Latency Driven Spatially Sparse Optimization for Multi-Branch CNNs for Semantic Segmentation

Georgios Zampokas[1,2] Christos-Savvas Bouganis[1] Dimitrios Tzovaras[2]

[1] Imperial College London, London, UK

[2]Information Technologies Institute, Centre for Research and Technology Hellas, Greece

Abstract

Semantic segmentation has gained significant attention in the field of computer vision, especially in the context of autonomous driving. Achieving superior performance and precise object localization is paramount for safe and reliable autonomous vehicles. This introduces the need to process high-resolution feature maps, resulting in increased computational requirements. Recently proposed multi-branch architectures address that by maintaining parallel computationally light high-resolution representations throughout the whole network. Since individual branches focus on different image regions by design, we believe that there is significant number of redundant computations, especially in high-resolution branches. To harness that, we propose a optimization scheme for multi-branch CNNs, which introduces spatial sparsity to the network to produce more efficient distribution of calculations. The proposed approach departs from the literature by introducing the actual latency in the optimization process, resulting in device-tailored and practically-efficient sparse architectures.

1. Introduction

The ability to generate a detailed understanding of the contents of an image vital role in a wide range of applications, such as autonomous driving, robotics, and medical imaging. Therefore, semantic segmentation is a fundamental computer vision task, with a significant research effort invested in it. Modern sensors have the ability to capture images at high resolutions, resulting in higher pixel count which increases the computational budget of convolutions. Such semantic understanding algorithms are usually targeted at low power devices such as embedded GPUs or mobile phones, which often do not posses the available resources to accommodate them.

High resolution is crucial for achieving accurate and detailed semantic segmentation results, as it enables the algorithm to capture fine-grained details and distinguish between objects with similar appearances, such as cells in medical imaging or pedestrians in crowded scenes. The main challenge lies in combining multi-scale and global cues which capture the image context, with high resolution details that preserve local structure. Researchers have tackled this problem by either by using an encoder-decoder like structure which extracts representations and upsamples them to high resolution or by introducing multiple paths of computation at different scales, to achieve information flow at different resolutions.

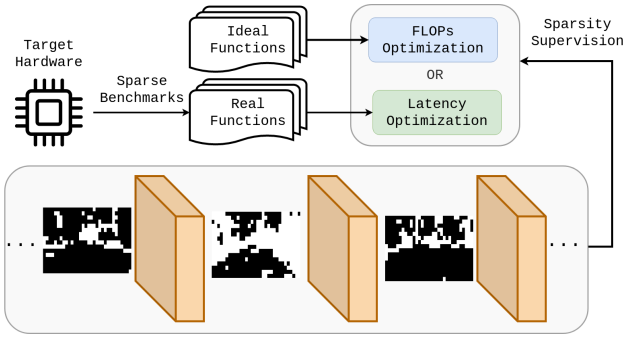

Figure 1. Figure presenting the overall overview of our method. The optimization framework can either optimize for FLOPs or latency, by using benchmark results for every layer under target hardware and sparse implementation.

However, in a typical CNN execution, computations remain unaware of the information and complexity within the image. To this end, a set of dynamic methods emerged, which decide a more efficient distribution of computations during inference, by enforcing channel and/or spatial sparsity. Most dynamic methods focus on the reduction in computational complexity relying on theoretical proxies such as floating-point-operations (FLOPs), which often fail to capture the practical impact of optimization. Besides that, several dynamic methods achieve practical speedup of execution time [18], [19], [21], [24] but it mostly comes as a byproduct rather than being included explicitly in the opti-

mization process.

Therefore, we propose a framework which has the ability to consider either the theoretical complexity or the actual latency of the execution, to optimize a CNN for efficiency, introducing spatial sparsity to the network. Our main contributions can be summarized as:

- An extension to an existing optimization framework which employs spatial sparsity to improve efficiency by introducing a projection of the latency when running on a device, to the optimization process.
- A latency projection methodology based on practical execution of CNN workloads on target devices.
- We demonstrate our method on state-of-the-art multi-branch CNNs for semantic segmentation, and show that it can reduce both FLOPs and latency with small reductions in mIOU%, outperforming relevant approaches.

2. Related Work

The current work is placed between the field of efficient semantic segmentation and dynamic CNN optimization. Relevant literature on recent CNNs for semantic segmentation is presented followed by works which attempt efficiency optimization of CNNs by introducing spatial sparsity.

2.1. Semantic Segmentation

In this section, the related literature of semantic segmentation CNNs is grouped into two main categories according to the number of computation paths they include, i.e., single-branch or encoder-decoder and multi-branch architectures.

