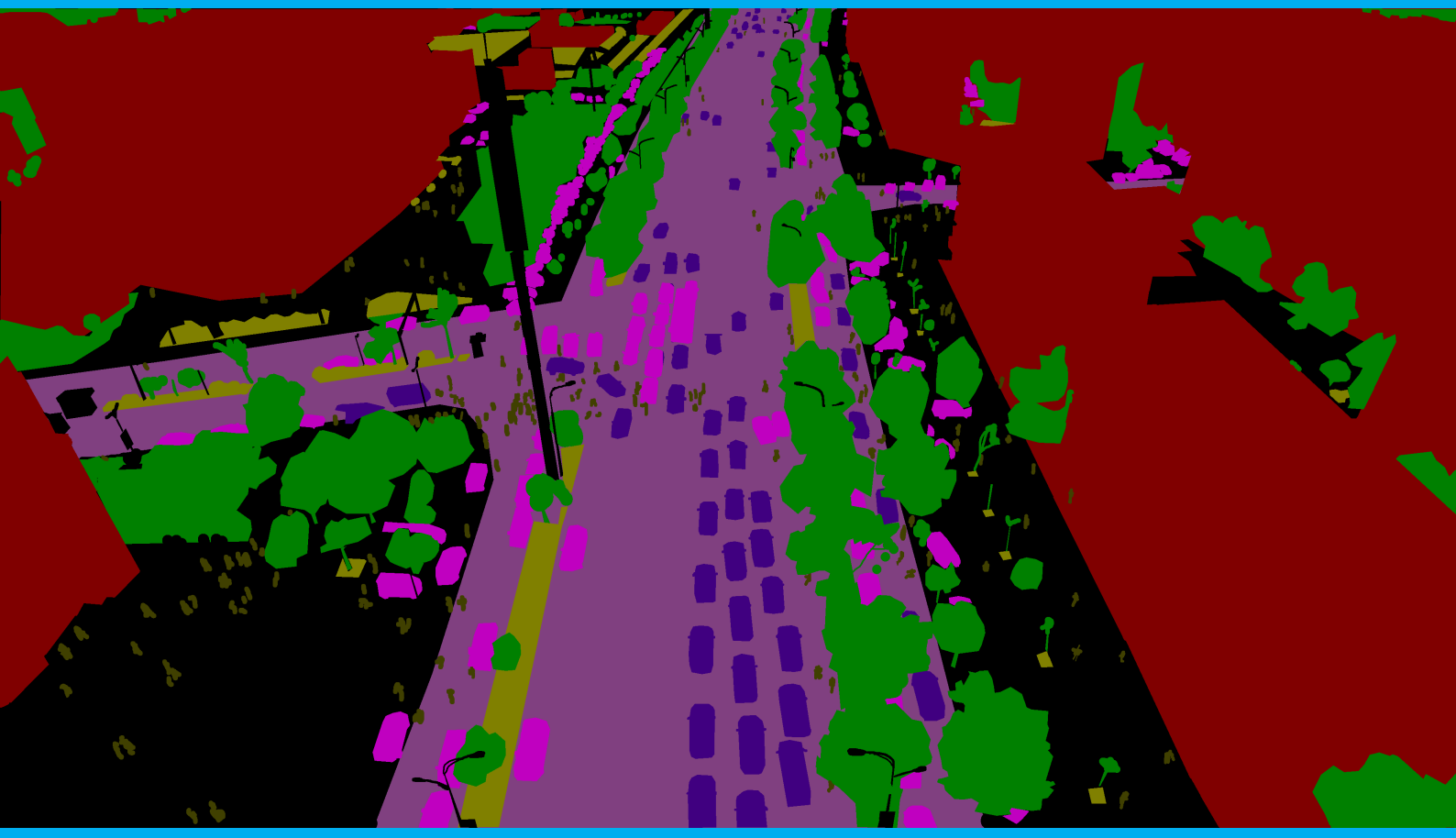
CABiNet

Efficient Context Aggregation Network for Low-Latency Semantic Segmentation

Saumya Kumaar Saksena

Such GPU,
Much Learning,
Wow.



UNIVERSITY
OF TWENTE.

CABiNet

Efficient Context Aggregation Network for Low-Latency Semantic Segmentation

by

Saumya Kumaar Saksena

to obtain the degree of Master of Science in Systems and Control
at the University of Twente.

Student number: s2084627
Project duration: February 1, 2020 – October 2, 2020
Thesis committee: Prof. dr. ir. George Vosselman, UT, Committee Chair
Dr. ir. Michael Ying Yang, UT, Academic Supervisor
Dr. ir. Nicola Strisciuglio, UT, External Examiner

UNIVERSITY
OF TWENTE.

Preface

It is a pleasure to submit my academic thesis towards the partial fulfillment of my Masters' course on Systems and Control. A summary of the accomplished tasks and milestones throughout the course of my research has been presented in this document. On a personal level, this graduation thesis presented me with multiple challenging tasks, which essentially led to a steep learning curve. My programming skills have been sharpened further and I have witnessed an overall development in my personality.

I would genuinely like to thank my PhD supervisor, Ir. Ye Lyu, who has been an immense inspiration and thoroughly supportive throughout my tenure and my academic supervisor Dr. Michael Yang, for constantly providing me with his esteemed guidance. I would like to thank my parents for making me capable enough to take on these challenges sportingly. Indeed there were local minimas and pitfalls along the way but attempted to make sure that eventually, everything falls into its place, which fortunately happened. So it almost seems like someone upstairs is happy with me for granting me with wonderful people and opportunities.

Saumya Kumaar Saksena
Enschede, August 2020

Contents

List of Figures

List of Tables

1

Abstract

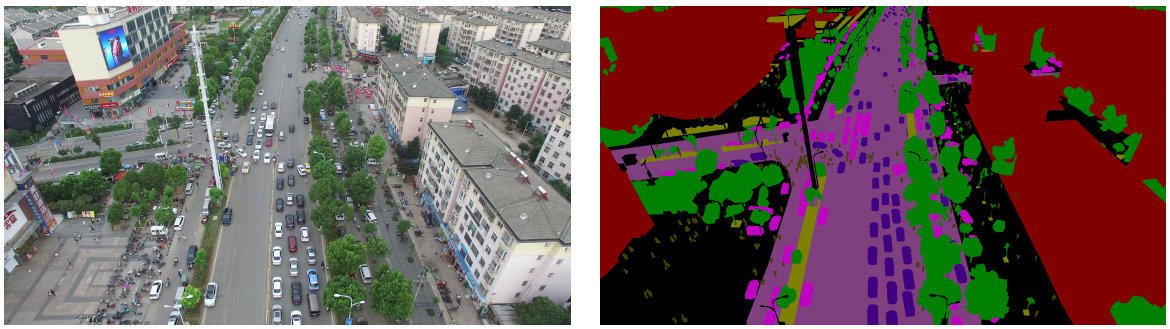
Real-time semantic segmentation is a challenging task as the optimal balance between accuracy and efficiency (computational complexity, memory footprint and execution speed) is hard to achieve. Conventional lightweight and real-time semantic segmentation architectures usually address only one of the above perspectives, thereby making high-accuracy designs computationally expensive and high-speed models relatively inaccurate. In this research, we introduce an approach to semantic segmentation (for images), which successfully reduces the computational costs by almost 88% and increases the execution speed by a factor of 1.5 compared to the current state-of-the-art, while maintaining a comparable mean intersection-over-union score. Building upon the existing multi-branch architectures for high-speed segmentation, we design a cheap high resolution branch for effective spatial detailing and a context branch with compact asymmetric position and local attention (collectively termed as Context Aggregation Block), potent enough to capture both long-range and local contextual dependencies required for accurate semantic segmentation, at low computational costs. Specifically, we achieve 76.6% and 75.8% mIOU on Cityscapes validation and test sets respectively, at 76 FPS on a single NVIDIA RTX 2080Ti and 8 FPS on a Jetson Xavier NX. Our superior context aggregation techniques also outperform the current state-of-the-art on another public benchmark, UAVid dataset, by a significant margin of 14%. Codes and pre-trained models shall be made available at https://github.com/dronefreak/CABiNet.

2

Introduction

2.1. Motivation and Research Statement

In the domain of computer vision or machine perception, semantic segmentation refers to the process of dividing a digital image into segments that share similar characteristics. The target of this process is to transform the complex input image into a notation which contains more information and can be easily interpreted by a machine. In this process, we normally assign a class-label to each and every pixel, such that the pixels with similar labels have some common features. For instance in Fig. 2.1, the input RGB image is shown on the left, whereas the semantic labelling is shown on the right, which clearly demarcates the boundaries and extent of different objects in the RGB image.



(a) RGB Image **(b)** Ground Truth for Semantic Segmentation

Figure 2.1: Single training sample from the UAVid dataset [60]

Semantic segmentation has found applications in several aspects. As we map every pixel into a target class, land usage mapping in satellite imagery becomes a very potential use-case. Land-cover information in turn, could be important for monitoring forest cover [17], agricultural lands [20] and urban settlement expansion [47, 98, 107]. This use-case relies on multi-class semantic segmentation, thereby partitioning roads, buildings, urban settlements etc into different segments. Another application is the field of self-driving cars or autonomous machines to be more general [67, 72, 73]. Autonomous driving is a very complicated task, requiring control, perception and accurate decision-making, all happening together within complex and variable environments. Furthermore, all operations in this aspect need to be extremely precise as safety is of utmost priority. Image segmentation techniques [7, 67, 72, 73, 78, 112, 113] can provide information about nearby objects, free space on roads, ego-lanes etc. more accurately than conventional object-detection and recognition algorithms, especially in the case irregular shaped objects like roads etc.

Facial segmentation is another extensively researched field, where age/gender prediction, expression recognition etc. become easier if the eyes, nose, mouth and other facial features are accurately segmented and separately studied [45, 75, 83, 85, 86]. It is to be noted that facial segmentation is affected by factors like face orientation, expressions and other environmental factors. Another very challenging task is cloth-parsing, i.e. understanding the type of fabric/cloth is present or used for manufacturing a certain product.

Clothing parsing is more challenging than others because there are immense number of classes to be categorized [21, 35, 50, 111], which is coupled with the fact that fine-grained clothing segmentation may require additional post-processing techniques to acquire reliable results.

Semantic segmentation also has applications in medical imaging [37, 38, 81, 91]. For instance, radiologists and other specialists have to analyse multiple medical images for a reliable diagnosis. However, the complexity in medical imaging like overlapping regions, contrast etc. can cause trouble to even trained specialists and hence, systems employing semantic segmentation could provide assistance to these professionals in understanding the images better [8, 25, 92]. Now that we have established the potential of the concept of semantic segmentation, we would like to introduce this research, where we develop a strategy for urban scene understanding specifically.

2.2. Research Objectives and Expected Outcomes

The primary objective of this research is to develop a robust, convolutional neural network (CNN)-based design for real-time urban scene understanding, which has an optimal balance between computational expenses, execution speed and the overall prediction accuracy. Specifically, we perform the below-mentioned tasks in this research:

1. Conceptualize an effective semantic segmentation architecture for real-time applications
2. Implement and train the above conceptualization in a targeted deep learning framework on public benchmarks
3. Validate and test the methodology on the testing sets of the chosen benchmarks
4. Compute inference speeds on mobile platforms
5. Provide a detailed comparison between the state-of-the-art real-time segmentation architectures and our proposed method

We further break down each of the aforementioned points into research aspects (RA) to make the overall analysis easier.

2.2.1. RA 1 - Conceptualization

Semantic segmentation for urban scene understanding has been addressed using various techniques. Conceptualization hence, can be further broken down into the following:

- Determine the specific feature requirements for accurate semantic segmentation
- Analyze how different architectures fulfill the above requirements
- Analyze the computational requirements of the above models
- Based on the above two analyses, determine which design is best suitable for fast semantic segmentation
- Analyze the shortcomings in the best selected architecture and look for possible remedies

So this is the stage where the shortcomings have been effectively studied and in the next stage we move on to implementing the newly designed strategy.

2.2.2. RA 2 - Implementation

A lot of deep learning frameworks are available today like TensorFlow [1], PyTorch [70], MXNet [13] etc. MATLAB also has support for deep learning applications, although multi-GPU configurations could be a bit tricky. Considering the ease of programming which allows the user to focus more on the architecture, we decided to move ahead with PyTorch [70], because it has a very consistent programming structure, full GPU support including multi-GPU setups and supportive documentation. Now once the draft model is ready (which attempts to solve the shortcomings in the best selected architecture tentatively), the need for training on datasets arises, so there is yet another list of questions that need to be answered:

- Determine the most commonly used public benchmarks for semantic segmentation
- Determine the types of challenges the chosen datasets present, like scale variations, class imbalance etc.
- Determine the type of problem the aforementioned design attempts to resolve
- Determine the label encoding and data-loading techniques for the chosen datasets

Once the data-loaders are ready we needed to initiate my training, where several parameters needed to be determined before-hand. For instance, the batch-size, total training iterations, initial learning rate, learning rate manipulation, etc. In order to determine these parameters, we read several state-of-the-art articles and experimented with different settings. All these values have been discussed in the later sections of this report, which were used to train the final model.

