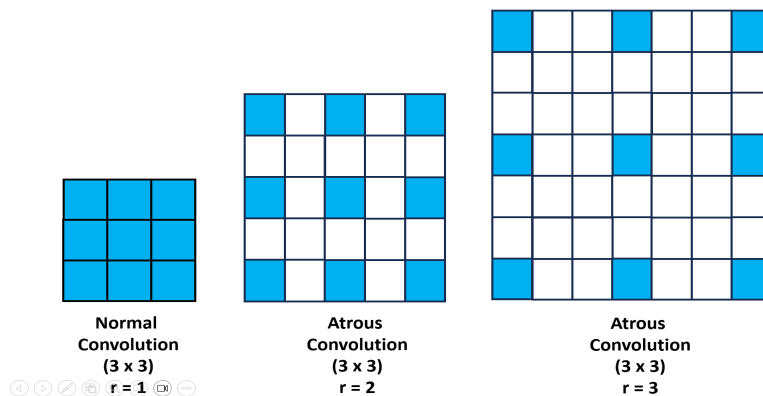


Figure 2.4: Dilated convolution differences, from [5].

From left to right: a normal 3x3 convolution kernel; a 3x3 kernel with dilation rate=2; a 3x3 kernel with dilation rate=4.

2.4.3 Spatial pyramid pooling

An additional evolution of the pure FCN is also using a Spatial Pyramid Pooling (SPP), a pooling layer that eliminates the fixed-size constraint of a CNN, allowing for variable-sized input images.

This is achieved by adding an SPP layer on top of the last Convolutional layer. The SPP layer pools the features in a different way depending on their sizes, and generates fixed-length outputs, which are then passed to the fully-connected layers or other classifiers.

This allows information aggregation at a deeper stage of the network hierarchy, between the Convolutional layers and Fully-connected layers (Figure 2.5).

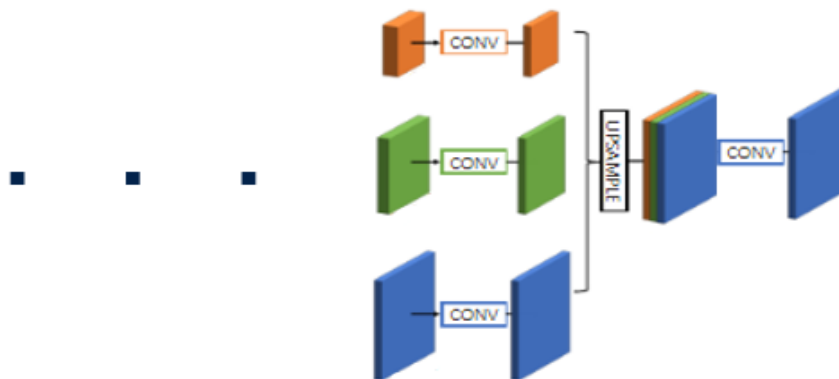


Figure 2.5: Example of a Multiscale Spatial Pyramid Pooling, obtained upsampling and concatenating three initial features of different resolutions

2.4.4 Relevant examples

A good example of most of those previous methods putted together well is the Pyramid Scene Parsing Network (PSPN) of Zhao et al.[6], a multi-scale network designed to improve the learning of global context representation of a scene.

The input image is processed by a residual network (ResNet) as a feature extractor, using a dilated network to extract different patterns.

The feature maps are then passed through a pyramid pooling module to differentiate patterns of varying scales. The feature maps are pooled at four different scales, each corresponding to a pyramid level, and reduced in dimensionality using a 1×1 convolutional layer. The outputs of the pyramid levels are up-sampled and

combined with the initial feature maps to capture both local and global context information.

Finally, a convolutional layer is used to generate pixel-wise predictions.

A more modern example is given by the FastFCN, a fully convolutional neural network architecture designed for semantic segmentation tasks. It was proposed in the paper "FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation" by W. Chen et al[7].

The architecture of FastFCN consists of a backbone network, a feature pyramid network (FPN), and a segmentation head.

The backbone network is based on a ResNet-like architecture with dilated convolutions to increase the receptive field of the network. The FPN is used to combine features from different levels of the backbone network to generate a multi-scale feature map.

Finally, the segmentation head is a series of convolutional layers that produce a pixel-wise classification of the input image.

One of the key innovations of FastFCN is the use of a lightweight FPN that reduces the number of parameters and computation required compared to previous FPN designs. This allows for faster training and inference times while maintaining high accuracy.

2.4.5 Other Approaches

Another research direction comprehends Attention-based Networks: these networks use attention mechanisms to selectively focus on important regions of the input image for segmentation.

Examples of attention-based networks include Attention U-Net[8] and DANet[9]. As far as self-attention is concerned, Transformers have recently been applied to semantic segmentation tasks with promising results. Transformers are a type of neural network architecture that was originally developed for natural language processing tasks, but has been applied to computer vision tasks as well (like Vision Transformer ViT[10]).

Overall, transformers have shown promising results for SS tasks, particularly in cases where it is required to capture long-range dependencies between pixels in the input image.

2.5 Challenges to adapt to the edge

After all this theoretical research, it's time to make some practical considerations. For example the lack of memory resources in edge devices hits particularly the Semantic Segmentation task, because there is the need of a lot of parameters to

classify well every single pixel of an image. So quite complex or quite big structures, like Resnet, or some attention-based ones can't be used as backbone to extract image features, but there is the need to resort to something more feasible and smaller.

For what instead concern Upsampling Layers the situation is not improving, for the lack of implementations at the edge (because more common layers are prioritized) leading to a difficult adaptation. So Atrous convolutions, Deconvolutions, or even quite simple Upsampling layers that use bilinear or nearest neighbours interpolations are unusable, not to mention Transformers.

Another problem for the inference speed is that the ArgMax final function of a Neural Network, which is useful to extract the predicted class, must be done pixel by pixel a lot of times, leading to significant slowdowns of the model.

2.6 The Q-Segs: the final custom models

Based on everything it has been said before, now it's time to create a semantic segmentation model on which to base all subsequent steps.

2.6.1 MobileNetV2 backbone

As backbone, for its smallness but also effectiveness, it's used a MobileNetV2[11], a lightweight and efficient deep learning architecture designed for mobile and edge devices with limited computational resources. It achieves efficiency through several key features, like:

- **depthwise separable convolutions** to reduce computations, splitting convolutions into depthwise and pointwise convolutions.
- **shortcut connections** to allow direct gradient flow, enabling the training of deep networks.
- **clipped version of the Rectified Linear Unit (ReLU)** which ensures that the values do not exceed 6 (ReLU6), preventing large activations and enhancing network robustness.

So at the end the MobilNetV2 is cut before the final Pooling and Classification layers, and it's expanded putting parameter $\alpha=4$ to have some more features, very useful for this task.

2.6.2 Joint multi-scale upsampling

For the upsampling part, it's used a joint multi-scale upsampling, obtained selecting significant features from the backbone, exactly those who precede the reduced resolution steps.

First they are reduced in dimensionality using a 1×1 Convolutional layer. Then they are jointly upsampled, letting everything as simple as possible, to obtain a sort of nearest neighbour interpolation. It is done using a combination of Reshape and Concatenate Layers iteratively, followed by 3×3 Convolutional layers to extract features at each step. In this way multi-scale context information can be extracted from multi-level features, and this leads to better performances (Figure 2.6).

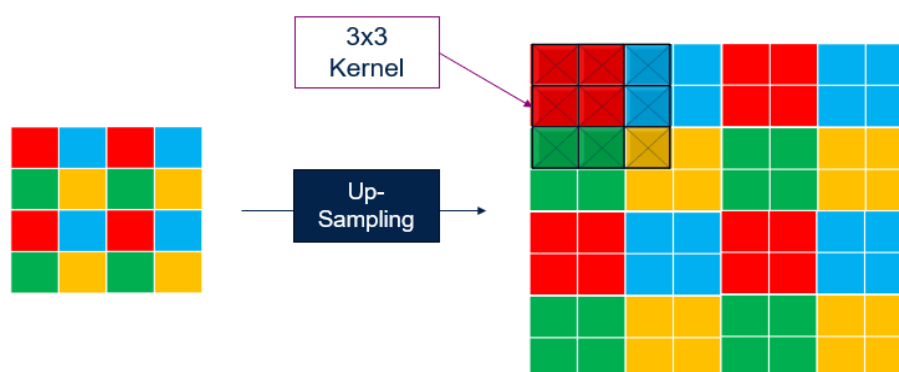


Figure 2.6: Custom upsampling, obtained with a combination of Concatenate and Reshape layers

That can be implemented in slightly different modalities which will be explained later, and in order to test both of them, the final models will be two.