Single-Branch architectures appeared since the early work of FCN [10] and proposed a method to transform existing classification networks into fully convolutional networks that can predict pixel-wise labels for an input image, including the introduction of skip connection to allow the network to capture both high-level and low-level features. Since then, a common practice for semantic segmentation is to utilize architectures that are well established in the classification task as encoders, such as VGG [16], Resnet [6], MobileNetV2 [15], ShuffleNetV2 [11], EfficientNet [17], and add an upsampling module to recover fine details. SwiftNet [12] combines those backbones, pretrained on ImageNet [9], along with a spatial pyramid pooling (SPP) module [5] and basic decoder, formulating a versatile architecture.

Information lost by downsampling is impossible to recover by upsampling, which introduces an upper bound on the performance of encoder-decoder methods. To address that, architectures with multiple paths (**Multi-Branch**) have been proposed in the literature, aiming to maintain high resolution representations during execution. BiSeNetV1 [23] and BiSeNetV2 [22] introduce a deep path with strided convolutions to increase receptive field, with a parallel shallower path to obtain high-resolution information. The two paths are separate and a fusion module combines the outputs of the two paths efficiently. The recent CABiNet [14] follows a similar dual-branch approach and introduces a low-cost, compact asymmetric position and local attention block.

HRNet [20] architecture maintains high-resolution representations throughout the entire process by connecting parallel convolution streams operating at different resolutions and repeatedly exchanging information across resolutions. This approach enhances the quality of the representation, resulting in a semantically rich and spatially precise output that improves the overall performance of the network, while the complexity is curtailed by employing thinner branches on high resolution. The recently proposed DDRNets [7], further push the efficiency of multi-branch approaches by introducing a family of efficient backbones, carefully designed for semantic segmentation. They include a high and a low resolution branch, which communicate information using bilateral connections, while a Deep Aggregation Pyramid Pooling Module (DAPPM) is applied on the output of the low-resolution branch in order to extract multi-scale and global context information.

In general, multi-branch architectures boast increased efficiency against the single branch counterparts since they can accommodate high and low resolution paths without a significant increase in computations.

2.2. Leveraging Spatial Sparsity for Dynamic Optimization

There is a growing body of work on optimizing the computational efficiency of a CNN model, by exploiting spatial redundancy of computations. To eliminate redundancy in computation, [4] introduced various perforation configurations as binary masks to guide the skipping of computations on feature maps. Later, [3] proposed an approach to dynamically adjust the computation time of residual neural networks based on the spatial complexity of the input image, by halting the execution after a specific number of layers have been executed.

A method to dynamically apply convolutions conditioned on the input image was proposed by [18], introducing a computational block where a small gating branch learns which spatial positions should be evaluated. By dynamically altering the processing resolution of image regions, Segblocks [19] reduces the computational cost of existing neural networks for semantic segmentation. The input image is divided into high and low resolution blocks, guided by a policy network, which takes into account the both the

performance of the network and the complexity ratio between the two block categories. A methodology to control the computational complexity of a stereo-matching network on-demand by ranking the upside of applying disparity refinement on various image regions was proposed by [24]. Focusing on the super-resolution task, [21] uses a sparse masking module to simultaneously skip both unimportant spatial regions and mask redundant channels in them, resulting in more effient inference.

Previous research has primarily concentrated on optimizing in terms of FLOPs space, considering FLOPs as a proxy for latency. However, this approach fails to capture the additional overheads and complexities that influence the actual execution time of the CNN. In this work, practical latency is used to guide the optimization process, focusing on the task of semantic segmentation. Rather than focusing on overparameterized baselines, the target networks involve modern state-of-the-art multi-branch CNNs, showcasing the potential gains and challenges of practical utilization of such compression technique in real and challenging scenarios.

3. Motivation

As discussed in the previous section, multi-branch networks have emerged as the state of the art in semantic segmentation, taking into account the trade-off between performance and complexity. However, there is a pressing need to generate models that can further push the performance-accuracy trade-off, as this has significant implications for various applications, such as real-time deployment and resource-constrained scenarios. This makes it crucial to explore approaches that can optimize both performance and accuracy efficiently. One promising avenue is data-driven spatial sparsity optimization, which focuses on identifying and exploiting spatial redundancies within the data to achieve computational efficiency.

The ultimate objective of optimizing sparsity is to minimize latency, which is a critical factor in real-time applications, but since directly measuring latency can be challenging, researchers often rely on the number of floating-point operations (FLOPs) as a proxy. Previous work which performs spatial sparsity optimization either demonstrates subpar results when applied to multi-branch networks [19] or focuses on one specific layer type which does not translate to recent state-of-the-art approaches [18]. In light of these limitations, our work aims to address these challenges by targeting state-of-the-art networks while being cognizant of the desired latency constraints.