2.2.3. RA 3 - Validation and Testing

Once a deep learning architecture has been trained, it needs to be evaluated and tested on the evaluation sets and testing sets respectively. Once again this domain involves answering certain questions for effective progress to the next stage:

- Determine the metrics for evaluation and testing, based on the problems that the proposed design attempts to solve
- Determine the validation strategy like k-fold etc. and the parameters like crop size, batch-size etc.
- Obtain predictions on testing sets and submit to the official servers for final evaluation

Once we have the above results with us and if they are satisfactory, we move on to the next stage of inference.

2.2.4. RA 4 - Inference on Mobile Platforms

Real-time architectures are usually, in literature, benchmarked on high-end GPUs, like Titan XP or Titan V. Inferencing on full scale GPUs like Titan X, RTX20 series etc. is unlikely to provide a real-world analysis, as self-driving cars and other autonomous vehicles like UAVs or UGVs are more likely to have low-power consumption modules, with limited memory resources like Drive AGX, Jetson TX2, Xavier NX etc. Hence, it is important that we choose a low-power consumption module for inference that has a better real-world value.

2.2.5. RA 5 - Comparison with SOTA and Others

As we mentioned earlier, most of the real-time models are not benchmarked on embedded platforms. Hence, it is important to:

- Check for the official implementations of the SOTA architectures
- Benchmark the official models on an embedded platform
- Compute GFLOP, MAdd count and the execution speed on a common input resolution for reasonable comparison
- Highlight the advantages (or disadvantages, if any) of the proposed model over SOTA

So in a nutshell, the innovation in this research is aimed at developing a neural architecture that requires lesser memory footprint, has lesser computations thereby a larger inference speed and a comparable overall mIOU score as compared to the SOTA.

2.3. Report Structure

The following document contains 7 chapters. We begin with a detailed related work analysis, and complete each of the above mentioned stages in a sequential order. The above mentioned strategy complies more or less with the overall structure of the report and the research questions have been answered in respective sections. In the last two chapters, we present certain possible improvements that could be made and present a short conclusion for this research.

3

Related Work

The concept of image segmentation can be broadly classified into three major categories, namely, **semantic** segmentation, **instance** segmentation and **panoptic** segmentation. Each of the three essentially aim to map every pixel in the image to a certain possible category, but they differ in their core concepts and final visualizations. Semantic segmentation related literature has been covered in detail as it is the focal point of this research. Instance and panoptic segmentation techniques have been covered briefly. Some commonly used strategies for image segmentation have been shown in **Fig.** 3.1.

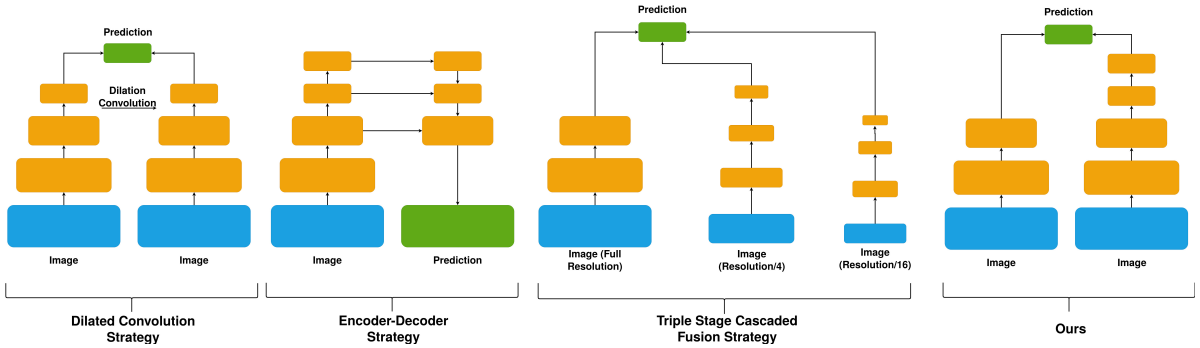


Figure 3.1: Common image segmentation strategies found in literature. Starting from left we have dilated convolution technique utilized by [11, 114]. Second is the standard FCN [58, 65] type structure, also called as U-Net [80] or Encode-Decoder [11, 44, 109, 123]. Next up, is the triple stage fusion strategy proposed in [119], and the last technique is a generic multi-branch architecture design which we use as a baseline in this research.

3.1. Semantic Segmentation

Semantic segmentation has witnessed a significant research input, thereby resulting in various methodologies and their numerous possible variations. In this chapter, instead of writing about every semantic segmentation paper in a random fashion, I categorize them into multiple domains thereby covering the different designs more effectively.

3.1.1. FCNs

The realm of semantic segmentation was revolutionized in 2015, when the concept of using fully convolutional networks was established by Long *et. al.* [58]. They adapted the feature representation of commonly used image classifiers like AlexNet [41], VGG [90] etc. into fully convolutional networks and finetune the representations on segmentation tasks. Similarly, Noh *et. al.* [65] also implemented a learning-based deconvolution network for enhancing the spatial information from the previous encoding module. These architectures are commonly termed as encoder-decoder designs, where the first part, the encoder, enlarges the receptive field by reducing the size of the convolution features. Then the next block upsamples the downsized feature vectors to create a full-scale semantic prediction of the input image. Skip connections were introduced in [3]

and [4] to improve the learning performance of decoders. A study on the importance of global context encoding for fully convolutional networks was performed by Zhang *et. al.* [116], where a novel context encoding module was introduced for selective strengthening of class-wise feature maps.

3.1.2. CRFs

A unique discriminative probabilistic modelling technique in machine learning that handles contextual information embedding effectively. Introduced in 2001 by Lafferty *et. al.* [42], the concept has been widely implemented for several computer vision applications like object recognition and gesture prediction [74, 82, 100, 103]. Modern research approaches incorporate CRFs for image segmentation as well [7, 22, 62, 97, 104]. This concept was introduced in 2014 by Chen *et. al.* [10], where they were able to make the design end-to-end trainable and achieved better results when compared to the then state-of-the-art algorithms. [7, 97] used the Gaussian variations of CRFs for semantic segmentation.

3.1.3. Spatial Pyramid Pooling

This concept was introduced in 2015 by He *et. al.* [28] to tackle the challenge of arbitrary input sizes for convolutional neural networks. Putting the concept to test in visual object recognition, the methodology outperformed all the previous architectures. With this inspiration, later in 2017, Chen *et. al.* [12] modified the original pooling module by replacing the pooling layers with dilated convolutions of variable weights, specifically for the target of semantic segmentation, thereby creating a new atrous spatial pyramid pooling module (ASPP), which became the gold standard for encoder-decoder architectures. Zhao *et. al.* [118] further improved the overall results by strategically placing the pooling module after certain layers for effective context embedding over different scales. Recently, improvements were suggested for ASPP module considering its computational requirements [31, 57] and its restricted receptive field [109], which assist in overcoming the limitations.

3.1.4. Self Attention

Attention modules have the capability to model long-range dependencies, and several researchers have employed the concept of attention in various works [51, 54, 87, 96]. The introduction of attention to machine understanding was achieved first in [54], where the global dependencies of inputs were learnt, which were then applied to natural language processing. Since then, a lot works have utilized this concept for several scene understanding tasks at both single and multiple scales [23, 34, 48, 76, 94, 120, 121], thereby outperforming the previous conventional context embedding methodologies. Oktay *et. al.* [66] apply the concept of attention-based segmentation to medical imaging, whereas Niu *et. al.* [64] employ it for semantic segmentation from aerial images.

3.1.5. Convolution Variations

In order to further optimize the performance of semantic segmentation models, several researchers have often used different convolution strategies like atrous convolutions [11, 12], dilated convolutions [71], depthwise convolutions [72, 73] etc. It was also established in [71], that large kernel sizes could be key to effective spatial detailing for accurate semantic segmentation.

3.1.6. Real-time Semantic Segmentation

Coming to the focal point of this research, real-time segmentation has addressed using multiple approaches. Romera *et. al.* [78] proposed to use factorized convolutions with residual connections for maintaining a balance between accuracy and execution speed. Poudel *et. al.* [72] suggest a dual-branch network with bottlenecks to effectively capture local and global context for fast segmentation. Later they propose an improved learning-to-downsample module in [73] for improved trade-offs between execution speed and accuracy. Two other highly accurate dual-branch segmentation networks were suggested by Yu *et. al.* [112, 113], where they designed novel feature fusion and attention refinement modules for accurate semantic segmentation tasks. In the second work, they redesign the feature aggregation methodology to further improve the execution speed, at considerable costs of accuracy. Multiple encoder-decoder pairs with multi-scale skip connections were also studied in this regard in [123]. This ensemble of shallow and deep paths is viewed as a shelf of multiple networks allows for effective feature representation with shallower backbones like ResNet-34, as compared to [112, 118]. Triple branch cascaded feature fusion strategy was extensively studied in [119], with significant resource consumption. Another approach to real-time segmentation is by using depth-wise

asymmetric bottlenecks [43], which theoretically provides for sufficient receptive field as well as captures dense context.

Neural architecture search techniques have proven to outperform the state-of-the-art designs in several aspects such as image classification [31] etc. which search optimal building blocks of networks. These search techniques however, fail to determine several other crucial aspects such as depth, downsampling etc. Sun *et. al.* proposed join-search framework which automates the search of optimal building blocks, as well as network depth, downsampling techniques and feature aggregation. Recently, graphic-guided architectural search pipelines were suggested in [52], which assist in alleviating the mechanical inputs researchers have to put in while designing real-time scene comprehension models. They introduce a novel search mechanism which explores through cell-level diversity and latency-based constraints.

Another context-focused research was published by Jiang *et. al.* [36] where they introduced context refinement and context integration modules for efficient scene segmentation. They employ dense semantic pyramids with image-level features, which encode contextual information while maintaining a large receptive field. An interesting technique of calculating spatial and contextual features was recently presented in [26], where the spatial details are evaluated in the forward path and the context is recorded in the backward flow. Light-weight feature pyramid encoding model was suggested in [56], which is an adaptation of the regular encoder-decoder architecture with depth-wise dilated convolutions. Multi-scale context aggregation was presented in yet another couple of approaches [89, 117], where [89] uses class boundary supervision to process certain relevant boundary information and [117] use optimized cascaded factorized ASPP module to balance the trade-offs between accuracy and execution speed. Orsic *et. al.* [67] developed a methodology which exploits light-weight upsampling and lateral connections with a residual network as the main recognition engine for real-time scene understanding. This particular algorithm is deemed as the current state-of-the-art network for real-time semantic segmentation on Cityscapes test dataset, whereas for the UAVid dataset, multi-scale dilation net [60] is the state-of-the-art.

3.2. Instance-aware Semantic Segmentation

Instance-aware semantic segmentation or instance segmentation is the process where the algorithm attempts to identify each **instance** of the different objects present in the image, instead of categorizing each pixel into a label class. For example, if there are five cars in an image, instead of labelling all cars with a single label, it will label each car separately. An illustration can be seen in **Fig.** 3.2.

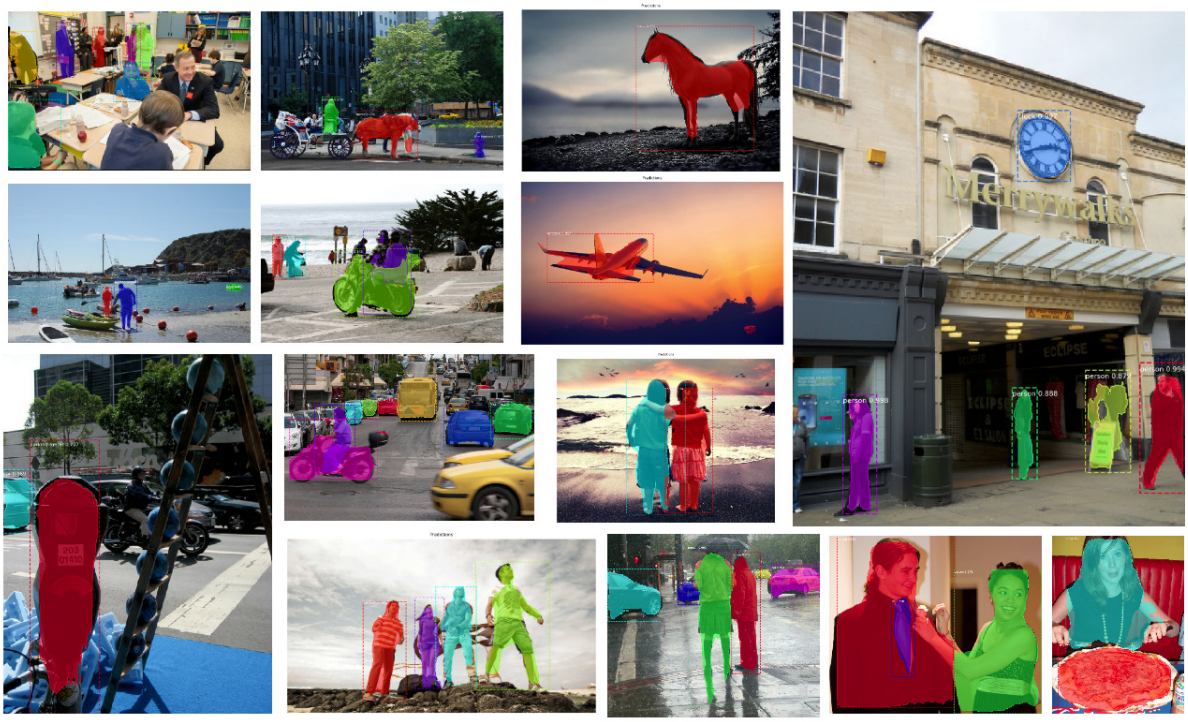


Figure 3.2: Instance-aware semantic segmentation outputs from Mask-RCNN [29]. Best viewed in color.