2.6.3 Final part

At the end it's utilized a final Convolution Layer together with an ArgMax to extract pixel-wise predictions, instead of a Fully Connected Layer, always for the reason of saving resources.

2.6.4 Q-Seg1

Q-Seg1 is one of the two final models.

It has the particularity that upsamples all the features independently from each other in the middle steps, and only at the end concatenates them to extract global knowledge.

In particular it has 3922680 total parameters (subdivided in 3877896 trainable and 44784 non-trainable) (Image 2.7)

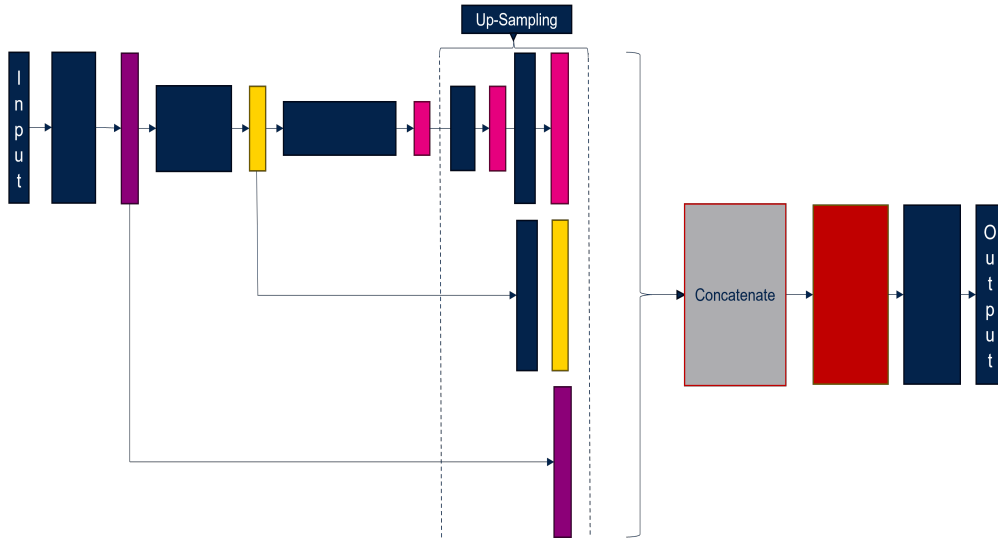


Figure 2.7: Q-Seg1. The NN inputs and outputs are indicated explicitly, the blue polygons enclose a sequence of layers, the grey one stands for the Concatenate Layer, and the coloured rectangles indicates NN features of different depths and resolutions that are used for the upsampling part.

2.6.5 Q-Seg2

The **Q-Seg2** has the particularity that during the upsampling, it progressively compares and concatenates two by two the features, that become of the same size and so that have the same resolution.

Specifically it has 3946680 total parameters (subdivided in 3902328 trainable and 44352 non-trainable) (Image 2.8)

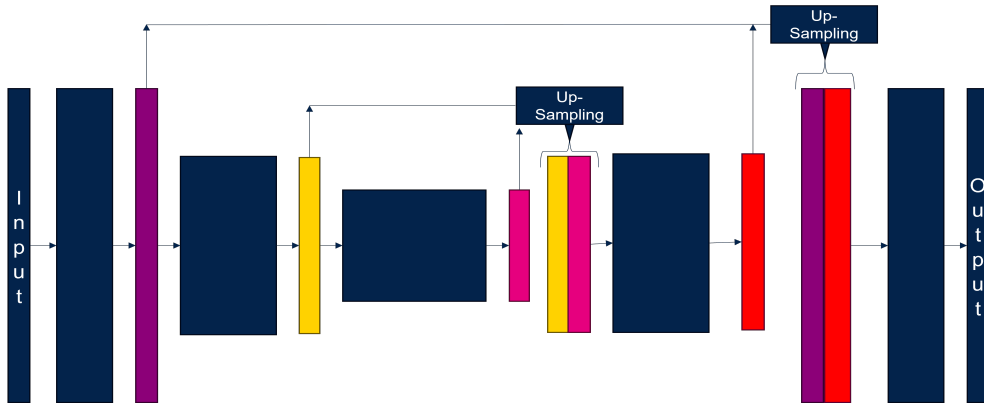


Figure 2.8: Q-Seg2. The NN inputs and outputs are indicated explicitly, the blue polygons enclose a sequence of layers and the coloured rectangles stands for NN features of different depths and resolutions that are used for the upsampling part.

Chapter 3

Training And Inference Set-Up

3.1 Metrics

Before getting into the practical part, here is a rundown of metrics that will be used from here on out.

3.1.1 Mean intersection over union

The Intersection over Union (IoU) is a metric used to evaluate the performance of semantic segmentation algorithms. It is calculated as the area of overlap between the predicted segmentation and the ground truth, divided by the area of union between the predicted segmentation and the ground truth (Figure 3.1).

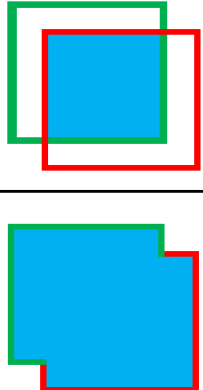
$$IoU = \frac{\textit{area of overlap}}{\textit{area of union}} = \frac{\text{img}}{\text{img}}$$


Figure 3.1: Intersection over Union (IoU)

Then the actual mean IoU of an image is calculated by taking the IoUs of each class and averaging them.

IoU can also be seen as:

$$iou = \frac{true_positives}{(true_positives + false_positives + false_negatives)} \quad (3.1)$$

Those are statistics taken by a confusion matrix, that is a table layout that provides a representation of an algorithm's performance, and each row of the matrix represents instances in an actual class, while each column represents instances in a predicted class.

True Positives (TP) occur when the actual value is positive and the prediction is also positive. True negatives (TN) occur when the actual value is negative and the prediction is also negative. False positives (FP), also known as Type 1 errors, occur when the actual value is negative but the prediction is positive. False negatives (FN), also known as Type 2 errors, occur when the actual value is positive but the prediction is negative.

3.1.2 Memory usage

Computational efficiency is an essential aspect of any algorithm implemented on a real system, and a comparative assessment of the speed and capacity of various semantic segmentation algorithms is a challenging task. The computational load of SS algorithms is evaluated using two primary metrics: computational memory usage and execution time.

Memory usage is crucial when semantic segmentation is used in limited performance devices such as the edge ones. Memory usage for a complex algorithm like semantic segmentation can change significantly during operation, which is why peak memory usage is such a common metric.

In the case of this thesis it is used mainly the static memory to measure a NN, so the weights/biases sizes saved in memory, since it's the most relevant and important aspect for the reference architecture.

But in a case it is utilized also an estimation of the run time memory needed for the model to being executed in the edge. It consists in measuring the output dimension (of every layer) and multiply it for the size assigned for that given output. It is obviously an estimate, an upper bound to be more precise, since it doesn't take into account the possibility that intermediate buffers may be reused.

3.1.3 Execution time

Execution time is measured as the total processing time, from the moment a single image is introduced to the system until the pixel-wise semantic segmentation results are obtained.

The performance of this metric is significantly dependent on the hardware used.

In this case, it will be measured as FPS, that stands for Frames Per Second. It is a measure of how many individual frames or images, a video or an animation displays per second.

3.2 Datasets

The success of any machine learning application is heavily reliant on the quality and quantity of data used for training.

For this reason both Cityscapes and Coco-Stuff datasets will be used to train and to assess the goodness of NN models, but they will be customized a bit to better fit our application scenario and to provide an improved prediction images quality; therefore the quality of the predictions cannot and does not aim to be directly comparable with the state of art of the models.