4. Method

Considering the conclusions from the previous section, we formulate a methodology that aims to increase the efficiency of CNNs for semantic segmentation by introducing spatial sparsity to multiple parts of the network as a way to reduce redundancy of computations. Additionally, instead of focusing on optimizing for theoretical computations (FLOPs), we introduce the actual latency of the sparse implementation into the optimization process. The optimization framework is inspired by [18], after adapting it to higher resolution data and introducing the latency optimization target. In the following sections, the key components of the optimization framework are described.

4.1. Mask Estimation

The mask estimation mechanism is responsible for predicting a sparsity mask which encodes whether calculations on the corresponding image location will be executed by the CNN layer or not. To estimate the mask we choose a lightweight module, comprising of a 2D convolution with input channels equal to the input feature map and output channels equal to 1. We further add Batch Normalization and Average Pooling modules to control the granularity of the output mask, by grouping image regions in square blocks. Departing from pixel level granularity [18], is not only helpful to reduce the latency when executing the spatially sparse variants, but also enforces a regularization during training leading to improved accuracy.

Training the mask decisions is not a straightforward task, since selecting top-k activations requires the minimization of the non-differentiable argmax function. Similar to [18], we use using the Gumbel-Softmax trick [8] to perform end-to-end training. Soft decisions can be transformed into hard decisions while maintaining the ability to perform back-propagation, which is necessary for optimizing the weights of the mask unit. Given $M_i \in \mathbb{R}^{N \times M}$ as the continuous output of mask estimation module for dynamic layer i, during training the argmax is replaced by

$$z_i = \frac{\exp((\log(M_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(M_j) + g_j)/\tau)} \quad (1)$$

with τ the temperature of Gumbel distribution where noise samples g_i are drawn from. During inference, locations where $M_i > 0$ are activated.

4.2. Sparse Implementation

Modern software libraries include highly optimized implementations for the main CNN operations, therefore to achieve reduced latency, a highly optimized sparse implementation is required for inference. We adopt the implementation of sbnet [13] to apply spatial sparsity to CNNs, which comprises of 3 main operations,

- The **mask reduction** module assigns image pixels to a grid and generates the grid indices which are activated, using the estimated binary mask as input.

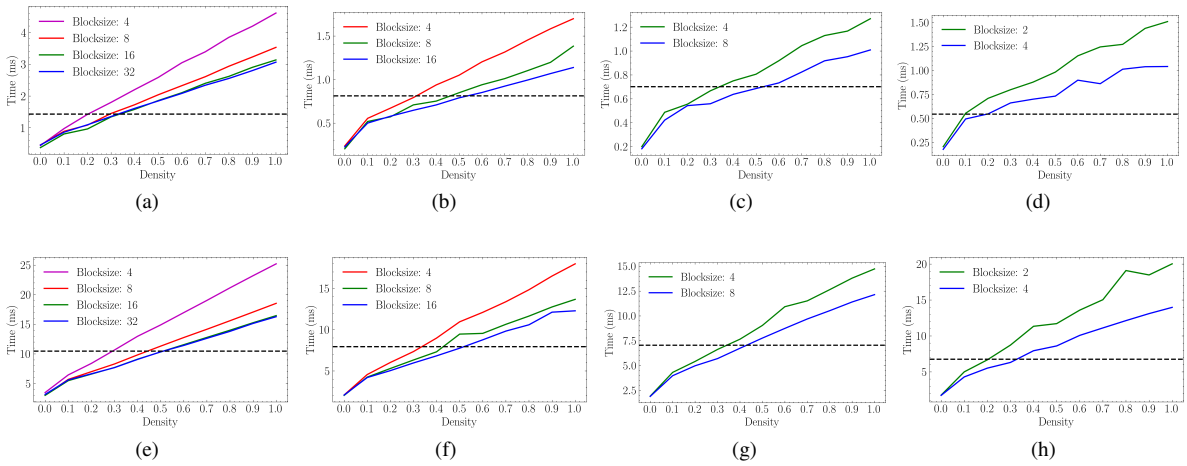

Figure 2. Figure presenting the sparse profile for different layers (rows) of DDRNet23-slim [7] in two devices: NVIDIA RTX 3060ti (a-d) and NVIDIA JetsonNX (e-h) under various blocksizes.

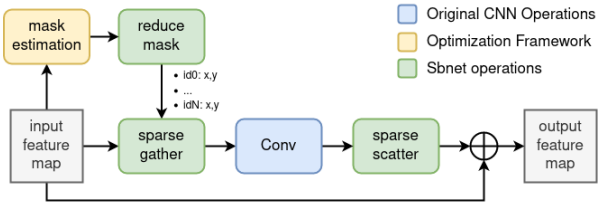

Figure 3. Sparse inference using sbnet [13] implementation. Guided by the mask estimation, only active regions of the dense feature map are extracted, and then processed. The result is used as a sparse residual, added subsequently on the dense input feature map.