3.2.1. Mask-RCNN

Mask-RCNN [29] was introduced in 2017 and it became the glod standard for instace-aware semantic segmentation. This algorithm has two stages, similar to its predecessors Fast-RCNN [101] and Faster-RCNN [77], a region proposal network (RPN) and the final stage of classification and mask generation. The primary feature extractor for this network is either ResNet-50 or ResNet-101. RPN generates associated outputs for every anchor (set of predefined locations) and the mask generation and alignment is taken care by the **ROI Pooling** and **ROI Align** operations. In the end of the network, there is a final convolution layer that generates 28×28 sized features which represent the possible masks and they are later upscaled during inference to match the size of the ROI bounding box.

3.2.2. Other Techniques

Bai *et. al.* [5] introduced a simple intuitive technique for instance segmentation based on the classical watershed algorithm fused with deep learning, as opposed to other approaches to instance segmentation that employ complex techniques such as CRFs [42], RNNs [79] RPNs [29, 77]. Romera *et. al.* [79] suggested a sequential approach to finding objects in an image and their associated instances one at a time, using recurrent neural networks. Hybrid task cascading was suggested in [9], where the authors adopted a FCN to generate effective spatial details and interweave the cascaded refinement into a single join multi-stage processing block. Real-time instance segmentation on the MS-COCO dataset [53] was also suggested in [6], which achieved 35 FPS on a single Titan XP GPU.

3.3. Panoptic Segmentation

The concept of panoptic segmentation was introduced in 2019 by Kirillov *et. al.* [40], which basically unifies the concepts of both semantic and instance segmentation. This requires the generation of a coherent and complete scene segmentation head. An illustration is shown in **Fig.** 3.3.

Figure 3.3: Panoptic Segmentation illustration. Starting from left, semantic segmentation is shown, followed by instance-aware segmentation and the rightmost image shows the unification of both processes, called panoptic segmentation. Best viewed in color.

Since the inception of this concept, significant amount of efforts have been put in by several researchers across the globe. UPSNet [108] was proposed in 2019 to solve the panoptic segmentation challenge. They adopt a deformable convolution based segmentation strategy and Mask-RCNN style for instance aware segmentation, which together, solve these problems simultaneously. Another attention-guided network AUNet, was presented in [49] by Li *et. al.* where they proposed to distribute the object-level and pixel-level tasks into two different attention-guided structures that together again solve the panoptic segmentation challenge. Pyramid-based structures for panoptic segmentation were also studied extensively in [15, 40], whereas Li *et. al.* studied weakly and semi-supervised techniques for the same [46, 55].

4

A Preliminary Overview

Before we jump into the methodology and the architecture design of the proposed model, we would like to divert the attention of the reader towards an initial basic analysis of the several algorithms. Specifically, we would like to compare the different high-speed and high-accuracy models and discuss the different trade-offs presented across the literature more qualitatively. Consider for instance, Table 4.1. It can be seen clearly from the table that most of the accurate algorithms provide an execution speed of around 45 FPS.

4.1. Accuracy

Model	mIOU	Memory	MAdd	Flops	Params	FPS	FPS*
ICNet [119]	71.0	1094.47MB	162.43G	81.02G	28.30M	14.03	–
BiseNetV2 [113]	73.0	2784.99MB	207.64G	103.37G	3.65M	37.90	2.01
BiseNet [112]	74.7	1941.39MB	208.18G	103.72G	12.89M	47.20	2.42
ShelfNet [123]	74.8	1158.12MB	187.37G	93.69G	14.6M	44.37	2.59
SwiftNet [67]	75.5	1671.66MB	207.64G	103.37G	11.80M	45.40	2.61

Table 4.1: High-accuracy architectures. All computational expenses including FPS measured on a single RTX 2080Ti on full Cityscapes resolution (2048×1024). FPS* is measured on Jetson Xavier NX on full resolution again. – indicates the algorithm was too heavy to be executed on an embedded platform.

Now an average of 45 FPS is acceptable but it is to be noted that this value is obtained a powerful GPU (NVIDIA RTX 2080Ti in this case), which is unlikely to present in real-world robotic solutions. Once these speeds are computed on an embedded platform, we see that none of them reach beyond 4 FPS, which is slightly unacceptable. The average accuracy of the models in this table is around 74.5%.

4.2. Speed

Model	mIOU	Memory	MAdd	Flops	Params	FPS	FPS*
CGNet [106]	64.8	3134.91MB	55.01G	27.05G	0.5M	34.91	2.91
DABNet [43]	70.0	3287.50MB	82.83G	40.88G	0.76M	40.35	–
DFANet [44]	70.1	1778.09MB	30.68G	15.28G	2.19M	47.88	4.71
SINet [68]	68.2	672.00MB	2.99G	1.24G	0.12M	68.61	12.02
ESNet [59]	70.7	1176.29MB	66.81G	33.81G	1.81M	75.64	9.65
ContextNet [72]	66.1	1429.43MB	13.98G	6.74G	0.88M	118.65	10.49
Fast-SCNN [73]	68.4	1239.33MB	13.85G	6.72G	1.14M	128.97	11.49

Table 4.2: High-speed architectures. All computational expenses including FPS measured on a single RTX 2080Ti on full Cityscapes resolution (2048×1024). FPS* is measured on Jetson Xavier NX on full resolution again. – indicates the algorithm was too heavy to be executed on an embedded platform.

Now let us consider Table 4.2, which shows the relatively faster algorithms as compared to the previous ones. The picutre is now seemed to have reversed. Even though the fastest algorithm reaches around 120 FPS

[73], it shows a significant drop in accuracy, and it can be confirmed from this table as that as the speed tends to increase, the accuracy seems to drop considerably. The average speed of this table specifically, is around 60-70 FPS.

Let us compare the best algorithms from both the tables for a better analysis. Consider Table 4.3 for instance. We would like to divert the attention of the reader towards the difference in the overall mIOU score and FPS of both the algorithms. Also, the computational complexities of both the architectures are significantly far apart.

Model	mIOU	Memory	MAdd	Flops	Params	FPS	FPS*
SwiftNet [67]	**75.5**	1671.66MB	207.64G	103.37G	11.80M	45.40	2.61
Fast-SCNN [73]	68.4	1239.33MB	13.85G	6.72G	1.14M	**128.97**	11.49

Table 4.3: Best algorithms from both the above tables with highlighted advantages.

This huge gap between the two of the fastest and most accurate architectures is exactly what we intend to fulfill in this research. With this metric-based analysis now established, we can now move on to the methodology sections where we attempt to shorten this gap by means of context aggregation and cheap spatial detailing techniques.

4.3. Contributions

With respect to the above mentioned perspectives, our architecture has the following primary contributions to offer:

- Building upon the existing dual-branch architectures, we offer a cheap robust methodology to decouple the spatial and contextual feature extraction, where we design two branches, one for fast and effective spatial detailing and the other for dense context embedding.
- We introduce a plug-n-play context aggregation block (CAB) with Compact Asymmetric Position (CAPA) and Local Attention (LA) as the two sub-modules for deep global and local context aggregation.
- Our superior speed-accuracy trade-offs and effective spatial-contextual feature fusion allows us to outperform the previous state-of-the-arts for real-time semantic segmentation on Cityscapes and UAVid. Specifically, we achieve a mIOU score of 75.8% on Cityscapes and 63.5% on UAVid at 76 and 15 FPS respectively.

5

Network Architecture

We begin this chapter by introducing and explaining every component of our proposed Context Aggregated Bilateral Semantic Segmentation Network (CABiNet) in detail. An overall visualization of the design can be seen in **Fig.** 5.1.

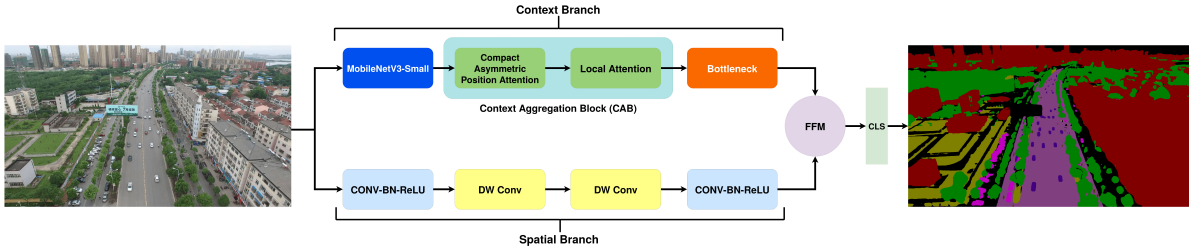


Figure 5.1: Architecture of CABiNet with cheap spatial branch and deep context branch. FFM and CLS stand for feature fusion module and classifier respectively. Input image is shown on the extreme left and on the extreme right CABiNet's prediction output for the input RGB image.

5.1. Spatial Branch

In order to encode sufficient spatial information, multiple existing approaches [71], [99], [12], [11] have employed the usage of dilated convolutions, while others attempt to capture large receptive fields with either pyramid pooling or large-sized kernels [71], [12], [118]. These methodologies indicate that sufficient receptive fields and effective spatial information encoding, could be crucial for accurate semantic segmentation. It is however, difficult to satisfy both the requirements in parallel, especially while designing real-time segmentation architectures. Conventional real-time designs usually either downsize the image to a smaller resolution [119] or use a lightweight reduction model [4], [69] for speeding up the overall architecture. Downsizing the image however, incurs loss in the spatial information and light-weight models tend to damage the receptive fields because of the incessant channel pruning. This problem was addressed in [112], [113], but at the cost of significant increase in the computations, thereby imparting a lower execution speed on mobile and embedded platforms. Based upon these observations, we propose a shallow branch that encodes rich spatial information and maintains an adequate receptive field, while maintaining a significantly low computational cost. We design this branch to extract the low-level information, which is the rich spatial content in a full-resolution image. This branch therefore requires a rich channel capacity, and since it focuses only on the low-level details, this branch has a shallow structure with small strides. Specifically, this path has four layers, where the first layer is a convolutional layer (large kernel size) followed by batch-normalization and ReLU, followed by two depth-wise convolutional layers. A strategic use of depth-wise convolutions results in the same outcomes as that of conventional convolutions, but with reduced computations, and the marginal loss in features can be compensated by enlarging the output channels. Finally, the last layer is another convolutional layer with kernel size of 1. Strides for the first three layers are fixed at 2, whereas the last layer has a unit stride. This branch, hence generates an output that is $(\frac{1}{8})^{th}$ of the input resolution, thereby maintaining

the required spatial information with a significant reduction in computations. A detailed graphic in **Fig.** 5.1 shows the overall structure of the shallow branch.

5.2. Context Branch

As already established previously, detailed spatial information coupled with adequate receptive field, significantly affects semantic segmentation accuracy. While the shallow branch takes care of the spatial details, we design a new attention branch, with light-weight compact asymmetric position and local attention (CAPA+LA) for providing a sufficient receptive field and capturing both global and local context. We use a pretrained MobileNetV3-Small [31] as the lightweight feature extractor in this branch, which can downsample the input images effectively and efficiently to provide rich high level semantic features. These features are however unrefined, and hence, need to be passed on to a refinement stage, which in this case is the context aggregation block.

5.3. MobileNetV3-Small

Interesting thing about mobile networks such as [19, 31, 32, 61, 84] is that they are built upon efficient building blocks such as depth-wise convolutions, depth-wise separable convolutions, atrous convolutions etc. This concept imparts an acceptable inference speed on mobile platforms while maintaining the required accuracy. For instance, depth-wise separable convolutions were introduced in MobileNetV1 [32].

Similarly, MobileNetV2 [84] introduced the concept of linear bottlenecks and inverted residual structure for enhancement of individual layer structures. This structure is defined by a series combination of 1×1 expansion convolution, depth-wise convolutions and a final 1×1 projection layer. If the input has the same number of channels as the output, they are connected with a residual link. It is noteworthy that such a structure maintains a dense representation of both the input and output, while internally expanding to a higher-dimensional feature space. Building upon MobileNetV2, MnasNet [93] introduced efficient attention blocks in the bottleneck structures with the help of squeeze-and-excitation modules [33], which were placed after the depth-wise convolutions, such that maximum receptive field is available to the attention blocks for feature extraction. The squeeze block suggested in [33] is primarily meant for global information embedding whereas the excitation block takes care of the adaptive re-calibration of assimilated features from the squeeze operations.

Later in 2019, MobileNetV3 [31] was introduced which uses a mixture of layers suggested in MobileNetV2 and MnasNet, to construct the most effective and efficient neural network for mobile applications. Modified `swish` non-linear functions were used to improve the performance of layers, along-with hard sigmoid for squeeze-and-excitation modules. The exact specifications of the backbone are mentioned in Table 5.1, with the associated notations. The final feature vector after the backbone is of size $64 \times 32 \times 576$.