3.2.1 Cityscapes

Cityscapes[12] is a dataset that focuses on the semantic understanding of urban street scenes, so it is used for outdoor generic tries (Image 3.2). It includes high-resolution images from 50 different cities, captured at different times of the day and in different seasons, with varying backgrounds and scene layouts.

The images with fine annotations, suited for semantic segmentation, are 2975 for training and 500 for validations. Standardly there are 30 different class labels (vehicles, people, riders, etc.), but in this work some of them, those similar to each other, were merged for having a better output representation, so at the end they are only 19.

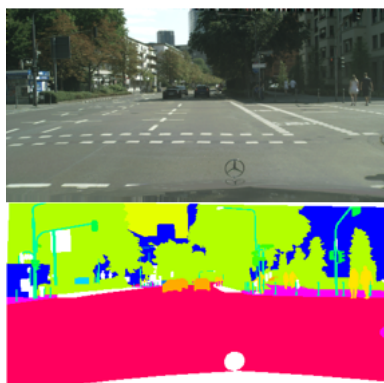


Figure 3.2: Cityscapes dataset

3.2.2 Coco-Stuff

Common Objects in Context (COCO)[13] is a massive image set that includes 200,000 labelled images, 1.5 million object instances, and 80 object categories. It covers almost every possible type of scene and is used for object detection, semantic segmentation, and captioning.

COCO-Stuff[14] is an extension that provides more detailed annotations, including object and stuff (non-rigid, background objects) categories. It includes over 10,000 images from the original COCO dataset, annotated with 80 object categories and 91 stuff categories. This dataset is valuable for training and evaluating algorithms related to semantic segmentation, where the goal is to categorize every pixel in an image into a specific object or stuff category.

Coco-Stuff is a pretty generic dataset, so in this case, for an indoor offices application more suitable for a DEMO, only images that depict offices were selected (Image 3.3). Therefore at the end there are 6350 train images, 280 val images and 20 classes, in which it is included an open set class, since it is thought for a very detailed application and it is not possible to classify individually all the immeasurable types of objects that may be in a room.



Figure 3.3: CocoStuff dataset subset

3.2.3 Data augmentation

In both cases, given that the selected images aren't a lot and to try to have a more robust and generic NN, it is used data augmentation of the following typologies:

- **Horizontal Flip** with $p=0.5$.

- **Color Jitter** that tweaks images' brightness, contrast, saturation and hue with $p=0.5$.
- only in the case of Cityscapes dataset, **Random Crop** of half the input size with $p=0.4$ (since Coco-Stuff annotations/images aren't so targeted and with this type of augmentation there is the risk of letting out all the relevant information).

3.3 Optimizer and scheduler

For the training phase it is utilized an Adam optimizer [15] with initial learning rate(LR) of $5e-4$ and batch size of 4.

As scheduler, instead, it has been created a custom one that every 2 epochs divide by 1.5 the current LR, until it reaches $6e-6$.

And it stops the training either if it reaches 40 epochs, or if there isn't a mIoU improvement for 5 consecutive epochs.

3.4 Batch Normalization Fusion

Before the inference phase of the trained model, there is a process of incorporation of Batch Normalization layers information into the previous Convolutional or Depthwise Convolutional layers.

This is done first to simplify the model structure for the hardware, and then to cut some redundant parameters (the γ and β of the Batch Normalization layers), in order to have to search for fewer quantization configurations. The following formulas explain how, at inference time, the weights and the biases of the Convolutional/Depthwise Convolutional layers are adjusted by means of the Batch Normalization parameters and the Moving Average and the Moving Standard Deviation of the current batch:

$$Conv2d(x) = x * W + bias$$

$$BatchNorm(x) = \gamma * \frac{x - MovingAvg}{MovingStd + \varepsilon} + \beta$$

$$W = W * \frac{\gamma}{MovingStd + \varepsilon}$$

$$bias = \frac{bias - MovingAvg}{MovingStd + \varepsilon} * \gamma + \beta$$

(ε is a hyperparameter with value near 0 that is used to prevent eventual divisions by 0)

Chapter 4

Quantization Model Statistics

4.1 Target device

Now that these considerations have been done, let's dig a little deeper into the target.

The embedded device on which these neural networks need to be adapted, as already said, is a STM microcontoller (MCU) with AI HW acceleration.

It is also composed of a neural processing unit (NPU), which is a specialized type of processor designed to accelerate machine learning and artificial intelligence workloads. NPU devices are designed to perform complex mathematical operations quickly and efficiently, using techniques such as matrix multiplication and convolutional neural networks. They are optimized for low power consumption and high performance, making them ideal for use in mobile devices and other embedded systems.

Like almost every embedded device, they support also a fixed-point notation to store fractional numbers, and this method is utilized in this study for the quantization of the NN model.

4.2 Fixed-point notation

The main difference between floating-point notation and fixed-point notation is that in floating-point notation, the position of the decimal point is not fixed, while in fixed-point notation, the position of the decimal point is fixed.

In floating-point, a number is represented as a combination of a mantissa and an exponent. The mantissa represents the significant digits of the number, while the exponent represents the position of the decimal point. This allows for a wide range of values to be represented with a high degree of precision.

In fixed-point notation, a number is represented with a fixed number of digits

reserved for the fractional part of the number. The position of the decimal point is fixed, which means that the range of values that can be represented is limited by the number of bits used to represent the integer part of the number.

However, in embedded computation, fixed-point notation is more efficient than floating-point notation, as it requires less memory and processing power.

In particular, the used device utilizes a type of fixed-point notation called Qmn notation, adding also the signed version when the number to be quantized requires it. The **Q** in **Qmn** notation stands for "quantization", which refers to the process of converting a continuous number into a digital number by rounding the values to the nearest quantization level. The **m** and **n** values in the Qmn notation represent the number of bits used to represent the integer and fractional parts of the number, respectively, and the total of the two can vary based on the context.

In pure Qmn notation, the most significant bit (MSB) is used to represent the most significant bit of the integer part of the number, and the range of values that can be represented is from 0 to $2^m - 2^{-n}$.

While in Qmn signed notation, the MSB is used to represent the sign of the number, and the range of values that can be represented is from -2^{m-1} to $2^{m-1} - 2^{-n}$.

This means that Qmn notation can only represent non-negative numbers and the total number of bits is m+n, while Qmn signed notation can represent both positive and negative numbers and the total is m+n+1 (Figure 4.1).

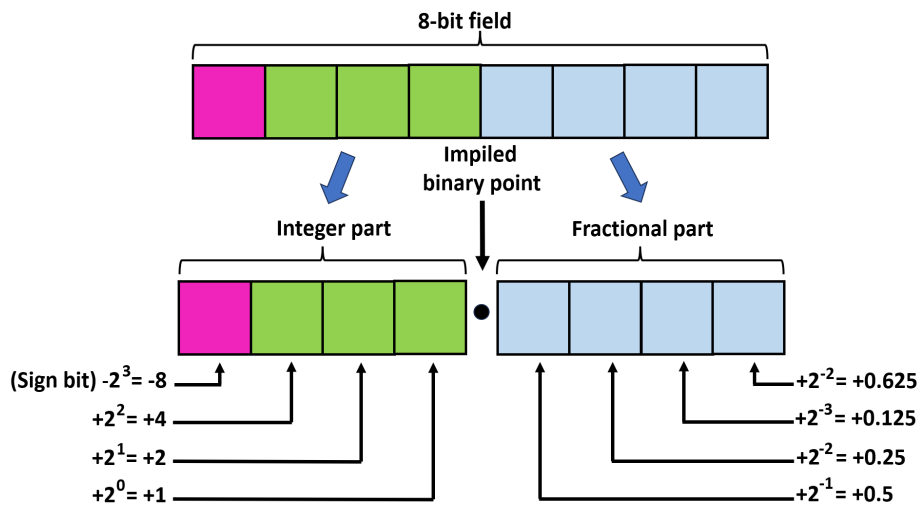


Figure 4.1: QMN notation, 8 bit case. The violet square stands for the optional sign bit. The green squared indicates the bits reserved for the integer part (m), while the blue ones are the bits for the fractional part (n)

4.3 Main goal

The main goal of this study is to quantize both weights and outputs of the chosen NN model in the cleverest possible way, that means to quantize as much as possible without losing too much prediction quality.