- The **sparse gather** operation is responsible for transforming the data from dense to sparse domain. It extracts the activated blocks and stacks them in the batch dimension resulting in a tensor containing only the areas to be processed.

- The **sparse scatter** operation handles the transformation from sparse back to the dense domain. After the processing the activated areas are placed back to the original dense tensor, guided by the extracted grid indices.

The effectiveness of sbnet relies upon the ability to segregate the input data in blocks, resulting in multiple smaller workloads which benefit from the optimization of GPU devices for such workloads. Figure 3 depicts the flow during inference using sbnet operations to move between dense and sparse domains.

4.3. Sparse Benchmark Bank

When using FLOPs as an optimization objective, a linear function to model the density of computations of a single layer is used (blue line in Figure 4). When departing from such proxy to actual latency optimization, using such function would only hold if the sparse execution time is equally proportional to density, corresponding to an ideal sparse implementation with no overheads. Instead, a function which models the relation of mask density to actual latency for the given software and hardware implementations needs to be used. Such function $t(C, d), \quad d \in [0, 1]$ is selected, with $C = (layer, overheads, device, blocksize)$ representing a set of configurations and d the densities of the sparse layer.

To construt this function, we conduct a series of benchmarks for each individual layer of the CNN, using the sparse implementation which was introduced above. By noting the execution times for multiple density levels from 0% to 100% in steps of 10% and interpolating in between, a function is created. The execution time of the corresponding layer under the same configuration, $t_{dense}(C)$ is also calculated. To account for the mask estimation overhead, we further include the mask estimation computations in the benchmarks, resulting in a realistic latency profile of the CNN layers. Latency profiles for multiple layers of DDRNet23-slim [7] network for two devices can be seen in Figure 2. It is worth noting that optimized libraries perform multiple under-the-hood optimizations when executing CNNs, such as grouping computations, that individual layer benchmarking might fail to capture.

4.4. Latency Aware Training

To introduce actual latency to the optimization process, the timing function is used to map density of each layer to actual latency, replacing the FLOPs $t_{ideal}(d)$function which

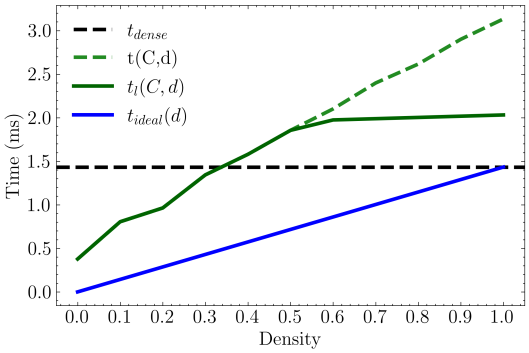

Figure 4. Example leaky latency function for the first residual layer of DDRNet23-slim used for training.

maps density to the number of theoretical computations. However, $t(C, d)$ function represents the actual execution time using the sparse implementation, which is specifically suited for low density workloads. Therefore, when dealing with higher densities, the execution times become considerably slower compared to t_{dense}. To exploit this, a mock sparse implementation can be employed at these density levels, executing the layer in a dense manner and subsequently applying the sparsity mask to negate the convolution effect for inactive regions. Although this implementation incurs a slight overhead for mask estimation, it remains faster than the sparse implementation at higher density levels. This introduces a two-fold benefit: an upper bound to the latency of each layer, and the ability to explore higher densities for critical layers, without significant latency costs. In order to introduce this concept to training, we formulate a two-part function to map density to latency as,

$$t_l(C,d) = \begin{cases} t(C,d) & d < d_{thresh} \\ t_{dense}(C) + s * t_{dense}(C) & \\ \quad *(d - d_{thresh}) & d >= d_{thresh} \end{cases} \tag{2}$$

where d_{thresh} is the density threshold where beyond that, the sparse execution time becomes slower than the dense. A slight slope is also added to the second part to allow convergence during training, set to $s = 0.01$. The resulting $t_l(C, d)$ which is used during training, is demonstrated in (Figure 4). This allows the optimization process to push only the layers which can benefit from latency gains by introducing spatial sparsity to them and at the same time avoid sparsifying layers which would not be pushed under the latency threshold of t_{dense}.