Input	Operator	Exp. Size	C	SE	NL	s
2048×1024×3	Conv2D, 3×3	-	16	✗	HS	2
1024×512×16	BNeck, 3×3	16	16	✓	RE	2
512×256×16	BNeck, 3×3	72	24	✗	RE	2
256×128×24	BNeck, 3×3	88	24	✗	RE	1
256×128×24	BNeck, 5×5	96	40	✓	HS	2
128×64×40	BNeck, 5×5	240	40	✓	HS	1
128×64×40	BNeck, 5×5	240	40	✓	HS	1
128×64×40	BNeck, 5×5	120	48	✓	HS	1
128×64×48	BNeck, 5×5	144	48	✓	HS	1
128×64×48	BNeck, 5×5	288	96	✓	HS	2
64×32×96	BNeck, 5×5	576	96	✓	HS	1
64×32×96	BNeck, 5×5	576	96	✓	HS	1
64×32×96	Conv2D, 1×1	-	576	✓	HS	1

Table 5.1: MobileNetV3-Small specifications for our use-case. The **Input** column shows the size of input vector to the associated layer in $W \times H \times N$, where W, H and N are width, height and the number of channels in the tensor respectively. The expansion size is mentioned in the **Exp. Size** column, whereas the **C** column tells about the output number of channels after the vector is passed through the associated layer. ✗and ✓indicate the absence and presence of squeeze-and-excite modules in the associated block respectively. **NL** column indicates what kind of non-linearity is present in the block whether Hard-squish (HS) or ReLU (RE). The final column **s** indicates the stride of the block.

The hard-swish non-linearity is defined as:

$$h-swish[x] = x\frac{ReLU6(x+3)}{6} \tag{5.1}$$

which significantly increases the accuracy of deep neural networks in image classification tasks [18, 30].

Interesting aspect of MobileNetV3 series is that these models were not designed by any humans. Instead, the authors used block-wise platform-aware neural architectural search (NAS) [93] for finding the global structures. Then they performed a layer-wise search for searching the optimal number of filters using NetAdapt [110]. All proposals that were generated with NetAdapt, were filtered based on a single criteria that maximizes the ratio $\frac{\Delta accuracy}{|\Delta latency|}$. All of the above factors combined result in an accurate and low-power image classifier that could potentially be used a backbone for many computer vision tasks.

5.4. Context Aggregation Block

Theoretically, when an image is passed through the forward flow of a feature extractor like ResNet [27] or MobileNetV3 [31], the output is a feature vector of certain size ($W \times H$) and has certain number of channels (N), which depends upon the output layers of the extractor. These channels ideally, are (N) different interpretations of the input image, which implies that each and every interpretation contains some or the other information about the contents of the original image. Intuitively, it is very likely that each position in a particular channel has some sort of mapping or a link in another channel. Furthermore, positions in a certain channel could also possibly have local connectivity within the same channel. These long-range and local dependencies could be crucial for accurate semantic segmentation. The key ideology of context aggregation block is therefore, is to capture effectively and efficiently such inter-channel and intra-channel mappings. Couple of previous works have explored this concept but at the cost of significant computations, which becomes the focal point of this research.

Position attention module (PAM) was introduced in [23], which is potent enough to capture long-range dependencies crucial for accurate semantic segmentation. However, the module is computationally expensive and requires significant GPU memory for execution. In this section, we review the shortcomings the components of PAM block and attempt to overcome them.

5.4.1. Revisiting Position Attention Module

The following sections are inspired from [23, 48, 122]. The solutions presented in [48, 122] have been employed as a part of this research to refine the features obtained from the previous backbone (MobileNetV3-Small) stage.

The original position attention module is shown in the upper part of **Fig.** 5.2, where the output from the previous backbone stage is fed to three parallel convolutional layers to generate new embeddings. After the embeddings have been generated similarity matrices are calculated using matrix multiplications, followed by a Softmax normalization process. This output contains the semantic cues for every position in the input feature vector. The entire pipeline can be seen in **Fig.** 5.2 on top.

5.4.2. Compact Asymmetric Position Attention (CAPA) Module

A careful observation of the pipeline in **Fig.** 5.2, indicates that there could be two possible limitations to the position attention module suggested in [23]. Firstly, the matrix multiplications of the `Key` and `Value` convolutions followed the by the next multiplication process after the softmax activation stage, increase the time complexity, as these computations are performed on relatively large matrices. Secondly, the original design proposes to extract the contextual information directly from the outputs of the backbone which has a very large number of channels, thereby increasing the required number of parameters. One possible solution to the first identified challenge was suggested in [122] which assists in the reduction of computational expenses in such self-attention modules. However, for our real-time application case, we believe that the complexities could be reduced even further by adjusting the number of parameters and computations in the convolutional layers of the self-attention module. So first, we discuss the solution presented in [122] in detail, which we employ in this research and then we discuss the additional improvement in the convolutional layers, which tackles the second identified challenge.

For our real-time scene understanding architecture, the input to the context aggregation block has a size of $64 \times 32 = 2048$. Therefore, it can be said that the simple, yet large matrix multiplication is the basic cause of increased computations in the PA block. The limiting factor in the above step is the number A (**Fig.** 5.2), and

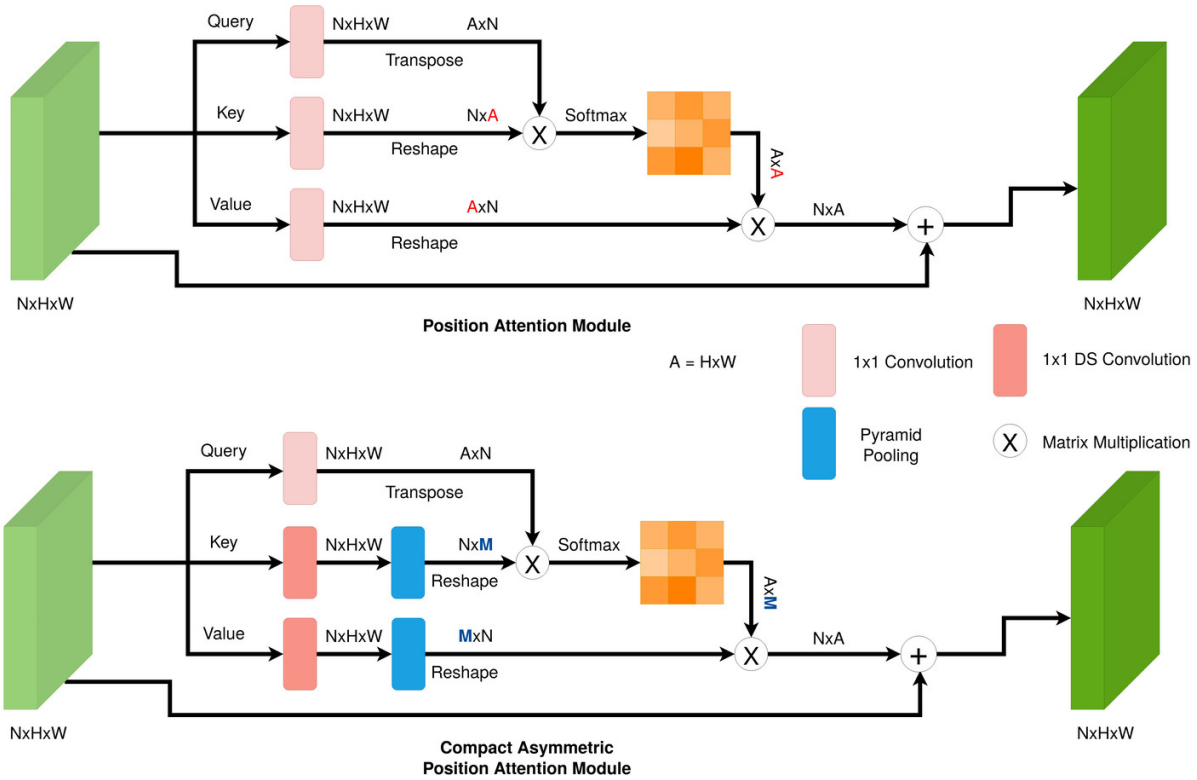


Figure 5.2: Position attention module (top) and the compact asymmetric position attention module (bottom). Here, $A = W \times H$. Our CAPA module leverages the benefits of spatial pyramid pooling and depth-wise separable convolutions. Image best viewed in color.

if it were changed to a smaller value M, (where $M << A$) might help in alleviating some of the computations. Although, changes have to be made in such a way that the output size of the vector remains unchanged.

Hence, we adopt the suggestion of [122] of employing spatial pyramid pooling modules [118] after the convolutional layers in the position attention module to effectively reduce the size of the feature vectors for easier computations. Therefore, instead of feeding all the spatial points to the to multiplication process, it would be advisable to sample the points and feed only certain representative points to the process.

This is precisely what the PSP-module [118] does. From previous works [28, 72, 118] we know that the spatial pyramid pooling module [118] has been proven to be effective in capturing multi-scale representations. Furthermore, this pooling module is free from parameters and has high efficacy. Therefore, for our real-time application, an appropriated choice would be to employ this module for sampling the `Key` and `Value` vector representations. We use four adaptive maximum pooling at four scales to reduce the amount of computations in the PA block (similar to what was suggested in [122]) and then the four pooling results are flattened and concatenated to serve as the input to the next layer.

For our experiments, we set the number of scales at four, as was also suggested by [122]. The number of sparse representations can be formulated as:

$$S = \sum_{n \in 1,3,5,8} n^2 = 110 \tag{5.2}$$

thereby reducing the complexity to $\mathcal{O}(\hat{N}AM)$, which is much lower than $\mathcal{O}(\hat{N}A^2)$. Specifically, for our input to the PA block of $64 \times 32 = 2048$, this asymmetric multiplication saves us $\frac{64 \times 32}{110} \approx 18$ times the computation cost. Furthermore, the feature statistics captured by the pooling module are sufficient to provide cues about the global scene semantics.

So far we have discussed the solution presented in [122]. But as mentioned earlier, there is still a scope of improvement in this block with respect to the convolutional layers present, considering a real-time application at hand. This block employs three 1×1 convolution layers, which results in a relatively larger number of parameters. This might not have a direct influence on the overall execution speed, but a neural design with lesser parameters indicates the effectiveness and efficiency of the model. The idea proposed in this approach

is simple [24] and is shown in **Fig.** 5.3.

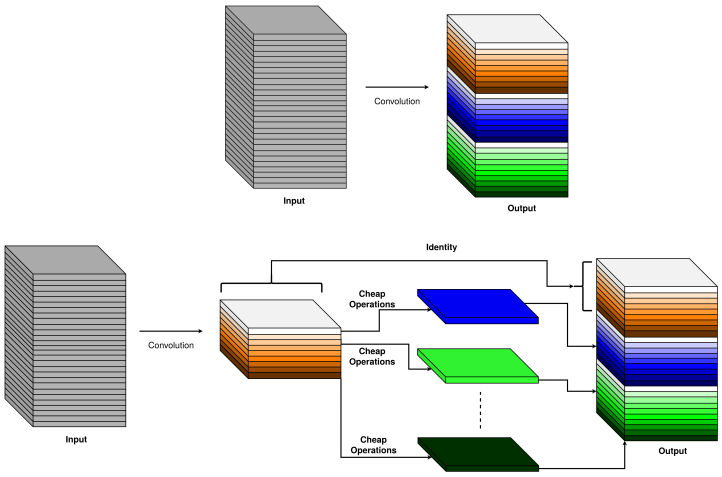


Figure 5.3: Cheap operations concept. Best viewed in color.

Regular convolution layers have learnable filters that convolve on the input feature vector. The repetitive convolution strategy results results in a certain number of parameters based on the kernel size, input size etc. and also in redundant features. Basically, if there are N output channels, it is unlikely that all the channels contain absolutely dissimilar information, which means that some channels (if not all) could literally be duplicated and need not have repeated convolutions. This technique is called cheap operations, where a convolution layer is applied first to generate a smaller number of channels, and this collection of features is then passed to a set of cheap linear operations, which result in exactly the same output as compared to a full convolution, but with reduced parameters and computations. Cheap linear operations are a generic strategy and in this research the implementation is done with depth-wise separable convolutions. A quantitative breakdown of this procedure is later presented in the ablation studies.

5.4.3. Local Attention

In the previous section, we calculate the global statistics for every group which are later multiplied back to features within. The aspect to be noted here is that the windows in which the statistics are calculated are relatively large, and hence there is a possibility that the statistical cues could be biased to towards the larger patterns as there are more samples within, which can further cause and over-smoothing of the smaller patterns. This over-smoothing should be avoided to create an accurate semantic segmentation algorithm.

In this regard, a local attention (LA) module was proposed in [48] to adaptively use the features, considering patterns at every position encoded by the previous global attention block. We directly employ the above suggested module in this research without any additional modifications. Our ablation studies indicate that this module is efficient and fast and hence requires no additional improvements. Fundamentally, the LA block predicts local weights by re-calculating the spatial extent, which is primarily targeted to avoid coarse feature representation issues, which were present in the previous CAPA module. Here, the predicted local weights add a point-wise trade-off between the global information and local context. CAPA module lacks in details which is complemented by the use of LA block, thereby generating a more fine-grained representation of the input features. The local attention block is hence, modelled as a set of three depth-wise convolutional layers, which allows for fine-tuning the feature representations from the previous CAPA module.