In this case, the STM microcontoller (MCU) with AI HW acceleration is pretty flexible about the total bits to choose from for the quantization of every weight, with some exceptions they range from 2 to 16. On the other side concerning the outputs' quantization, they are limited to 8 and 16.

And let's not forget that, even if it's quite safe and with almost irrelevant consequences to assume that almost every tensor also needs the bit for the negative part, from each total number of bits it is necessary to choose what is the right number of bits for the integer part(m) and for the fractional part(n).

In the next section, taking for granted having for each tensor a fixed total number of bits (a value from the bits possibilities for the quantization, from now on called bit-space), there will be a study on how to choose m and n correctly, by means of some statistics.

4.4 Qmn stats

The framework used is Qkeras, given its great quantization customization and the fact that it can simulate in software how a quantized NN would behave.

Using as inputs the chosen dataset and a not quantized model already trained for the target application, for each tensor of each layer different statistics are computed. If the tensor is a weight, getting his values is easy because they are already saved, but if it's an output there is the need of doing a real time inference using the dataset as input to get the dynamic ranges of the values, however the general methodology is anyway the same and it is the following.

For every bit of the bit-space:

- The maximum value of the tensor is recorded.
- If possible, the m parameter of Qmn quantization is set in a way to keep it as little as possible but at the same time to avoid cutting the maximum value. If the maximum value is greater than what the chosen total number of bits can represent, then it will necessarily be clipped.
- Parameter n is derived accordingly.
- Given the chosen quantization, the integer and fractional part errors obtained quantizing are also computed, in order to get an estimate of how much precision is lost.

The first one is simply the count all the clipped values that exceed the range of the possible interval of values with that \mathbf{m} , and then normalizing it by the total number of values.

The second error is calculated elevating by \mathbf{n} every float value to get the approximate correspondent fixed-point notation, and then casting it to integer to find out if there are any differences with the previous value. Then it needs to be normalized by the elevation by \mathbf{n} and the total number of values.

Those are called **Qmn stats** and they are useful to calculate the next statistics, called the **Metric stats**, that are needed to estimate the bits in the bit-space best suited for each tensor.

4.5 Metric stats

By taking as inputs the chosen dataset, a not quantized model already trained for the target application and the **Qmn stats** previously calculated, for each tensor and for every bit of \mathbf{B} is measured the impact that the quantization following the stats has on the whole model.

And it's done quantizing only one tensor at the time and looking at the impact that has on the model metric, the mIoU in this case.

In this way it can be somehow estimated which are the bits in the bit-space best suited for each tensor, and what are the configurations to absolutely avoid, since they cause a significant mIoU drop even quantizing only that tensor in that way.

4.6 Heuristic criterion and retraining

With the **Qmn stats** and the dynamic ranges, given a total number of bits, it's immediate to find the best subpart to represent the integer part (and consequently the fraction part). But how can the total number of bits be decided for every weight/output of the model?

One simple but effective heuristic criterion is using directly the **Metric stats** just found and looking at the drop in the metric score:

For every parameter of the NN, starting from the smallest bits of the bit-space, if the mIoU is below a chosen threshold, then it is chosen a less strict quantization, for example 16 instead of 8 bits.

Otherwise it is used the stricter quantization.

By looking at the metric statistics, it can be noticed that only a few layer lead to a great reduction of the model performance when using an 8 bits quantization.

Simply quantizing in this way and so using only a post-training quantization, even if the previous precautions have been taken, the metric of the whole model will

drop anyway, because the **Metric stats** refer to the quantizations of only single parameters and when it is done for the whole model all the little mIoU drops accumulates their effects.

So it is necessary doing also a Quantization-Aware Training, and in this way the model can get approximately the same mIoU score of the non-quantized version.

Chapter 5

Genetic Algorithm Search

5.1 Post-training quantization

Using a quantization-aware training to boost the quality of the quantized model predictions isn't always a feasible solution. First of all because there is a lack of control, then it is very hard to define in advance which quantizations are better suitable for a retraining and finally it is impossible to predict in advance how good will perform a certain model, it can work very well or it can under-perform. Moreover, if validation data is enough to do an optimum post training quantization, training data and high-performance AI environments with high storage capacities are also needed to perform a quantization-aware retraining and there isn't often the opportunity to do like that.

So this chapter will focus on perform a very good post-training quantization, in particular using a custom specialised Genetic Algorithm (GA) search, and it is the main contribution of this study.

5.2 Genetic algorithm

Metaheuristic algorithms have gained popularity in recent years for solving complex real-life problems in various fields such as economics, engineering, management and politics. These algorithms rely on intensification and diversification as key elements, and striking a proper balance between them is crucial for effective problem-solving.

Most meta-heuristic algorithms get inspired from biological evolution, swarm behavior and physics laws, and they can be broadly classified into two categories: single-solution and population-based.

Single-solution algorithms use a single candidate solution and improve it through local search, but may get stuck in local optima.

Population-based algorithms, on the other hand, use multiple candidate solutions to maintain diversity in the population and avoid getting stuck in local optima. GA, which mimics the Darwinian theory of survival of the fittest in nature, is a popular population-based algorithm among metaheuristic algorithms. It was proposed by J.H. Holland in 1992 [16] and consists of chromosome representation, fitness selection and biological-inspired operators as its basic elements.

So each individual of a population has encoded in itself, more precisely in its genome, the parameters to optimize in this search. Each of the variables is called a gene.

As time passes and new generations are created, the individuals of the population are substituted by others that derive from them (their offspring) that in general are nearer to the optimal solution of the search. More in details, as also shown in Figure 5.1, the steps of a genetic algorithm are as follows:

- **Initialization:** The algorithm starts by creating an initial population of potential solutions to the problem. Each solution is represented as a set of parameters or genes.
- **Selection:** The algorithm selects the fittest individuals from the population to be used as parents for the next generation. The selection process is based on a fitness function that evaluates how well each individual solves the problem.
- **Crossover:** The algorithm combines the genes of the selected parents to create new offspring. This is done by randomly selecting a crossover point and swapping the genes between the parents.
- **Mutation:** The algorithm introduces random changes to the genes of the offspring to create diversity in the population. This is done by randomly flipping or changing the value of a gene.
- **Evaluation:** The fitness of the new offspring is evaluated using the fitness function and only the best are kept.
- **Termination:** The algorithm terminates when a satisfactory solution is found or when a maximum number of generations is reached.

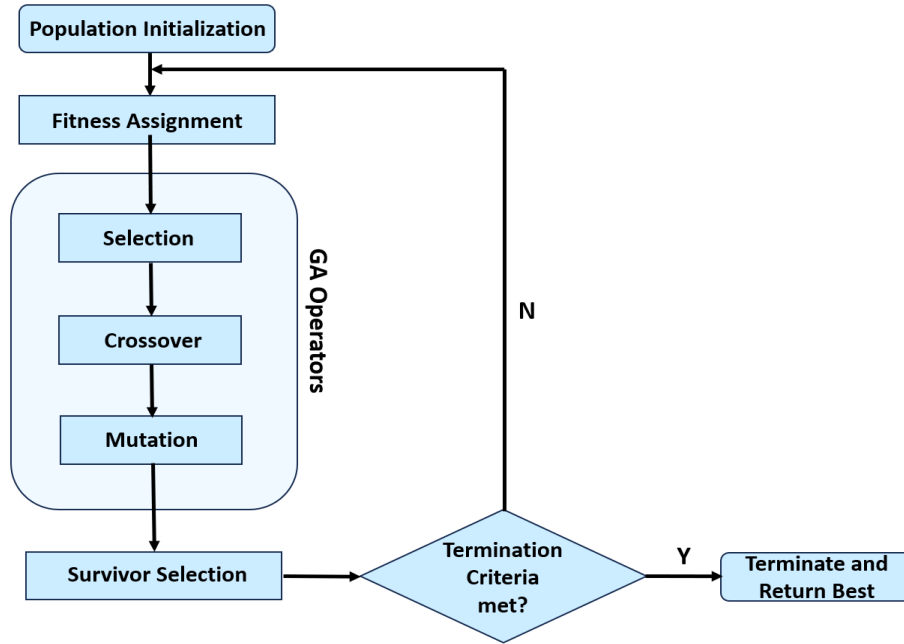


Figure 5.1: Genetic Algorithm Search

5.3 Fitness

Optimization problems often involve multiple objectives and this study is not an exception.