4.5. Sparse Optimization Framework

Given a set of activation masks, we denote the sum of FLOPs of all dynamic block as F_{sp} and the corresponding maximum total of FLOPs as F_{tot}. Similarly, when optimizing for latency budget l_b, the density of the activation masks is projected to execution time using equation 2, resulting in latency sums of dynamic and dense executions, T_{sp} and T_{tot}. Similar to [18], providing a target budget l_b or $f_b \in [0, 1]$, the goal of training is to minimize network sparsity loss ($L_{sp,network}$) or the ratio between F_{sp} and F_{tot} using $(\frac{F_{sp}}{F_{tot}} - f_b)^2$ or T_{sp} and T_{tot} using $(\frac{T_{sp}}{T_{tot}} - l_b)^2$ correspondingly. The total loss is given by:

$$L = L_{\text{task}} + \alpha L_{\text{sp, network}} \tag{3}$$

with L_{task} being the Cross Entropy Loss for semantic segmentation and α is a hyperparameter to control the difference in order of magnitude between task and sparsity loss. We omit the boundary terms used by [18] to encourage the freedom of activations to explore the whole space from the start, since latency optimization usually performs best with activation rates close to boundaries.

5. Experimental Results

5.1. Train Setting

The Cityscapes [1] dataset is a popular dataset for semantic segmentation, which consists of 2975 finely annotated images for training training, 500 validation and 1525 test images of 2048×1024 pixels. We use pretrained checkpoints for DDRNet23-slim, DDRNet23 [7] and HRNetV2-W48 [20] on Cityscapes, and further train for 150 epochs under the sparse optimization framework. DDRNet models are trained with an input resolution of 1024×1024, a batch size of 8, whereas 1024x512 and batch size of 4 is used for HRNetV2 instead. The networks are trained using the Adam optimizer. The initial learning rate for network weights is $1e{-}5$ and $2e{-}3$ for mask estimation weights. At epochs 60, 90, and 120, the learning rate decreases by a factor of 10. The Gumbel temperature starts at 4 and gradually decreases to 1. The preprocessing pipeline follows [7] including argumentation of random cropping images, random scaling in the range of 0.5 to 2.0, and random horizontal flipping. For the comparison with state-of-the-art compression methods, we use the publicly available code for Segblocks to train the multi-branch models. As for Dynconv [18] we simply set $blocksize = 1$ and include the boundary sparsity loss terms using the FLOPs optimization objective.

5.2. Quantitative Results

Results of the FLOPs optimization are presented in Figure 5, where f_b denotes the target budget for sparse layers and "hb" denotes compression of high resolution branch only. Introducing sparsity masks separately for each layer while skipping processing in inactive regions shows improved efficiency in exploiting spatial sparsity in multi-

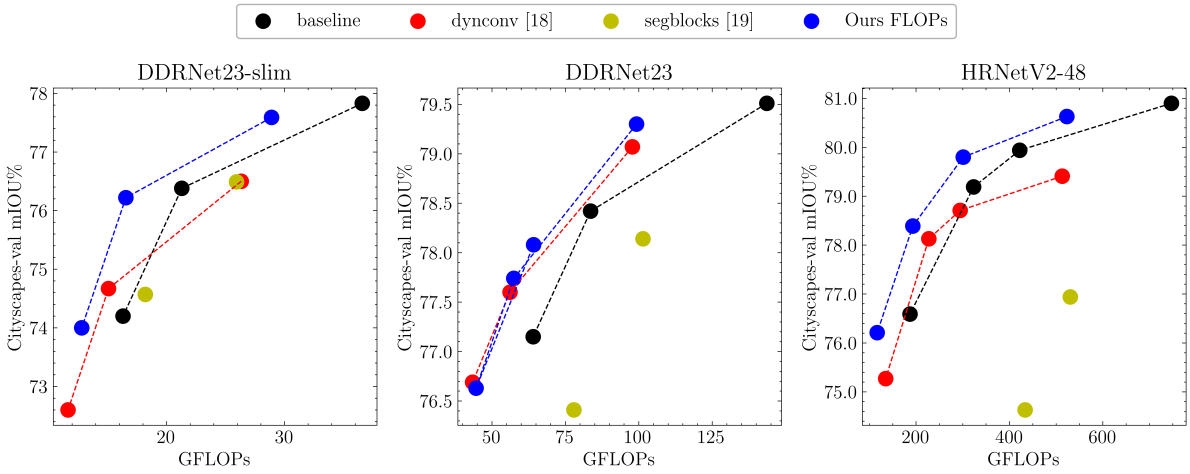

Figure 5. FLOPs optimization of DDRNet23-slim [7], DDRNet23 [7] and HRNetV2-48 [20] multibranch architectures compared to SoA compression methods using spatial sparsity. Multiple connected points are generated by evaluating the full resolution models at different scales.

branch architectures. On the other hand, Segblocks [19] efficiency in comperssing single-branch networks does not directly transfer to the multi-branch domain (Figure 5). Additionally, grouping image areas into blocks, as opposed to individual pixels like dynconv [18], proves advantageous in tasks involving high-resolution images, such as semantic segmentation.