5.4.4. Plug-n-Play Concept

Both the attention modules employed in this research were suggested in different literature and were, hence utilized in different sections of the architectures. However, the combination of these two modules allow for the creation of a linear sub-structure of self-attention mechanism. Since, the local attention module operates on the feature vectors coming from the CAPA module, this linearity in approach allows this overall block to be used as a plug-n-play structure for other multi-branch architectures for semantic segmentation. Details have been presented in the ablation studies section.

5.5. Downsampling Bottleneck

Feature representations while context extraction could increase significantly, thereby increasing the inference time and introducing redundancy in semantics. Hence, keeping a check on the number of channels becomes important while designing the attention branch. In deep learning, a bottleneck is a neural block that has lesser input channels as compared to the previous layers. This block is usually added to deep structures to assist in reducing the number of features (channels) to best fit in the GPU space available. This further assists in avoiding over-fitting or exploding weights. In our research, we use a downsampling bottleneck with 1×1 convolutions with reduced channels to reduce the number of feature representations in the attention branch, thereby reducing the number of parameters required to learn effectively. Furthermore, this bottleneck generates a class-wise discriminated output representation which is later used to directly supervise the learning of this particular branch. In common terms this is called deep supervision of internal structures of a model. Since this branch generates the crucial low-level features a doubly deep supervision ensures the appropriate extraction of features.

5.6. Feature Fusion

It is to be noted that the features extracted from both the branches are in different levels of representation, i.e. a higher level and a lower level. Hence, a simple addition of both the feature vectors is very unlikely to produce the desirable results. [72] and [73] follow the addition-of-features strategy in order to save computations. This in turn, tends to have significant impacts on the final accuracy of the model. Therefore in this research, we implement a feature fusion technique as suggested in [112], with certain adaptations.

In order to fully utilize the vector representations from both the branches, we concatenate both the features first, followed by a downsampling bottleneck. The initial idea of fusion suggested in [112] incurs a great deal of computational expenses and the reason for this the fact that the concatenated feature vector is large in all the three dimensions. Hence, computations on this extended representation is the primary reason for the slow performance of this particular module. Adding a downsampling bottleneck reduces the efforts of the later computations in this module by significant amount, without causing damage to the overall accuracy.

This if followed by a depth-wise convolution, which again assists in retaining the representations as conventional convolution, but with reduced computations. This is followed by a batch-normalization step to equalize the different feature scales. In the next step, we apply a reduced channel attention block to further enhance the vector representations, the output of which is multiplied with the initial features. A detailed schematic is shown in **Fig.** 5.4

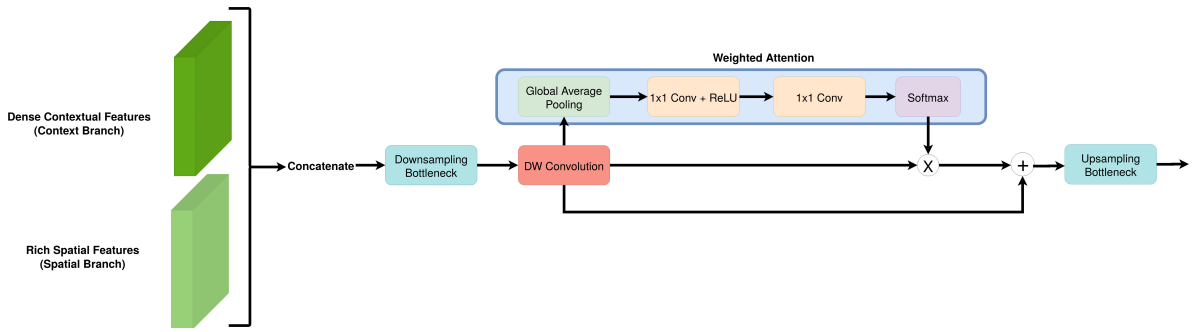


Figure 5.4: Feature fusion module. Best viewed in color.

Finally, we upsample the features again with an upsampling bottleneck, such that the retained features have the exact same representation as was described in [112].

5.7. Output Classifier

The output classifier block generates the final class-wise discriminated outputs for predictions. Empirically, we observe that the addition of a smaller number of layers boosts the performance of the fusion module. A possible reason for this could be that till the fusion module vector representations already have discrete separation of features within the channel representations and the only aspect left is implementing a class-wise separation. Hence, for a simple class-wise separation, multiple layers become unnecessary.

With this concept in mind, we utilize only two layers in the final classifier, one depth-wise separable convolution and one point-wise convolution as the final output layer. Since we use SGD as our optimizer, we use Softmax activation instead of Sigmoid [72, 73].

5.8. Loss Functions

Several loss functions have been proposed and used for scene understanding tasks. Datasets like Cityscapes, CamVid and UAVid contain a number of easy examples (over-weighted classes) and a relatively smaller number of hard examples. In order to create a suitable balance between the two, we use the regular weighted Cross-entropy loss given by Eq. 5.13-14. Apart from just monitoring the overall output of CABiNet, we use two additional auxiliary loss functions, one that monitors the output of the attention branch and one for the attention fusion module. These two auxiliary loss functions provide deep supervision of the two modules, thereby making sure that the right feature representations are learnt. The value of α is set to 1

$$loss = \frac{1}{N}\sum_{i} L_i = \frac{1}{N}\sum_{i} -\log\left(\frac{e^{p_i}}{\sum_j e^{p_j}}\right) \tag{5.3}$$

Here, p is the final output of the network (prediction).

$$L(X;W) = l_p(X;W) + \alpha \sum_{i=2}^{K} l_i\,(X_i;W) \tag{5.4}$$

where, l_p is the principal loss of our network, X_i is final feature output from stage i and l_i is the corresponding loss for that stage. K is three in this research and L represents the joint loss of the function. Utilizing a joint loss makes it easier to optimize the model, hence the auxiliary losses are only employed during the training stage.

5.9. Implementation Details

5.9.1. Training Objectives

Following [112], our model has a total of three supervisions, two for the context branch and one for the overall architecture. Mathematically, we express the loss functions as:

$$L_{final} = L_{output} + L_{C1} + L_{C2} \tag{5.5}$$

where, L_{C1} and L_{C2} are the two auxiliary losses for the context branch. We use regular cross entropy for the final loss and perform online hard example mining (OHEM) [88] for the auxiliary losses. We do not use weighing parameters to control the influence of the auxiliary loss functions. Instead we incorporate them completely into the overall loss calculation.

5.9.2. Training Settings

This research is based on an open-source deep learning framework PyTorch 1.4 [70], commonly used for semantic segmentation models. The backbone used in the attention branch is MobileNetV3-Small [31] with unit width. For optimizing the network, we use Stochastic Gradient Descent (SGD) [39] and set the initial learning rate as $1e^{-4}$ for Cityscapes and $5e^{-5}$ for UAVid. We employ the poly-learning rate strategy, where during training, the learning rate is multiplied with $1-(\frac{iter}{max\_iter})^{power}$, with power being equal to 0.9. For Cityscapes we randomly crop patches of [1024, 1024] from the original input images during training. For UAVid we adopt a slightly different technique as compared to [60]. The original author proposed to split the UAVid images into 9 overlapping regions of (1024×2048) during training and inference. Instead, we recommend to split the image into 4 equal quarters of (1920×1080). As a result, we do not have to average out the results of the overlapping sections, thereby improving the overall prediction accuracy at the cost of slightly slower inference. It is tedious to train on the full resolution of UAVid, as the image size is too large and requires a significant amount of GPU memory to store the intermediate features. We use data augmentation techniques like random horizontal flips, random scaling and color jitter for both the datasets. Scales range from (0.75, 1.0, 1.5, 1.75, 2.0). Batch-sizes are set at 6 for Cityscapes and 3 for UAVid and since we train and evaluate on a single GPU, we do not employ cross-GPU synchronized batch normalization. Furthermore, training iterations are set at 160k for Cityscapes and 240k for UAVid. All the above mentioned experiments are conducted on a single NVIDIA RTX 2080Ti, with PyTorch 1.4 and CUDA 10.2

5.9.3. Inference Settings

As a common notion, we do not apply any inference tricks such as multi-scale and left-right flip testing. These tricks tend to make the overall inference process slower, although provided mIOUs are significant. Ablation studies are done in single scale as well by feeding in images at full resolution. For inference, we use a single NVIDIA RTX 2080Ti, with PyTorch 1.4 and CUDA 10.2 and a Jetson Xavier NX with the same software setup. During inference, no other programs occupy the GPU on either of the two platforms.

6

Experimental Setup and Results

In this chapter, we demonstrate the effectiveness and the efficiency of our design by conducting multiple experiments on different platforms. Firstly, we compare our design with numerous other real-time semantic segmentation architectures on 2 different publicly available benchmarks. Secondly, we also perform multiple ablation studies for feature fusion, bottleneck and context embedding modules to effectively study every component of our architecture.

6.1. Datasets

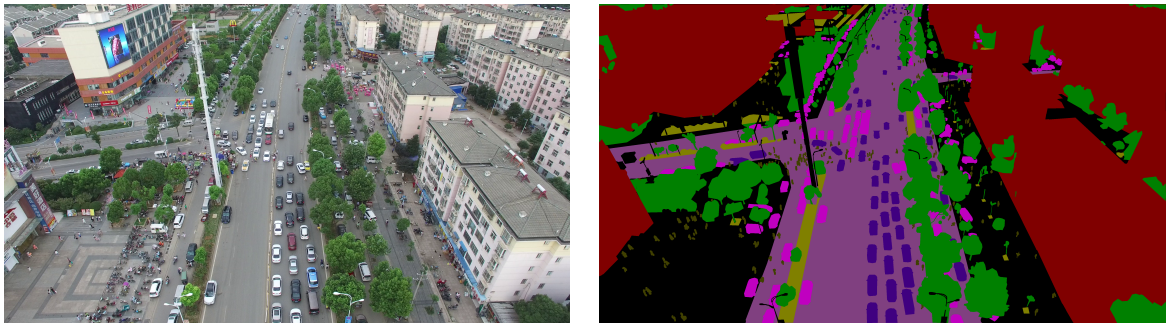
Robustness of an algorithm can be demonstrated by benchmarking it on multiple, possibly unrelated datasets. With this ideology in mind, we benchmark our system on Cityscapes [16] and UAVid [60] datasets. We begin by introducing both the datasets briefly.

Cityscapes is an urban-scene understanding dataset which was publicly released in CVPR 2016 [16]. The dataset contains a total of 5000 images (fine-grained) out of which, 2975 are for training, 500 for validation and the remaining 1525 for testing. The dataset also contains additional 20K coarsely annotated images, but we do not use them in this research. The image size for this dataset is 1024×2048, collected from across 50 different cities. The densely annotated data contains 35 classes, out of which 19 are used for urban scene understanding. Usage of coarse training data could be employed but usually it is not a practise for real-time low-latency architectures. An example is shown in **Fig.** 6.1.

(a) RGB Image **(b)** Ground Truth for Semantic Segmentation

Figure 6.1: Single training sample from the Cityscapes dataset [16]

UAVid [60] is another publicly available dataset for urban scene understanding that we use in this research, but unlike Cityscapes, this dataset is collected from unmanned aerial vehicles (UAVs). An example is shown in **Fig.** 6.2 for comparison. This dataset introduces large scale variations, making it a very competitive and a tough benchmark. The dataset contains 300 densely labeled images into a total of 8 classes and 100 images are available for testing. The resolution of the images is 4K i.e. 3840×2160, which coupled with the small size of the dataset, makes it a very challenging benchmark. The reason why the above two datasets were chosen is that they provide for a real-time analysis and benchmark for urban-scene understanding from different viewpoints, thereby effectively establishing the robustness of our architecture.



(a) RGB Image **(b)** Ground Truth for Semantic Segmentation

Figure 6.2: Single training sample from the UAVid dataset [60]

6.2. Evaluation Metrics

For evaluation, we use the standard mean of class-wise intersection over union (mIOU), memory footprint (in MB/GB), GLOP count (floating point operations) and the overall execution speed (Frames per second). For calculating the computational expenses we adopt the same strategies as mentioned in [67, 112].

6.3. Results

In this section, we compare the effectiveness of the attention modules (PAM-CAM) suggested in [23] with the proposed attention modules, namely Compact Asymmetric Position (CAP) and Local Attention (LA). The benchmark is drawn out based on GFLOPs count (Million), GPU run-time (ms), number of parameters and impacts on overall mIOU improvements over baseline, which can be seen in Table 6.1. Our proposed attention block receives an input tensor of size 64×32 during training and inference for Cityscapes dataset. We therefore, compare the two blocks individually and the overall combinations as well in a fixed testing environment, such that the both have identical testing conditions. We experiment with both on a single RTX 2080Ti with CUDA 10.2, with no other programs occupying the GPU. The major difference between the proposed attention modules and the original modules [23] is the presence of spatial pyramid pooling and depth-wise separable convolutions. Other processes are identical.