In the past these objectives were typically combined in an ad hoc manner to form a scalar objective function, often through a linear combination of attributes or by converting objectives into constraints.

Multiobjective GA (MOGA)[17] is a modified version of the simple GA that differs in how fitness functions are assigned. Despite this difference, the remaining steps are similar to those of GA. The main goals of Multiobjective GAs are convergence, diversity and coverage.

The genetic algorithm (GA) can be adapted to handle multiple objectives by incorporating the concept of Pareto domination[18] into its selection operator and applying a niching pressure to spread its population out along the Pareto optimal tradeoff surface.

5.3.1 Sub-metrics

As highlighted before, the quantization problem is to find out the best configurations (in terms of the number of bits for each parameter) to adapt NNs at the

edge.

It is important to notice that in this case the parameters are not intended as single numbers (trainable or not) of the NN, but as the type of the tensors of each layer. So for example a classic Convolutional layers has 3 different parameters: the kernel, the bias and the output tensors.

So this optimization can be formulated as:

$$\begin{cases} \max_q M(\phi_q(x)) \\ \min_q |\phi_q| \\ (\max_q \text{fps}(\phi_q(x))) \end{cases} \quad (5.1)$$

The sub-metrics are those explained in Section 3.1, and they are calculated for every individual:

- $M(\phi_q(\mathbf{x}))$ is the metric M computed on the prediction of the model ϕ (quantized following the configuration $q \in Q$) on data X, in this case the mIoU. It is calculated easily doing an inference with the validation dataset.
- $|\phi_q|$ is the size of the quantized model in bits. Depending on which parameter of the individual is considered (a weight/bias or an output), it can be the static size or the esteemed dynamic size.
- $(\text{fps}(\phi_q(\mathbf{x})))$ are the frames per second esteemed compiling the model ϕ for the target device, so it's a very hardware-specific metric. It is put inside curved brackets because it is calculated only optionally, and when it happens, it is used instead of the size metric (even if in the case of comparisons between individuals with very similar fps, the size metric still makes a little contribution). It is done in this way because, even if this metric is very suited and accurate for finding best model's adaptations speed performances, differently from the size it is very slow to calculate.

5.3.2 Pareto front

The Pareto frontier is a concept from economics that refers to the set of optimal outcomes that cannot be improved upon without making at least one other outcome worse off. In other words, it's the set of solutions that are considered "efficient" or "non-dominated" in a multi-objective optimization problem.

In particular, in a 2D Pareto front, the x-axis represents one objective and the y-axis represents another objective. Each point on the graph represents a solution to the optimization problem and the Pareto frontier is the set of points that are

not dominated by any other point. This means that there is no other solution that is better in both objectives than any point on the Pareto frontier. By identifying the set of non-dominated solutions, it can then be chosen the solution that best fits some priorities and trade-offs (Figure 5.2).

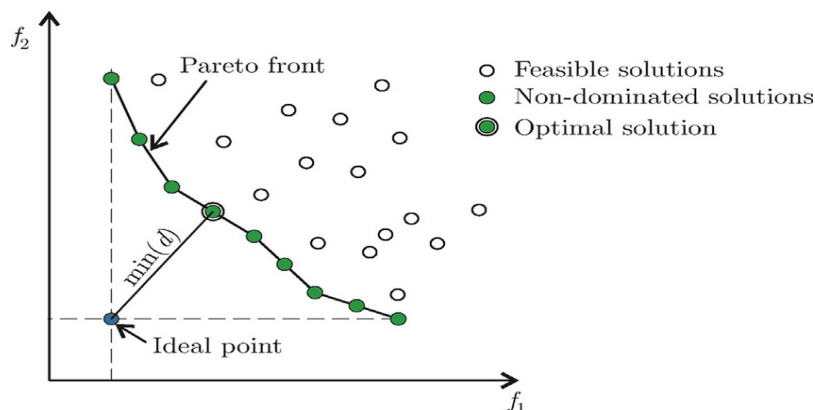


Figure 5.2: Pareto front graph, from [19].

In this example it is assumed that both metrics f_1 and f_2 must be minimized. The circles are all the feasible solutions, while the green ones indicates the non-dominated solutions (the best ones). In this case, the chosen optimal solution is the one with the minimum distance from the intersection of the lines, perpendicular to the axes, drawn from the two most external non-dominated points.

In this application, the optimization problem has those two objectives: minimizing size and maximizing the mIoU.

So a set of solutions(individuals) that trade off between these two objectives are generated and plotted on a 2D graph. The red dotted line corresponds to the mIoU of the not-quantized model, in order to have a point of reference.

The main Pareto frontier is the set of solutions that are both the lowest size and the highest mIoU, and any other solution is dominated by at least one point of them. This can be done iteratively to find out on which level of Pareto frontier assign each point, and that's the real fitness value for each individual.

So the GA search favors the solutions that are part of the most "external" frontiers, which have a smaller Pareto number, so it is actually a minimization of a single objective, the Pareto front.

5.4 Search space dimension

Now it is time to make some considerations about the search space dimension of the GA applied to this context.

Let's focus on the scenario of only the following number of bits permitted for

the quantization of every parameter: 4 – 8 – 12 – 16. In this case the bit-space dimension (B) is equal to 4.

Even with the Batch Normalization layers already removed (as explained on Section 3.4) the average number of parameters of a performing Semantic Segmentation model small enough to fit in the edge are many anyway. In particular the number of model weights(+ biases)(W) are more or less 135. The number of model outputs(O) are instead more or less 175.

5.4.1 Naive approach

With these premises, the total search space dimension would be the following: $B^{W+O} = 4^{310} \sim 4.35 * 10^{186}$ Obviously this is a naive approach, infeasible even for the GA search, with almost no hopes to find sub-optimal solutions.

5.4.2 Search division

A way to reduce this problem is dividing the search into 3 sub-tasks, all independently configurable:

- **Weight search** (4-8-12-16 bit-space): $B_W = 4$
- **Output search** (8-16 bit space): $B_O = 2$
- Put the best results of the previous steps together (**All search**, not properly a GA)

In this way, if necessary, different bit-spaces can be used for different sub-tasks and the process can be optimized in terms of search time and in terms of search space dimension, losing the complete independence of all the variables (a reasonable prize to pay). Doing that, it is also kept in consideration(as explained in Section 4.3) that in the target device the quantization of the outputs is stricter than the weights one.

So the total search space dimension is reduced like that: $B_W^W + B_O^O = 4^{135} + 2^{175} \sim 1.9 * 10^{81}$.

But sadly the weight search space dimension is still too big to be handled from the algorithm, so it is necessary to make other changes.

5.4.3 Weight search space cut

The weight search space can be further cut by removing the search of weights whose sizes are very small, because they are almost irrelevant in the optimization process and they don't give an effective benefit in terms of speed/memory occupation.

To do so, given that the size of a weight i is $|w_i|$, the mean of the sizes of all weights also needs to be calculated. And the weights to keep in the search are those whose $\frac{|w_i|}{\text{mean}(|w|)} > \text{threshold}$ (by default 0.01), and the others can be put to 16 bits automatically.

In this way about 35 weights can be cut and the weight search space dimension goes from 4^{135} to 4^{100} , that corresponds more or less to $1.61 * 10^{60}$; unfortunately once again there are too many possibilities to handle.

5.4.4 Biased probability intervals

Since cutting the search space isn't enough, there is the need to use complementarily another approach, that is pushing the search in a reasonably right direction. Usually when creating the initial population, the genes are chosen randomly from the possibilities (the bit-space) and also when it is needed to change a gene in the mutation operation, the choice is still random.

So the new idea is having biased choices and biased probability intervals for this operations (Figure 5.3), and for doing that, as it occurred in the heuristic criterion with the training-aware approach, the previously calculated **Metric stats** are utilized (Section 4.5).