Optimizing based on FLOPs consistently reduces computations but this does not always directly translates to latency. Table 1 1 illustrates our network optimization results considering actual latency. We denote l_b as the target latency budget of the sparse layers for our optimization framework. Latency aware optimization achieves improved execution times of DDRNets compared to latency agnostic optimization and improved efficiency when compared to other compression methods at similar compression rates. While Segblocks [19] achieves higher reduction in latency of HRNetV2-48 [20], accuracy deteriorates much more than our approach. Moreover, applying the optimization only on the high resolution branches rather than the full network demonstrates increased efficiency, highlighting the existence of more redundant computations in it. In general, 20% to 40% of total FLOPs can be reduced with minor impact on accuracy, depending on the sensitivity of the network.

To explore our latency optimization framework's portability, we assess it on an NVIDIA Jetson NX embedded GPU. These devices are relevant in real-world scenarios where low latency and high performance are paramount. To achieve that, the individual benchmark functions for this device (Figure 2, e-h) are used and the final benchmarks are performed on the device. The consistent results (Table 2) validate the successful transfer and effectiveness of our optimization technique. Moreover, the latency optimization framework can be extended to support other devices, needing only an implementation achieving practical speedup under sparsity.

6. Ablation Studies

6.1. Block Size

The choice of block size plays an important role in the current optimization framework as stated in 4.1 for two reasons. Firstly, it introduces a sort of structure to the data, which is essential to mitigate the overheads for sparse execution in GPU devices. Therefore it allows some space for the realization of latency gains on GPU devices using sbnet sparse operations. Secondly, grouping image pixels in a region seems to have a beneficial regularization effect in training as seen in Table 3. The 1×1 2D convolution operations used by mask estimation layers might be too isolated to reason about the potential upside of processing each pixel, whereas combining them with a spatial operation such as pooling, results in more effective reasoning about the region.

6.2. Target budget

We conduct an ablation study on the target budget of the optimization process, to investigate the relationship between network sparsity, model complexity, and predictive capability. The analysis is also helpful to get insights for the most efficient trade-offs between computational efficiency and performance. In the DDRNet-23 case (Figure 6), extreme sparsities in the full network optimization lead to

Δ FLOPs %	Method	Δ Latency %	Δ mIOU%
	DDRNet23-slim [7]		
-30%	dynconv [18] ($\tau = 0.4$)	10.35	-2.67
	segblocks [19] ($\tau = 0.6$)	-6.86	-1.34
	Ours $f_b = 0.5$	11.31	**-0.59**
	Ours $l_b = 0.8$	**-7.85**	-1.29
-20%	dynconv [18] ($\tau = 0.65$)	10.35	-0.73
	segblocks [19] ($\tau = 0.7$)	-4.13	-0.85
	Ours hb $f_b = 0.3$	-0.37	**-0.24**
	Ours hb $l_b = 0.8$	**-11.1**	-0.33
	DDRNet23 [7]		
-30%	dynconv [18] ($\tau = 0.4$)	7.39	-0.92
	segblocks [19] ($\tau = 0.6$)	11.15	-1.37
	Ours $f_b = 0.5$	5.52	**-0.21**
	Ours $l_b = 0.8$	**-7.31**	-0.90
-20%	dynconv [18] ($\tau = 0.65$)	7.39	-0.52
	segblocks [19] ($\tau = 0.7$)	-7.92	-0.89
	Ours hb $f_b = 0.3$	-6.72	**-0.21**
	Ours hb $l_b = 0.6$	**-12.73**	-0.31
	HRNetV2-W48 [20]		
-40%	dynconv [18] ($\tau = 0.6$)	18.49	-2.51
	segblocks [19] ($\tau = 0.4$)	**-30.71**	-6.27
	Ours $f_b = 0.5$	-6.64	**-0.80**
	Ours $l_b = 0.8$	-11.48	-1.09
-30%	dynconv [18] ($\tau = 0.7$)	18.49	-1.41
	segblocks [19] ($\tau = 0.6$)	**-16.00**	-3.96
	Ours hb $f_b = 0.5$	-2.73	**-0.27**
	Ours hb $l_b = 0.8$	-12.54	**-0.27**

Table 1. Table presenting the result of optimization for Cityscapes dataset for DDRNet23-slim, DDRNet23 and HRNetV2-48 multi-branch architectures, evaluated on the validation set, under various FLOPs reduction regimes.