Module	GFLOPs	Params (K)	Runtime (ms)	mIOU
ARM + ARM [112]	3.63	311	**3.24**	74.8
PAM + CAM [23]	1.01	82.24	17.62	76.3
GA + LA [48]	1.01	65.34	14.28	76.1
APNB + AFNB [122]	0.82	42.24	8.35	76.4
CAPAM + LAM (Ours)	**0.024**	**12.29**	3.48	**76.6**

Table 6.1: As compared to original position attention module proposed in [23], our design has much less computational complexity, lesser parameters and almost 5 times faster. Another attention refinement method was suggested in [112], which has a slightly lower runtime than ours but has lesser improvements on the overall mIOU. It is to be noted that [112] use two of such proposed modules (AR) in their actual architecture, which doubles all the above numbers. Since we only have single context aggregation stage, we offer much less computational overhead. Our attention fusion technique outperforms all the previously suggested methodologies in almost every aspect.

Apart from simple block-wise comparison, we also compare our overall architecture with the other low-latency semantic segmentation models on the Cityscapes dataset as can be seen in Table 6.2. Our superior speed-accuracy trade-offs outperform the current state-of-the-art algorithm in terms of computational expenses and execution speed while maintaining a slightly better overall mIOU score.

6.3.1. Cityscapes

A detailed comparison between our method and other architectures has been provided in Table 6.2, based upon the GPU memory footprint, MAdd/GLOPs count, execution speed (RTX 2080Ti and Jetson Xavier NX) and the overall mIOU score on validation and testing sets. We train our model directly on the training set of Cityscapes, without incorporating the validation set or the coarsely annotated data. As it can bee seen from the table, our model outperforms the previous SOTA methods for real-time scene understanding and achieves 75.9% mIOU score. We also provide a qualitative analysis with the SwiftNet [67] in **Fig.** 6.3 and **Fig.** 6.4 and establish the superiority of our model in terms of detecting smaller objects like poles, traffic signs etc. [67, 123] seem to have certain local inconsistencies on larger objects especially buses, as well as the smaller objects, whereas our model does not. Thanks to the efficient global and local context aggregation, our model does not suffer from such local or global inconsistencies.

Model	**mIOU**		**Memory**	**MAdd**	**Flops**	**Params**	**FPS**	**FPS***
	val	**test**						
CGNet [106]	–	64.8	3134.91MB	55.01G	27.05G	0.5M	34.91	2.91
ContextNet [72]	–	66.1	1429.43MB	13.98G	6.74G	0.88M	118.65	10.49
FPENet [56]	69.5	68.0	–	–	–	–	–	–
SINet [68]	69.4	68.2	672.00MB	2.99G	1.24G	0.12M	68.61	12.02
Fast-SCNN [73]	–	68.4	1239.33MB	13.85G	6.72G	1.14M	128.97	11.49
FarSee-Net [117]	69.8	68.4	–	–	–	–	–	–
ERFNet [78]	71.5	69.7	3244.00MB	213.88G	102.61G	2.07M	20.46	✗
DABNet [43]	70.1	70.0	3287.50MB	82.83G	40.88G	0.76M	40.35	✗
DFANet [44]	71.3	70.1	1778.09MB	30.68G	15.28G	2.19M	47.88	4.71
LedNet [105]	71.5	70.6	3031.75MB	90.71G	45.84G	0.93M	24.72	✗
ESNet [59]	–	70.7	1176.29MB	66.81G	33.81G	1.81M	55.65	4.65
ICNet [119]	72.5	71.0	1094.47MB	162.43G	81.02G	28.30M	14.03	✗
GAS [52]	73.5	71.8	–	–	–	–	–	–
BiSeNetV2 [113]	74.2	73.0	2784.99MB	207.64G	103.37G	3.65M	37.90	2.01
FasterSeg [14]	–	73.1	–	–	–	–	–	–
BiSeNet [112]	74.8	74.7	1941.39MB	208.18G	103.72G	12.89M	47.20	2.42
ShelfNet [123]	75.2	74.8	1158.12MB	187.37G	93.69G	14.6M	44.37	2.59
SwiftNet [67]	75.4	75.5	1671.66MB	207.64G	103.37G	11.80M	45.40	2.61
CABiNet (Ours)	76.6	75.9	1256.18MB	24.37G	12.03G	2.64M	76.50	8.21

Table 6.2: Computational expenses and run-time measurements for all the models have been done on a single RTX 2080Ti, on an input resolution of 1024×2048. The architectures mentioned in the table above have mostly computed their GFLOPS on different resolutions, thereby making the comparisons unfair. We recompute the MAdd and FLOPS on a common resolution from the official implementations to provide a better understanding of the architecture complexities. – indicates that the corresponding values could not be confirmed at the time of writing this report. ✗indicates that the execution of the corresponding models at 1024×2048 resolution resulted in < 1 FPS. Please note that the execution speeds for [112, 113] are observed to be lower than what were reported originally as the authors used TensorRT [95] optimization to enhance the inference speeds of their models. We report all execution speeds of the original models without any such modifications.

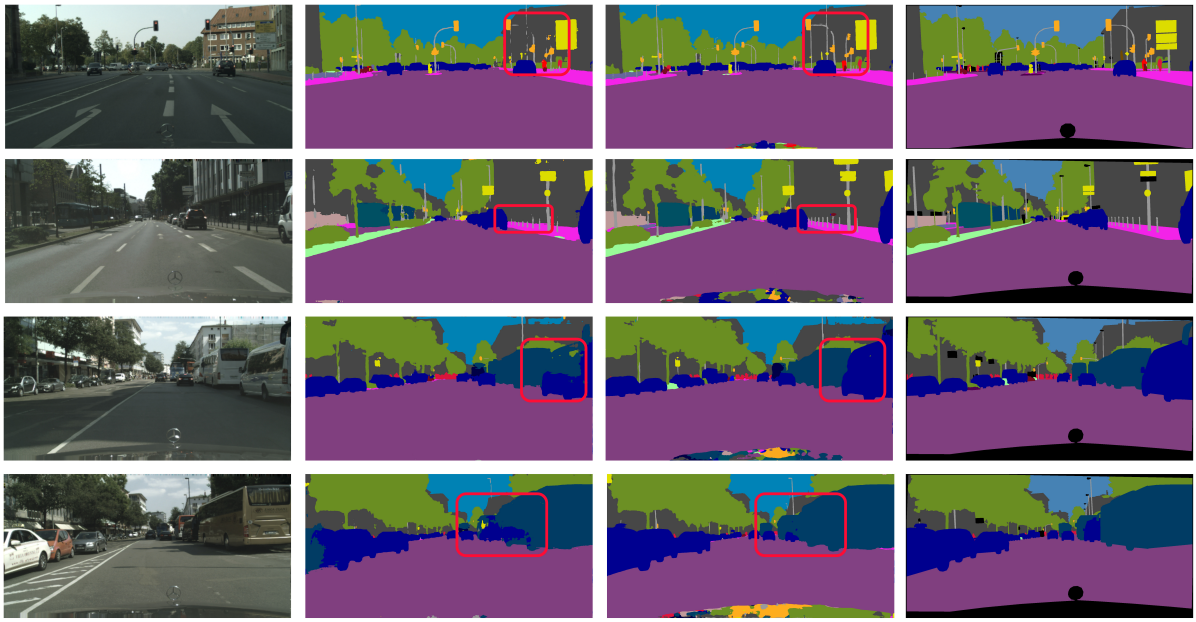


Figure 6.3: Comparative segmentation results on the Cityscapes validation set. From left, the first column consists of the input RGB images. Second column indicates the prediction results of the SOTA [67], whereas the third column shows the predictions from our architecture and the red boxes show the improvements we offer over the current state-of-the-art. Last column comprises of the ground truths. Best viewed in color.

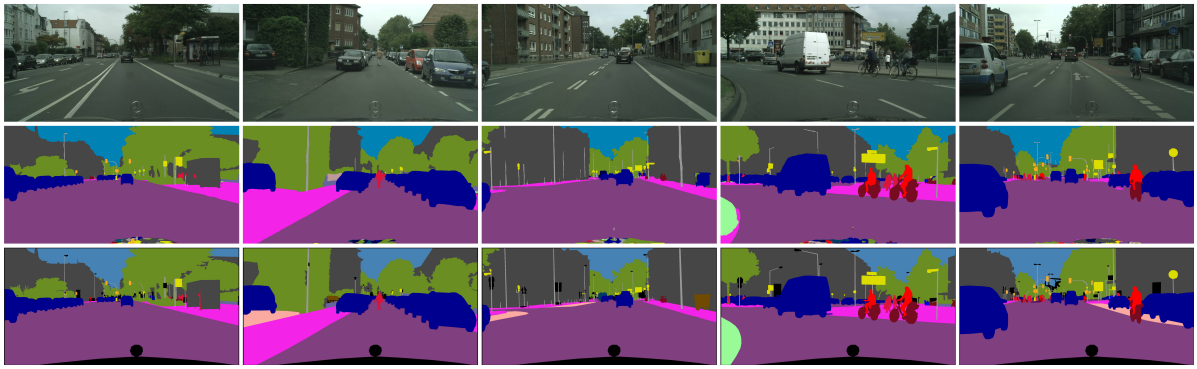


Figure 6.4: More segmentation results on the Cityscapes validation set. First row consists of the input RGB images. Second row contains the predictions from our architecture and the third row shows the ground truths of the input images. Bext viewed in color.

6.3.2. UAVid

As previously established, UAVid [60] is a challenging benchmark due to the large resolution of images and extreme complexities in the scenes. Hera again, we do not use the validation data during training. The validation data is only used for simple hyperparamter tuning post training. Testing is done on the UAVid official servers by submitting the predictions from the different models. Interestingly, our efficient context aggregation scheme outperforms the previous state-of-the-art on UAVid, MS-Dilation Net [60], by a large margin of 14%, all the while maintaining an execution speed of 15 FPS (on 4K resolution) on a single RTX 2080Ti. Furthermore, we also benchmark other multi-branch architectures on UAVid, to support the claim of split-and-extract concept of spatial and context details. We however, do not benchmark with this dataset on a Xavier NX as the SOM is only so powerful to run a full-scale Cityscapes prediction. A detailed quantitative description can be seen in Table 6.3.

As it can be seen from **Fig.** 6.5 and **Fig.** 6.6, our model does not suffer from local or global inconsistencies, thereby effectively capturing the cues to scene semantics. Previous SOTA models had either 0% or less than 5% detection rates for smaller objects like humans in UAVid. Our model has a detection rate of almost 20% on humans, and furthermore, the segregation between clutter and building is distinctively clear as can be seen in the figure. Local feature incorporation failure results in a mix-up in clutter and building, which

Model	Building	Tree	Clutter	Road	Vegetation	Static Car	Moving Car	Human	mIOU	FPS
FCN-8s+PRT	77.4	72.7	44.0	63.8	45.0	19.1	49.5	0.6	46.5	–
Dilation-Net+PRT	79.8	73.6	44.5	64.4	44.6	24.1	53.6	0.0	48.1	–
U-Net+PRT	77.5	73.3	44.8	64.2	42.3	25.8	**57.8**	0.0	48.2	–
MS-Dilation+PRT	79.7	74.6	44.9	**65.9**	46.1	21.8	57.2	8.0	49.8	–
Fast-SCNN [73]	75.7	71.5	44.2	61.6	43.4	19.5	51.6	0.0	45.9	**33.84**
ShelfNet [123]	76.9	73.2	44.1	61.4	43.4	21.0	52.6	3.6	47.0	9.65
SwiftNet [67]	85.3	78.2	64.1	61.5	76.4	62.1	51.1	15.7	61.1	11.84
BiSeNet [112]	85.7	78.3	64.7	61.1	77.3	63.4	48.6	17.5	61.5	11.08
CABiNet (Ours)	**86.6**	**79.3**	**66.0**	62.1	**78.1**	**68.3**	47.8	**19.9**	**63.5**	15.14

Table 6.3: Quantitative results on the UAVid test dataset from the official server. Please note that for training ShelfNet [123], we adopt the same strategy mentioned in [60], as the architecture functions with only fixed input-sizes which are multiples of 256. All models were trained on a batch-size of 3, for 50% larger iterations than were originally proposed in each. – indicates that the FPS of the algorithm could not be confirmed.

could theoretically have similar color and texture. The elevation however, makes the difference, which is captured effectively by our CAPLA block, thereby reducing the false positives. It is noteworthy that our algorithm performs better than MS-Dilation-Net [60], even though it is based on a FCN-8s structure.

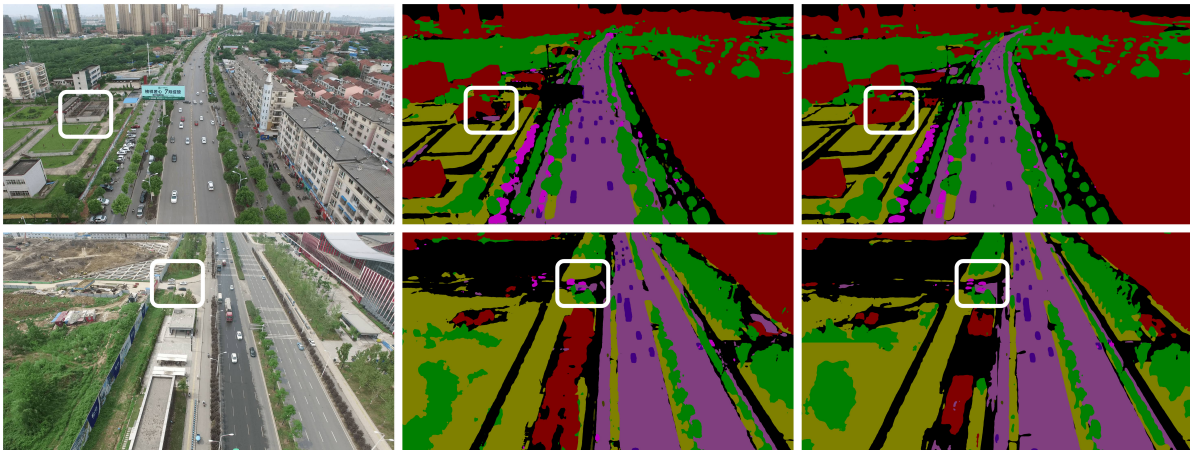


Figure 6.5: Comparative segmentation results from the UAVid [60] test dataset. First column shows the input RGB images, second column depicts the outputs of the previous SOTA [60] and the third column shows the predictions of our architecture. White boxes highlight the regions of efficient feature aggregation.