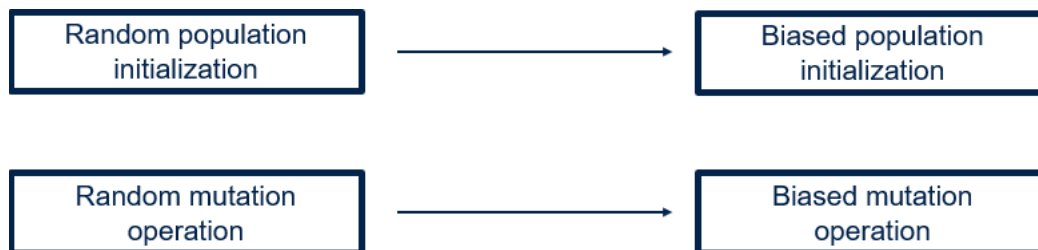


Figure 5.3: From random to biased intervals

For every parameter, intervals of probability for the biased choices can be extracted reversing and normalizing some special means of every bit of the bit space.

The mean is calculated summing all the distances from the mIoU found quantizing with that specific bit, to all the mIoU of the bigger bits (and also the mIoU of the not quantized model), and then dividing by the number of the distances.

In this way, mIoU of every single bit are evaluated relatively to those of other bits, as it is shown in Figure 5.4.

The main idea of this approach is that if already quantizing in a certain way just a single parameter the mIoU is quite low, then if I quantize also the entire model is very likely that the mIoU would drop significantly. And so it is chosen a very low probability for that quantization for that parameter.

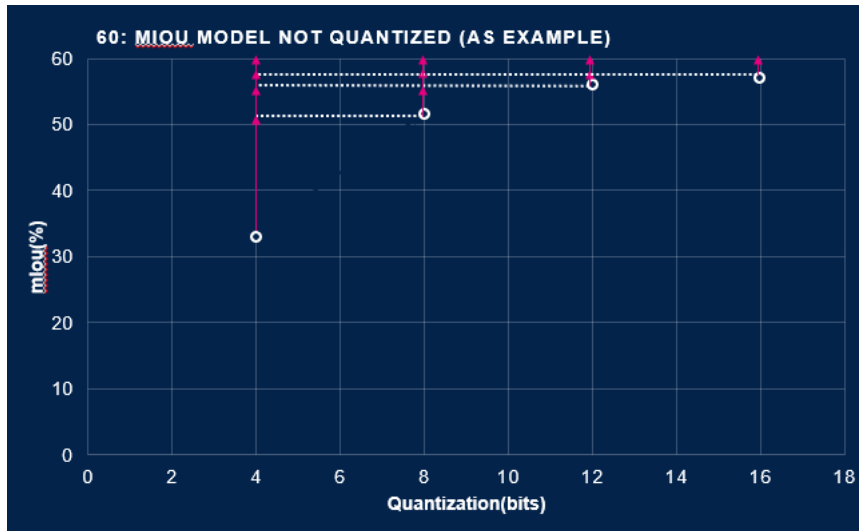


Figure 5.4: Biased probability intervals (of a single NN parameter). The X axis indicates the number of bits of the quantization, while the Y measures the mIoU obtained quantizing only that parameter in that way. In this example 60 is the mIoU of the not quantized NN. The red lines indicates all the distances from which each average is obtained. Then, confronting the different averages, those who are bigger will have smaller intervals of probability.

5.5 Final GA algorithm

5.5.1 Implementation

By putting all these components and concepts together, the custom GA algorithm is created.

At the beginning, 30 biased initial individuals are generated. Then they are evaluated with a fitness function to understand how good they are in respect to mIoU, size and fps, and the Pareto front put together all those informations.

And then, for each generation, 45 new offspring are generated starting from the current population, 30% of times with a mutation operation and the remaining 70% with a crossover operation.

For choosing the "parent" individuals, it is used a K-Way tournament selection. K=8 individuals are selected from the population at random and the best out of these is picked to become a parent.

The crossover operation is a simple one-point crossover, in which a random point is selected and the genes before are selected from a parent and those after from the other one, creating a new individual.

The mutation, as previous explained, is a biased resetting, so a biased value from the

set of permissible values is assigned to a randomly chosen gene. But this operation isn't done one single time for the creation of the same individual. It all depends from the amount of parameters that the search must optimize (so the number of genes). The thresholds set are every 150 parameters and when one of these is exceeded, the number of genes that can be modified is increased by one. However only the possibilities increase, to really decide how much genes are modified there is a random choice starting from one up to the number of possibilities.

So when all the offspring are generated, there is the need to choose some of them to not make grow population indefinitely. Elitism is the chosen approach, so a fitness based selection is used to keep only the best ones (obviously first removing any duplicates). But there isn't the usual fixed population threshold (that is only a lower bound), and as long as the individuals have Pareto front equal to 1, they are kept in the population. This, combined with Pareto front fitness, almost make up for the fact that there isn't a specific methodology to enhance diversity.

As termination criterion, for reasons of times, the algorithm stops when a certain amount of generations is reached, but it can easily be extended to halt at convergence.

5.5.2 Search results

The Figures 5.5, 5.6, 5.7 shows respectively the results of **Weight search**, **Output search**, and **All search** (done for the **Q-Seg1** on Cityscapes dataset as an example).

As it can be seen, by doing Post-Training Quantization, the model usually get approximately the same score of the non-quantized version, reducing at the same time decidedly the size, so the final results are satisfactory.

Since it is a Pareto front implementation, one of the final configurations can be chosen as needed, picking the suited trade-off between mIoU, size and fps.

In those cases, the model not quantized mIoU is $\sim 57.98\%$ (red dotted line).

For the first **Weight search**, since the bit space dimension is pretty high (4), 100 generations are used, and a Pareto fitness of mIoU and model size is adopted.

For the **Output search** with a B of 2, they are needed only 30 generations, and the metrics are the same of the previous one.

And at the end, for the **All search** that puts together the previous results, given that is the final search and it is used to actually decide the quantization configuration, there is the need for more accuracy and so the mIoU and the fps are used as metrics, even if in this way the process is slower.

It's worth noticing that the range of the mIoU in the graph is the usual 0-100 (%), while that of the size and the fps changes model by model, because it is normalized according to the minimum and maximum value the metric could get if the model is quantized in the smallest and in the biggest way possible.

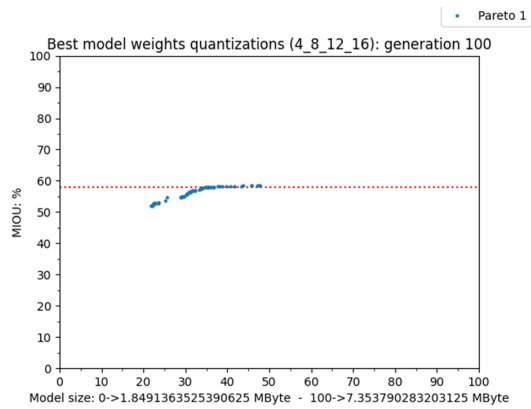


Figure 5.5: Weight search (Q-Seg1, Cityscapes)

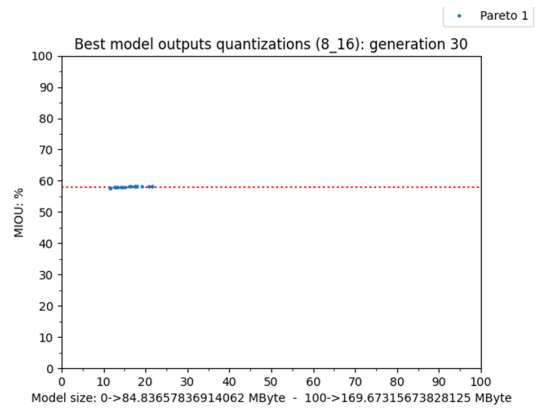


Figure 5.6: Output search (Q-Seg1, Cityscapes)

The estimates about the outputs size are an upper bound, as already said in Section 3.1.2, since they do not take into account the possibility that intermediate buffers may be reused