Method	Δ Latency %	Δ mIOU%
NVIDIA RTX 3060ti		
$l_b = 0.8$	-7.31	-0.90
hb $l_b = 0.8$	-10.20	**-0.22**
hb $l_b = 0.6$	**-12.73**	-0.31
NVIDIA Jetson NX		
$l_b = 0.8$	-7.50	-0.47
hb $l_b = 0.8$	-5.25	-0.34
hb $l_b = 0.6$	-10.07	-0.26

Table 2. Latency optimization results for DDRNet23 [7] network performed on two different GPU devices: a desktop NVIDIA RTX 3060ti and an embedded NVIDIA Jetson NX.

sub-optimal states, while the most efficient compression is achieved when targeting budgets within the range of 0.5 to 0.7. On the other hand when optimizing the high-resolution branch only, the effective target budgets fall within the interval of 0.2 to 0.4.

6.3. Slope and loss boundary terms

This section explores the impact of design choices of the training process when moving from FLOPs to latency optimization (Table 4). Omitting the boundary loss term allows the optimization to explore the full space

Block size	mIOU%	Latency (ms)
1	76.06	112.11
8	76.12	41.90
32	76.85	20.79
64	**77.00**	20.65
128	76.89	**20.59**

Table 3. DDRNet23-slim optimized with target budget $f_b = 0.50$ with varying blocksizes in full image resolution. Latency measured on a NVIDIA RTX 3060ti.

no boundary loss term	timing slope	mIOU%
✗	✗	79.78
✗	✓	79.99
✓	✗	80.51
✓	✓	80.63

Table 4. The impact of removing the initial boundary loss term and addition of the slope in the timing function, for latency optimization of the high resolution branch of HRNetV2-48 using $l_b = 0.8$.

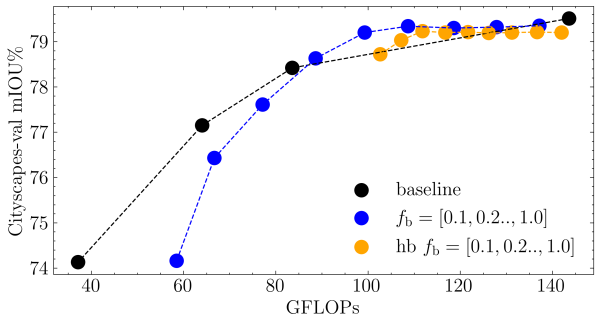

Figure 6. Performance of sparse DDRNet23 [7] under various target budgets $fb \in [0.1, 0.2.., 1]$

from the start, therefore reaching to more extreme sparsities/densities when necessary. The addition of slope in the timing function favors the mask estimation to push higher densities when the density threshold for practical gains is surpassed, but shows a smaller impact in practice.

7. Discussion

7.1. Class Sparsity Analysis

As we delve into the relationship between sparsity and multiple classes, it becomes evident that the impact of sparsity is far from uniform, as seen in Figure 7. Some classes exhibit high sparsity ratios, indicating that only a small portion of the available features contribute meaningfully to their representation. This phenomenon can be attributed to inherent characteristics of certain classes that render them

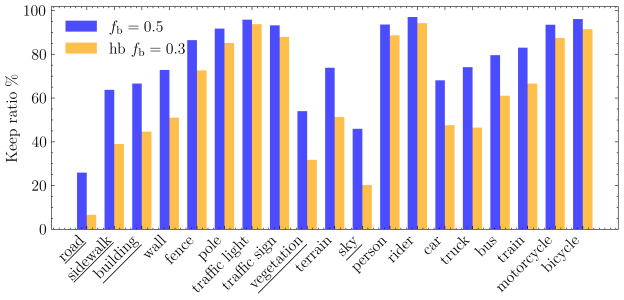

Figure 7. Percentages of active pixels for each class of sparse DDRNet23 [7] network. Pixels of *coarser* classes (underlined) are more frequently masked than those from *finer* classes.

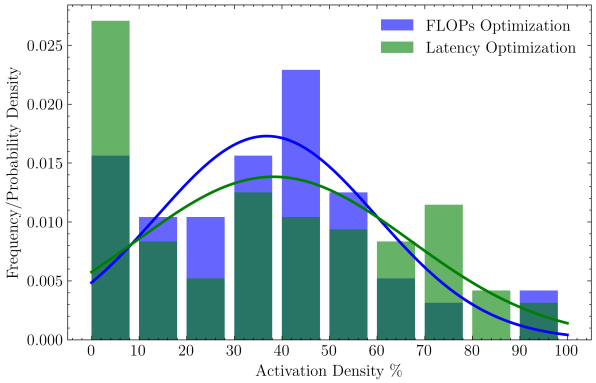

Figure 8. Histogram with actiation rates for sparse layers of HRNetV2-48 optimized for FLOPs ($f_b = 0.5$) and for latency ($l_b = 0.8$).

naturally less cluttered or complex. On the other hand, certain classes demonstrate lower sparsity ratios, where a higher portion of their features are actively engaged in their representation. This analysis proves the both the existence of redundancy of computations within the networks, which is exploited in order to reduce the computational budget, and the necessity of a masking strategy to select the optimal regions to skip.