6.3.3. Benchmarking on Jetson Xavier NX

In order to create effective comparisons, we benchmark SOTA real-time semantic segmentation models on an embedded device Jetson Xavier NX, a small form factor system-on-module (SOM). The device has a 384-core NVIDIA Volta™ GPU with 48 Tensor Cores and a 6-core NVIDIA Carmel ARM®v8.2 64-bit CPU. Inferencing on full scale GPUs like Titan X, RTX20 series etc. is unlikely to provide a real-world analysis, as self-driving cars and other autonomous vehicles like UAVs or UGVs are more likely to have low-power consumption modules, with limited memory resources like Drive AGX, Jetson TX2, Xavier NX etc. We benchmark our algorithm and others on multiple resolutions to demonstrate the efficiency. The mIOU scores have not been provided in Table 6.4 as the models were trained and evaluated on different sizes and not all official implementations were available to evaluate. Hence, we only provide the system execution speed, memory footprints and GFLOP count on $\frac{1}{4}^{th}$ and $\frac{1}{16}^{th}$ of the standard Cityscapes resolution (2048×1024).

6.4. Results on Other Datasets

Apart from the above-mentioned primary datasets, we also benchmark our algorithm on other public datasets in order to further demonstrate the efficiency and robustness of our algorithm. Specifically, we use AeroScapes [63] dataset in this section. The AeroScapes dataset contains 11 classes for semantic segmentation and contains a total of 3269 images, out of which 2700 are used for training and the 569 for testing. This split is

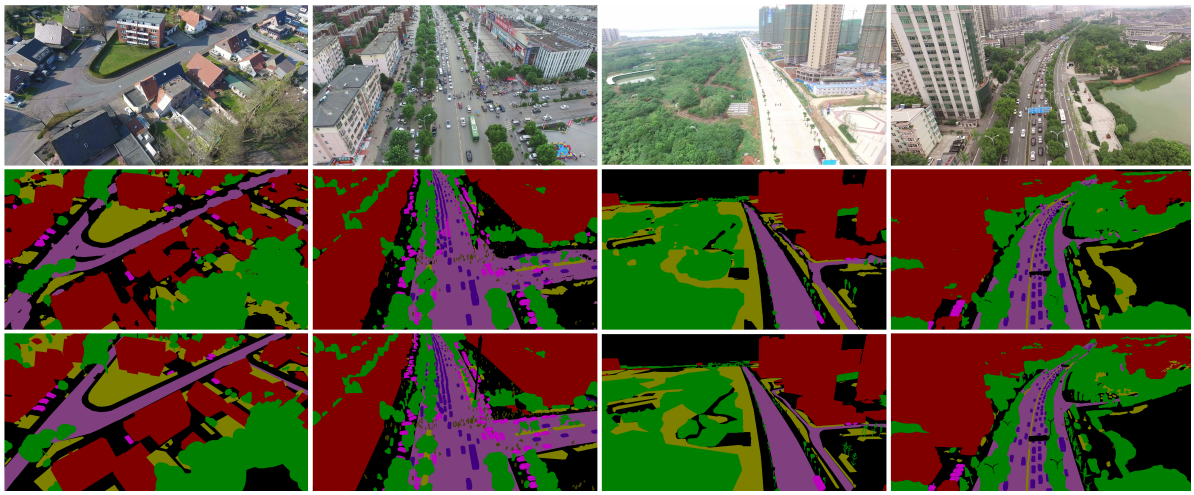


Figure 6.6: More segmentation results on the UAVid validation set. First row consists of the input RGB images. Second row contains the predictions from our architecture and the third row shows the ground truths of the input images. Best viewed in color.

Model	**Input Size**							
	256x512				**512x1024**			
	Memory	**MAdd**	**FLOPs**	**FPS**	**Memory**	**MAdd**	**FLOPs**	**FPS**
ERFNet	201.50MB	13.37G	6.41G	19.02	806.00MB	53.47G	25.65G	5.20
BiseNet	121.35MB	13.01G	6.48G	28.07	418.36MB	52.05G	25.93G	8.84
BiseNetV2	174.06MB	12.98G	6.46G	26.66	511.56MB	50.25G	24.93G	8.12
SwiftNet	104.66MB	12.99G	6.47G	26.19	418.06MB	51.95G	25.89G	9.14
ShelfNet	72.39MB	11.74G	5.86G	26.97	289.53MB	46.97G	23.42G	7.30
ICNet	68.40MB	10.15G	5.06G	8.49	273.62MB	40.61G	20.26G	3.38
DABNet	205.47MB	5.18G	2.62G	41.06	821.88MB	20.71G	10.22G	10.95
CGNet	195.96MB	3.44G	1.69G	25.77	786.75MB	13.75G	6.76G	11.43
DFANet	111.16MB	1.92G	955.16M	15.25	444.55MB	7.67G	3.82G	13.82
ContextNet	88.65MB	869.38M	419.61M	60.97	356.44MB	3.49G	1.68G	36.77
Fast-SCNN	77.37MB	865.74M	419.78M	69.64	309.71MB	3.46G	1.68G	42.34
SINet	42.00MB	187.02M	74.98M	28.28	168.00MB	784.04M	299.90M	24.78
CABiNet (Ours)	61.23MB	1.03G	502.22M	45.55	244.82MB	4.10G	2.01G	35.72

Table 6.4: Jetson Xavier NX has 6 modes of operation, depending on the power consumption and the number of cores utilized. For full resolution testing (1024×2048), we employ the maximum power mode (15W, all 6 cores). However, for the smaller resolutions (512×1024 and 256×512) we use a lower mode (10W, only 4 cores) to establish an effective comparison between the possible use-cases. For instance, implementing semantic segmentation on lower resolutions is likely to imply that there could be more processes running, and hence considering the usage of other cores for other threads, we utilize only 4. The execution speed is affected by the number of processors involved in computations.

inspired from the author of the dataset [63]. Qualitative results are shown in **Fig.** 6.7, and qualitative results are shown in Table 6.5. This benchmark was captured around several cities with the help of a UAV with height ranging from 5 to 50 meters.

Model	mIOU	Person	Bike	Car	Drone	Boat	Animal	Obstacle	Building	Vegetation	Road	Sky	FPS
Ensemble-Softmax	57.0	48.2	14.8	69.0	47.7	51.4	38.6	14.0	70.5	92.1	86.2	93.8	–
Ensemble-Average	56.6	48.0	14.5	68.5	47.5	49.2	38.2	13.9	70.1	92.0	86.0	93.5	–
Ensemble-MixMatch	55.0	48.5	13.5	71.8	41.8	43.6	36.5	13.8	67.5	91.8	82.6	94.0	–
Ensemble-Winner	53.8	47.5	11.2	66.0	43.5	40.0	38.0	14.0	68.4	91.5	83.5	92.5	–
CABiNet (Ours)	**69.2**	**48.7**	**32.1**	**83.0**	**74.2**	**95.2**	**40.3**	**21.6**	**80.0**	**96.9**	**94.0**	**95.2**	85.4

Table 6.5: Quantitative results on AeroScapes dataset [63]. Our superior context aggregation techniques outperform the previous SOTA on this dataset by a significant margin, while maintaining a real-time performance. – indicates that the corresponding values could not be confirmed.

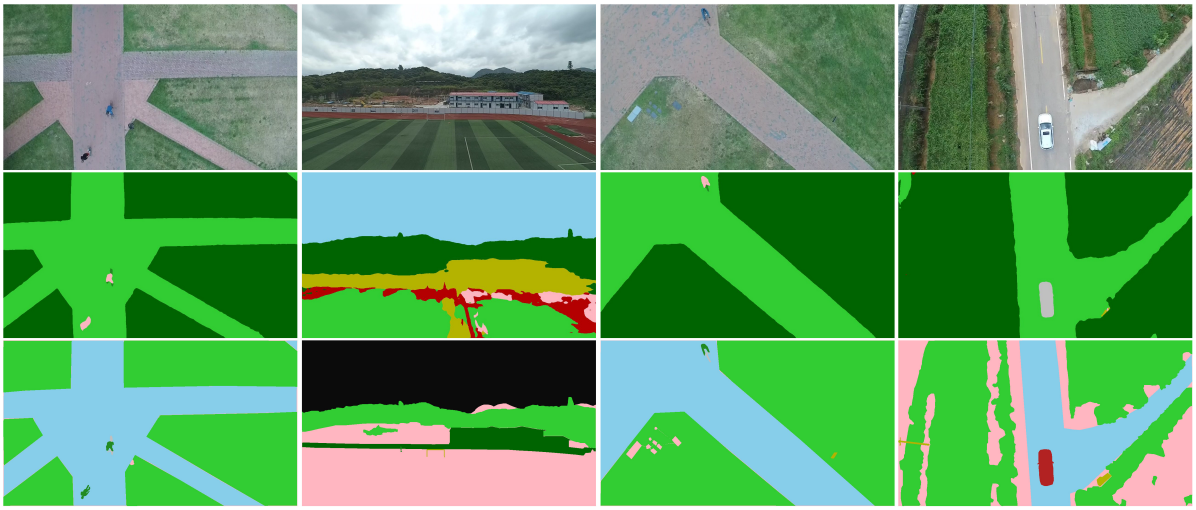


Figure 6.7: Segmentation results on the Aeroscapes [63] validation set. First row consists of the input RGB images. Second row contains the predictions from our architecture and the third row shows the ground truths of the input images. Best viewed in color.

6.5. Speed Computations

The methodology for computing the inference speed of a deep learning model has been studied and presented in certain literature [67, 72, 73, 112]. However, the implementations are naive in their approaches, in the sense that the suggested approaches use a Python library called `time` to start the clock when the model is called and stop the the clock immediately after the model returns the predictions. This may seem like a sufficient criteria for computing the inference speeds, but there is more to this than meets the eye.

When a deep learning model is instantiated on a GPU, the CPU first initiates the required kernels on the GPU, it then launches the kernels, then it copies the data from the RAM to the GPU and then the GPU gets to process. After this data is copied back to the RAM/CPU and the kernels are shut down. The point is not to disregard these operations, but to synchronize them. GPU operations are by default asynchronous, which implies that all the above-mentioned operations are not checked internally by a clock, hence resulting in sequential completion where parallel computations are possible. So it becomes necessary to synchronize these computations in order to get an accurate estimate of the inference speed.

Furthermore, when a model is instantiated on a GPU, the first few computations often take the largest amount of time. This reflects largely in the final average of the computed speeds while reporting. The reason why this happens is that the GPU tends to warm-up in the first few computations so the initial procedures are time consuming. Hence, it again becomes crucial to instantiate the GPU before computing the actual inference speeds. In this regard, we use the following algorithm for accurate inference speed computation, which takes care of all the aforementioned problems.

```
1
2  def compute_speed(model, input_size, device, iteration):
3
4    torch.cuda.set_device(device)
5    torch.backends.cudnn.benchmark = True
6
7    model.eval()
8    model = model.cuda()
9
10   input = torch.randn(*input_size, device=device)
11   logger.info('=========Warmup=========')
12
13   torch.cuda.synchronize()
14
15   for _ in range(50):
16     model(input)
17     torch.cuda.synchronize()
18
19   logger.info('=========Speed Testing=========')
20   logger.info("=========DEVICE:%s SIZE:%s=========" % (
21       torch.cuda.get_device_name(device), input_size))
```

```
time_spent = []
for i in range(iteration):
  torch.cuda.synchronize()
  t_start = time.perf_counter()
  with torch.no_grad():
    model(input)
  torch.cuda.synchronize()
  time_spent.append(time.perf_counter() - t_start)
  if (i+1) % 100 == 0: print("Iterations {} Complete".format(i+1))

torch.cuda.synchronize()
elapsed_time = np.sum(time_spent)
with torchprof.Profile(model, use_cuda=True) as prof:
  model(input)

logger.info(prof.display(show_events=False))
logger.info('Elapsed time: [%.2f s / %d iter]' % (elapsed_time, iteration))
logger.info('Speed Time: %.2f ms / iter    FPS: %.2f' % (
  elapsed_time / iteration * 1000, iteration / elapsed_time))
logger.info('\n')
```

Listing 6.1: Speed Computation Algorithm

As it can be seen here, we first initialize the model, synchronize the operations at every step and let the GPU run for the first 50 iterations. Immediately after the GPU is ready, we perform our speed computations while synchronizing all the operations.