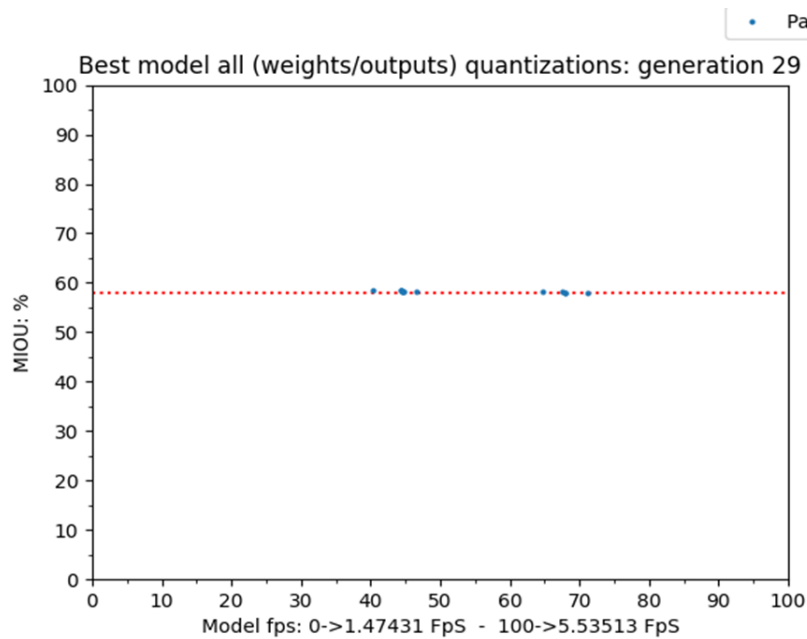


Figure 5.7: All search (Q-Seg1, Cityscapes)

5.5.3 Estimated search time

Obviously the following calculations are a not precise esteem of the duration of this post-training quantization, since they depend from the type of calculator it is being used and the workload it currently has.

In particular, the supercomputer used for this algorithm's execution has 2 Intel Xeon Gold 6240R CPU with a total of 48 cores, 96 threads', 2 Quadro RTX 8000 GPUs with 48 MB of memory each, and 1 TB of RAM. It is worth noticing that the resources of this calculator are often divided between different tenants.

Having said that, it's quite useful having some reference numbers.

The preprocessing steps (executed once per model) are these:

- **Qmn stats** to quantize well (dynamic ranges...): ~ 37 minutes
- **Metric stats** quantizing one parameter at a time: ~ 37 minutes * 4 (bit space size) : ~ 148 minutes

GA initializations:

- dataset caching for fast inference (1 minute) + (3 initializations (weights, outputs, all) * ~ 3 minutes): ~ 10 minutes

GA searches:

- **Weight search** with mIoU and size metrics: 30 individuals (default number of initial population) * ~ 1.6 s + 100 (default number of generations) * 45 (default number of offspring) * ~ 1.6 s: ~ 120 minutes
- **Output search** with mIoU and size metrics: 30 individuals (default number of initial population) * ~ 18 s + 30 (default number of generations) * 45 (default number of offspring) * ~ 18 s: ~ 414 minutes
- **All search** with mIoU and fps metrics: ~ 35 (best weight individuals) * ~ 18 (best output individuals) * ~ 39 s: ~ 409.5 minutes

So the total amount of search time is ~ 1138.5 minutes = ~ 19 hours

Chapter 6

Results

Results demonstrate the effectiveness of both the retraining approach with the heuristic criterion, and the post-training quantization with the GA search. With the latter one there is more control about the particularities of the chosen quantization configurations, both because they are optimized based on mIoU, size and fps, and because there is the possibility of choice of the best suited configuration for the application (thanks to Pareto front fitness). Moreover there isn't the need to use training data.

But in general the results are slightly better (on both datasets) if a retraining approach is used, both for mIoU and the model size.

But it is worth noticing that a perfect tuning of the GA hyperparameters was not done (for example diversifying them whenever a dataset or a specific model is changed), but an attempt was made to remain fairly general. Furthermore, GA search was stopped at a certain number of generations, giving up continuing the research, and the more accurate fps metric was only used on the final all search. Having said that, both the quantization approaches demonstrate to obtain more or less the same prediction accuracy as the not-quantized model, but with the model size reduce a lot. In effect, the results oscillate from a size of the quantized model equal to 23.96% compared to the original one, going up to 26.32%. So, regarding the average size of each parameter, there has been a transition from a 32-bit floating-point to approximately an 8-bit fixed-point.

For what concerns instead the different Segmentation models created at the beginning, they both perform well. However, it can be seen that **Q-Seg1** provides slightly better inferences, given that the images outlines are smoother and less segmented, differently from **Q-Seg2**.

Table 6.1 shows the results for Cityscapes dataset, and Table 6.2 for CocoStuff. "N" is an abbreviation for "Normal" and refers to the not quantized model, instead "Q" stands for "Quantized" and indicates the model already adapted for the edge.

Model name	Q. type	N. mIoU	Q. mIoU	MIoU ratio (Q/N)	N. size (MB)	Q. size (MB)	Size ratio (Q/N)	Fps
Q-Seg1	GA	57.98	57.82	99.72%	15.42	3.87	26.32%	4.36
Q-Seg1	Heuristic	57.98	59.95	103.4%	15.42	3.71	26.08%	5.44
Q-Seg2	GA	58.11	57.76	99.4%	15.52	4.14	26.7%	4.7
Q-Seg2	Heuristic	58.11	57.89	99.62%	15.52	3.86	24.88%	6.89

Table 6.1: Cityscapes dataset, comparison between different models/quantizations.

Model name	Q. type	N. mIoU	Q. mIoU	MIoU ratio (Q/N)	N. size (MB)	Q. size (MB)	Size ratio (Q/N)	Fps
Q-Seg1	GA	36.89	37.01	100.33%	15.42	4.03	26.11%	6.92
Q-Seg1	Heuristic	36.89	38.8	105.17%	15.42	3.8	24.61%	11.21
Q-Seg2	GA	36.42	36.21	99.42%	15.52	3.73	25.21%	6.87
Q-Seg2	Heuristic	36.42	37.08	101.81%	15.52	3.72	23.96%	12.66

Table 6.2: CocoStuff dataset, comparison between different models/quantizations.

The following images instead give some examples about how actual inferences looks like (for the different methodologies/datasets/models utilized).

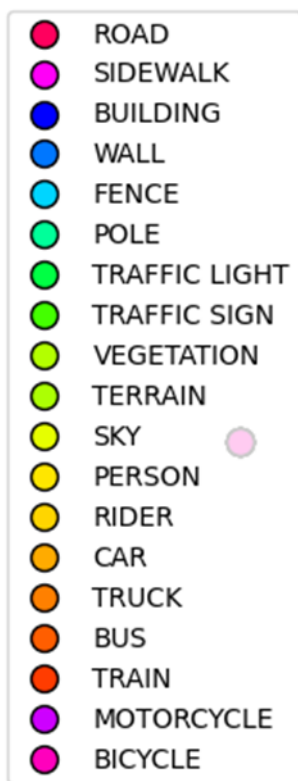


Figure 6.1: Cityscapes legend



Figure 6.2: Cityscapes image

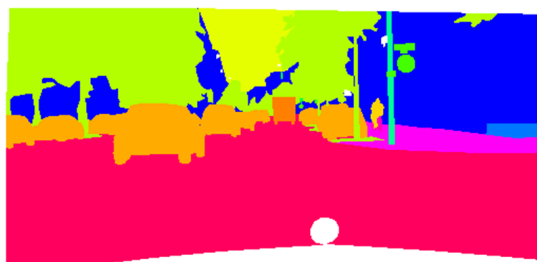


Figure 6.3: Cityscapes target

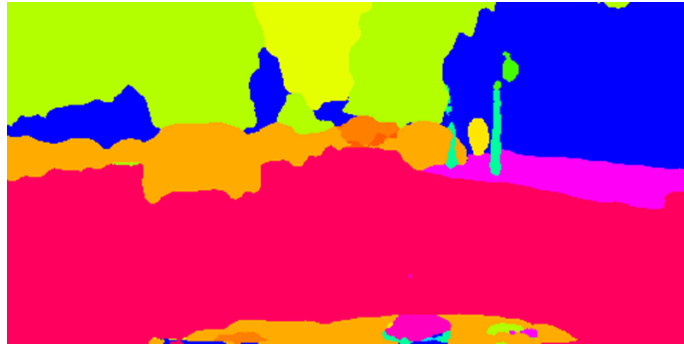


Figure 6.4: Cityscapes Q-Seg1 inference (without quantization)