7.2. Latency vs FLOPs Optimization

Both FLOPs and latency optimization attempt to minimize a target budget by skipping computations in spatial locations. However, there is an important difference in the way computations are distributed between layers. When optimising for FLOPs, the activation rates follow a distribution closer to mean activation aligning with the target budget, whereas when optimizing with a latency objective, activation rates tend to spread to more extreme values in both directions (Figure 8). The reason is that latency gains for a layer can be achieved below a certain density threshold and are maximized closer to zero density. On the other hand, if this threshold is exceeded, the dense implementation followed by binary masking can be used instead, taking equal time as 100% activation. Therefore, in latency optimization, extreme activation rates are preferred as they offer options to reduce latency or increase density, a behavior encouraged during training using the leaky timing function.

7.3. Comparison to SegBlocks

Given the relevancy of the proposed work with Segblocks [19], we discuss some interesting conclusions. Segblocks takes advantage of the over-parameterized nature of single branch architectures for semantic segmentation, which try to accommodate both the deep features and fine details in all feature maps, to *essentially generate two branches of computation*. However, recent multi-branch architecture have *already accounted for that* by defining multiple paths of computation, and therefore, redundancy is less apparent in those approaches constituting compression a more challenging task. Moreover, multi-branch networks don't seem to benefit from having the two paths of computation of Segblocks in the application of sparsity, since it practically doubles the number of branches, creating potentially inefficient overlaps of computation in some resolutions.

8. Conclusions

This work presents a methodology to increase the computational efficiency of multi-branch CNNs for semantic segmentation. For each layer, estimated sparsity masks encode if various image regions should be processed or not, formulating a spatially sparse execution. They key concept is the introduction of latency to the optimization process rather than relying in proxies such as FLOPs. To achieve that, actual latency profiles for each layer, calculated from benchmarking CNN operations on target hardware, are provided to supervise the optimization process. Results demonstrate increased performance-latency efficiency when compared to previous hardware agnostic theoretical optimization. We believe that this work can facilitate an optimization framework, where the practical impact of optimization can be controlled, leading to device-tailored sparse designs. Finally, including faster and more optimized sparse implementations closer to the ideal function, can further boost the framework to achieve improved latency gains. This work is also introduced in Large Language and Vision Models for Autonomous Driving (LLVM-AD) workshop summary [2].

Acknowledgement

For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

References

[1] 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. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 5

[2] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, Yang Zhou, Kaizhao Liang, Jintai Chen, Juanwu Lu, Zichong Yang, Kuei-Da Liao, Tianren Gao, Erlong Li, Kun Tang, Zhipeng Cao, Tong Zhou, Ao Liu, Xinrui Yan, Shuqi Mei, Jianguo Cao, Ziran Wang, and Chao Zheng. A survey on multimodal large language models for autonomous driving. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, 2024. 8

[3] Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks, 2017. 2

[4] Michael Figurnov, Aijan Ibraimova, Dmitry Vetrov, and Pushmeet Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions, 2016. 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *Lecture Notes in Computer Science*, page 346–361, 2014. 2

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[7] Yuanduo Hong, Huihui Pan, Weichao Sun, and Yisong Jia. Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. *arXiv preprint arXiv:2101.06085*, 2021. 2, 4, 5, 6, 7, 8

[8] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 3

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017. 2

[10] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 2

[11] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 2

[12] Marin Orsic, Ivan Kreso, Petra Bevandic, and Sinisa Segvic. In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12607–12616, 2019. 2

[13] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. Sbnet: Sparse blocks network for fast inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8711–8720, 2018. 3, 4

[14] Saumya Saksena. Cabinet : Efficient context aggregation network for low-latency semantic segmentation, August 2020. 2

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 2

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 2

[17] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. 2

[18] Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2020. 1, 2, 3, 5, 6, 7

[19] Thomas Verelst and Tinne Tuytelaars. Segblocks: Block-based dynamic resolution networks for real-time segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):2400–2411, 2023. 1, 2, 3, 6, 7, 8

[20] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(10):3349–3364, 2021. 2, 5, 6, 7

[21] Longguang Wang, Xiaoyu Dong, Yingqian Wang, Xinyi Ying, Zaiping Lin, Wei An, and Yulan Guo. Exploring sparsity in image super-resolution for efficient inference. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4917–4926, 2021. 1, 3

[22] 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

[23] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation, 2018. 2

[24] Georgios Zampokas, Christos-Savvas Bouganis, and Dimitrios Tzovaras. Pushing the efficiency of stereonet: Exploiting spatial sparsity. In *VISIGRAPP*, 2022. 1, 3