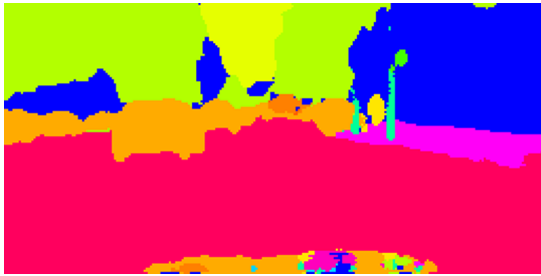


Figure 6.5: Cityscapes Q-Seg1 inference (Ga search quantization)



Figure 6.6: Cityscapes Q-Seg1 inference (Heuristic quantization and retraining)

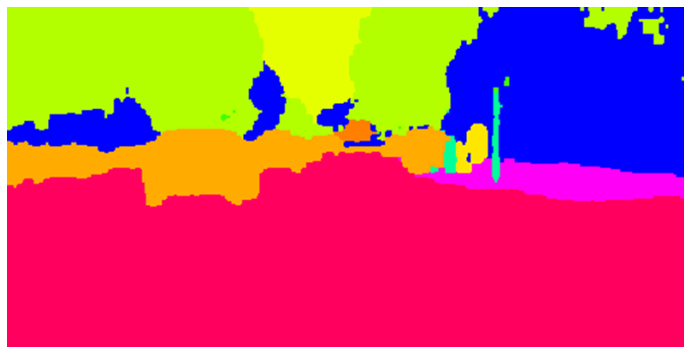


Figure 6.7: Cityscapes Q-Seg2 inference (without quantization)

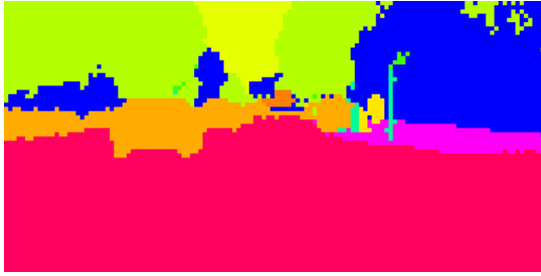


Figure 6.8: Cityscapes Q-Seg2 inference (Ga search quantization)

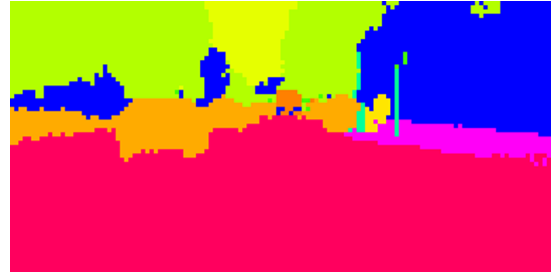


Figure 6.9: Cityscapes Q-Seg2 inference (Heuristic quantization and retraining)



Figure 6.10: CocoStuff legend



Figure 6.11: CocoStuff image



Figure 6.12: CocoStuff target

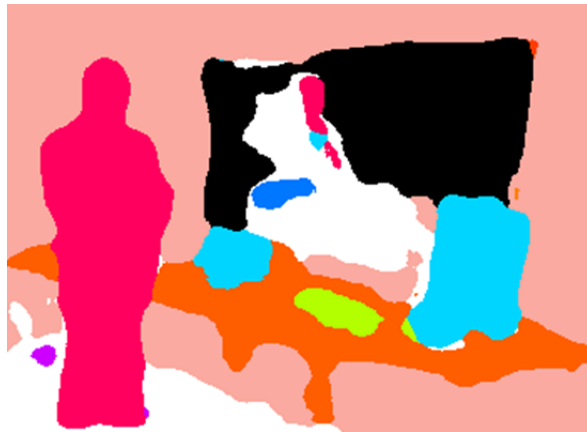


Figure 6.13: CocoStuff Q-Seg1 inference (without quantization)



Figure 6.14: CocoStuff Q-Seg1 inference (Ga search quantization)

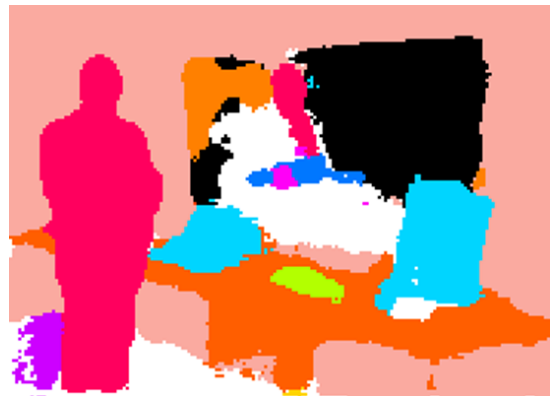


Figure 6.15: CocoStuff Q-Seg1 inference (Heuristic quantization and retraining)



Figure 6.16: CocoStuff Q-Seg2 inference (without quantization)



Figure 6.17: CocoStuff Q-Seg2 inference (Ga search quantization)



Figure 6.18: CocoStuff Q-Seg2 inference (Heuristic quantization and retraining)

Chapter 7

Conclusions And Future Works

In conclusion, this thesis demonstrate that it's possible to achieve good results in the Semantic Segmentation task using Neural Networks with not many parameters and constituted by basic layers.

Furthermore the process of optimization and quantization has proven to be effective for edge devices adaptation, reducing considerably the size of the model while maintaining the overall effectiveness of the neural network.

The just explained process has been done analyzing the trained model in the right way and doing targeted and clever configuration search.

The results obtained from this process have shown that neural networks store a significant amount of redundant information, which can be compressed in a clever way without affecting the overall effectiveness.

This finding has important implications for the field of artificial intelligence and machine learning, as it suggests that there is significant potential for improving the efficiency and performance of neural networks through optimization techniques. Further research in this area could lead to the development of more advanced and effective neural network models.

This could be done also trying to quantize even more and testing binary nets[20] on the Semantic Segmentation task.

Another possible branch of research is attempting to use other types of quantization different from fixed-point QMN, with a wide range of applications in various fields.

Bibliography

- [1] Statista. *Number of Internet of Things (IoT) Connected Devices Worldwide in 2018, 2025 and 2030*. [URL](#), last accessed on 2023-10-25. 2020.
- [2] Shelhamer, E., J. Long, and T. Darrell. “Fully convolutional networks for semantic segmentation.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39(4) (2017), pp. 640–51.
- [3] Ronneberger, O., P. Fischer, and T. Brox. “U-net: Convolutional networks for biomedical image segmentation.” In: *In Medical Image Computing and Computer-Assisted Intervention* (2015), pp. 234–41.
- [4] Chen, L., G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution and fully connected crfs.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40(4) (2018), pp. 834–48.
- [5] Almustafa Abed, Belhassen Akrouf, and Ikram Amous. “Semantic Heads Segmentation and Counting in Crowded Retail Environment with Convolutional Neural Networks Using Top View Depth Images”. In: *SN Comput. Sci.* 4.1 (2022).
- [6] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. “Pyramid Scene Parsing Network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).
- [7] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yizhou Yu. “Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation”. In: *arXiv preprint arXiv:1903.11816* (2019).
- [8] Ozan Oktay et al. “Attention U-Net: Learning Where to Look for the Pancreas”. In: (2018).
- [9] J. Fu et al. “Dual Attention Network for Scene Segmentation”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 3141–49.

- [10] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations* (2021).
- [11] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018), pp. 4510–20.
- [12] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).
- [13] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* (2014).
- [14] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. “COCO-Stuff: Thing and stuff classes in context”. In: *Computer vision and pattern recognition (CVPR), 2018 IEEE conference on* (2018).
- [15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014).
- [16] John H. Holland. “Genetic Algorithms”. In: *Scientific American* 267.1 (1992), pp. 66–73.
- [17] Carlos M. Fonseca and Peter John Fleming. “Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization”. In: *International Conference on Genetic Algorithms* (1993).
- [18] J. Horn, N. Nafpliotis, and D.E. Goldberg. “A niched Pareto genetic algorithm for multiobjective optimization”. In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence* 1 (1994), pp. 82–87.
- [19] Facundo Bre and Víctor Fachinotti. “A computational multi-objective optimization method to improve energy efficiency and thermal comfort in dwellings”. In: *Energy and Buildings* 154 (2017).
- [20] Chunyu Yuan and Sos S. Agaian. “A comprehensive review of Binary Neural Network”. In: *Artificial Intelligence Review* (2021), pp. 1–65.