6.6. Ablation Studies

In this chapter we aggressively study the impacts of several components in our architecture on the overall mIOU and the execution speeds. We present design choices for future developments and their effects on the final results. All architectures presented in this section use MobileNetV3-Small [31] as the feature extractor. Baseline is defined as a simple dual-branch network with two convolution layers in the spatial branch and untrained feature extractor in the second branch. The baseline is devoid of CAPLA and bottleneck modules and is similar in structure with [72]. For fusing the features from both branches we simply add them which are later discriminated by a small classifier block into the respective number of classes. Both the branches are fed images at the same resolution, unlike [72] and all the ablation experiments are performed on this baseline.

Model	mIOU
Baseline	68.4
Baseline + SB + CB	72.3
Baseline + SB + CB + CAB	74.7
Baseline + SB + CB + CAB + FFM (WA)	76.6

Table 6.6: Basic ablation study. SB and CB stand for spatial and context branches, whereas FFM (WA) stands for feature fusion module with weighted attention respectively.

6.6.1. Context Aggregation Block

The context aggregation block (CAB) is designed specifically to capture local and global context effectively and efficiently. For the case of ablation studies, it is interesting to see how the model performs without this specific block. If we remove CAB from the design keeping all other modules and training/inference parameters intact, we observe a drop of 2.1% in the overall mIOU score, along with a drop in inference time by almost 3ms. This establishes the fact that the segmentation head is capable enough to obtain a decent mIOU score on the Cityscapes validation set. The addition of the context block enhances the feature representations, while having minimal impact on the overall execution speed and complexity.

Now since this block has been introduced as a plug-n-play module, it is worth observing the impacts of CAB if it were to be implemented in other dual-branch architectures.

Table 6.7 proves the efficacy of the proposed context aggregation block, in a manner that it canbe used with any dual-branch architecture for semantic segmentation, indicating the superiority in design.

Model	mIOU w/o CAB	mIOU w CAB
ContextNet [72]	66.1	69.2 ↑
Fast-SCNN [73]	68.4	71.2 ↑
BiSeNet [112]	74.7	75.3 ↑
BiSeNetV2 [113]	74.2	75.1 ↑

Table 6.7: CAB implemented in other algorithms. Straightforward addition to [72, 73, 113] results in significant improvements over the baseline models. In [112], the proposed attention refinement modules were replaced with CAB.

6.6.2. Backbone Choice

A lot of previous real-time semantic segmentation architectures [67, 112, 119, 123] employ powerful feature extractors like ResNet-18 [27]. Even though this choice is justified for accurate semantic segmentation, the implications on execution speed and computational complexity are profound. Consider, for instance Table 6.8, where we provide a detailed comparison between the various possible efficient backbone designs for real-time perception applications like object detection, segmentation etc.

Model	mIOU	Memory	MAdd	Flops	Params	FPS
EffNet [19]	$M-0.5$	4712.00MB	41.46G	21.41G	1.76M	$F-10$
ShuffleNetV2 (0.5×) [61]	$M-0.6$	469.50MB	3.49G	1.65G	0.34M	$F+5$
ShuffleNetV2 (1.0×) [61]	$M-0.1$	871.12MB	12.44G	6.29G	1.25M	F
ShuffleNetV2 (1.5×) [61]	$M+0.3$	1225.50MB	25.36G	12.45G	2.48M	$F-15$
ShuffleNetV2 (2.0×) [61]	$M+0.6$	1651.12MB	49.82G	25.05G	5.35M	$F-20$
ResNet-18 [27]	M	944.00MB	112.01G	56.81G	8.18M	$F-20$
MobileNetV3-Large (0.75×) [31]	$M+0.6$	1946.92MB	13.31G	6.33G	1.79M	$F-20$
MobileNetV3-Large (1.×) [31]	$M+1.3$	2138.05MB	18.48G	8.88G	2.97M	$F-10$
MobileNetV3-Small (0.75×) [31]	$M-2.6$	619.65MB	3.69G	1.74G	0.57M	$F+5$
MobileNetV3-Small (1.×) [31]	M	672.78MB	4.78G	2.28G	0.93M	F

Table 6.8: Computational comparison between common light-weight feature extractors. M and F indicate the ground value of mIOU and FPS (measured on RTX 2080Ti on 2048×1024 resolution) on Cityscapes validation set, which are 76.6 and 76.50 respectively. All other models are evaluated against these references. All other computational expenses are measured for the extractors (backbones) alone and not for the overall segmentation model. Relative improvements over the ground values are shown in the mIOU and FPS columns.

Let us begin by analyzing the ShuffleNetV2 [61] series first. As we can see from Table 6.8, ShuffleNetV2 (0.5×) and ShuffleNetV2 (1.0×) do not provide the same mIOU score as the reference model with MobileNetV3-Small (1.0×). Furthermore, the deeper versions of ShuffleNetV2 although provide an improvement in the overall mIOU score, they tend to have significant increase in their memory footprints and computational complexities (increased GFLOPs count). This in turn, causes a decrease in the overall FPS of the segmentation model. EffNet [19] has an undesirable memory footprint, thereby rendering the segmentation model ineffective on embedded platforms.

Next, let us compare the internal variations of the MobileNetV3 design. As we can see from the table, the larger versions have significant memory footprints, even though the complexity is lower than those of ShuffleNetV2 and ResNet-18. The number of parameters is marginally acceptable for both the large designs of MobileNetV3. So even though the larger variations have decent improvements in the overall mIOU score, the memory footprints become the limiting factors, the effects of which can be seen on the overall FPS as well.

Therefore, we are essentially left with the smaller versions of MobileNetV3. Both of these (MobileNetV3-Small (0.75×) and MobileNetV3-Small (1.×)) have almost comparable memory requirements and complexities. However, the mIOU scores on both backbones are significantly apart. Naturally, the optimal choice for our use-case (low-latency high accuracy) becomes MobileNetV3-Small (1.×).

It is also interesting to observe the relative complexities of the SOTA [67] and our architecture with the same backbones. Hence, we utilize ResNet-18 as our primary feature extractor for effective comparison between the SOTA [67] for real-time semantic segmentation and our model. The quantified results are shown in Table 6.9.

As it can be seen from the table above, our segmentation head is lighter, faster and more accurate as compared to both SwiftNet [67] and BiSeNet [112], even if we use the same feature extractor as them which is ResNet-18. This, when further coupled with an even more light-weight backbone, creates significant impacts

Model	mIOU	Memory	MAdd	Flops	Params	FPS
BiseNet [112]	74.8	1941.39MB	208.18G	103.72G	12.89M	47.20
SwiftNet [67]	75.4	1671.66MB	207.64G	103.37G	11.80M	45.40
CABiNet-R18 (Ours)	76.7	1502.58MB	132.51G	66.41G	9.19M	54.50
CABiNet-MV3 (Ours)	76.6	1256.18MB	24.37G	12.03G	2.64M	66.50

Table 6.9: Relative complexity comparison between our approach and the current state-of-the-art. With ResNet-18 [27] as the backbone, the computational complexities become more comparable between the two architectures. CABiNet offers a 35% reduction in computations, with comparable mIOU along-with a 16% reduction in the overall inference time. Both the approaches [67, 112] use ResNet-18 as the primary feature extractor. R18 and MV3 stand for ResNet-18 and MobileNetV3-Small (1.×) respectively.

on the overall inference. Furthermore, the comparison between CABiNet-R18 and CABiNet-MV3 from Table 6.9 and Table 6.8 reveals that the computational overheads added by ResNet-18 are larger as compared to MobileNetV3-Small even though it provides similar mIOU scores. Since the segmentation head is light, using a heavy backbone and limiting the capabilities of the overall model in terms of inference, does not seem to be a logical decision.

6.6.3. Spatial Branch

In this study, we replace the spatial branch with the final feature representations from the backbone. A small convolutional layer with batch normalization is applied to reduce the number of channels as per the requirements of the feature fusion module. As a result, the model becomes lighter but also tends to have a lower mIOU score as compared to the original model. Specifically, we see a drop of 0.5% in the overall mIOU on the Cityscapes validation set. Another aspect that is observed is the increased variance in the overall design. Without the spatial branch, the segmentation head tends to have a larger variance in terms of the final mIOU and was observed to have a variance as large as 2%. This implies that the final mIOU is within ± 2 range. The spatial branch reduces this uncertainty in the final architecture at the cost of slight increase in computations.

6.6.4. Feature Fusion Module

Several fusion techniques have been suggested in literature, and designing the right technique could have significant impacts on the final outcome. We experiment with simple feature addition, simple feature concatenation and feature concatenation with weighted addition. Consider Table 6.10 for a quanititative comparision between the various fusion techniques.

Fusion Style	mIOU	FLOPs
Feature Addition [72]	73.2	0.5G
Feature Concatenation w/o AW [73]	74.5	0.8G
Feature Concatenation w AW [112]	76.6	1.5G

Table 6.10: AW here stands for attention weight based fusion.

As can be seen from the table above, feature concatenation with with weighted attention provides the best mIOU scores out of the three variants, at the slight cost complexity which is acceptable.

6.6.5. Sampling Method Choice

The sampling strategy used in the CAPA module has to potential to affect its performance significantly. Normally, there are three variations, $random$, max and $average$, but in this research we experiment only with the max and $average$ as the random sampling technique is less efficient as compared to other two [118]. Furthermore from the same literature, it can be established that the $average$ sampling method is likely to perform better than the max method. Theoretically, this could be attributed to the fact that the $average$ sampling method considers all the input spatial operators, unlike the other two, which can be seen from Table 6.11.

6.6.6. Number of Sparse Representations

The layers of the pyramid pooling module determine the number of sparse anchor points which, in turn have an impact on the performance of the AP block. Following the footsteps of [118], we experiment with multiple variations of the output sizes of PSP module. Specifically, we experiment with bin sizes of $1,2,3,6$ and $1,3,6,8$

Sampling Strategy	mIOU
Random Pooling	76.2
Max Pooling	76.4
Average Pooling	76.6

Table 6.11: Different pooling strategies and their impacts on the overall mIOU.

and their combinations with max and $average$ methods. Considering the impacts on mIOU score and the computational expenses, even though $average$ pooling with a bin size of $1,2,3,6$ provides sufficient semantic coverage, we choose the former combination at the slight cost of increases computations because it also gives us a slightly better result of 76.6%.

6.6.7. Bottleneck

The combination of the bottleneck and the attention fusion provides for the double supervision of the attention branch. Since the context branch generates a decent semantic representation and comprises of the most crucial components in the model, the bottleneck is added to create a semantic distinction from the extracted features, before the representations are fed to the feature fusion module. This bottleneck allows us to supervise the context branch directly, thereby improving the overall performance. The bottleneck provides for additional supervision of the entire model, along-with a slight boost in the overall mIOU score by 0.3%.

7

Discussions and Future Work

In this research, a new methodology was presented to tackle the challenge of real-time semantic segmentation with a novel context aggregation block design. Local and global context assimilation were observed both qualitatively and quantitatively in the report. The pipeline presented is simple in terms of design, implementation and execution. We start with the input RGB image and perform parallel operations to extract deep features and spatial features using two separate heads. It is observed that both long-range and short-range dependencies prove to be crucial for accurate semantic segmentation. No post-processing techniques are applied like SegFix [115] or Deep CRFs [42, 74] for further enhancement of the obtained predictions. Predictions were directly submitted to the official servers for evaluation. While improving the overall mIOU score, we am also able enhance the execution speed significantly with the help of the novel context aggregation block and a strategic use of depth-wise convolutions.

A possible future work would be to implement the suggested approach using distributed training technique. Instead of training on a single GPU, the architecture could be trained on multiple-GPUs in parallel with a larger batch size and reduced number of iterations. This can bring about an increase in the overall mIOU score on both datasets.

Another possible extension of this project is forecasted in the temporal domain. By training on spatio-temporal data, instead of static images, the temporal information of videos could be harnessed to further boost the performance. It is possible to design attention blocks that incorporate temporal abstractions as well. In fact, non-local operators that were initially proposed in [102] did have a temporal representation layer and were proposed for understanding object semantics in videos.

There are a couple of other datasets available such as DroneSeg, SkyScapes [2] etc. which could be used for benchmarking this algorithm. However, unlike UAVid and Cityscapes, these dataset are not well managed and do not provide official testing servers for evaluation. They also suffer from relatively poor documentation. Hence, only two primary datasets were chosen for this research along-with one other, Aeroscapes [63].

If the output layer of pixel-wise classification is replaced with an object detection head (a region proposal network and bounding box regressor), this same pipeline could be used for accurate high-speed object detection in image and in videos as well.

Furthermore, this segmentation head could very well be modified to suit the requirements of instance-aware segmentation. Coupled with this new methodology, segmentation head and instance head could further be combined to address panoptic segmentation as a whole.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Seyed Majid Azimi, Corentin Henry, Lars Sommer, Arne Schumann, and Eleonora Vig. Skyscapes fine-grained semantic understanding of aerial scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7393–7403, 2019.

[3] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[5] Min Bai and Raquel Urtasun. Deep watershed transform for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5221–5229, 2017.

[6] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 9157–9166, 2019.

[7] Siddhartha Chandra and Iasonas Kokkinos. Fast, exact and multi-scale inference for semantic image segmentation with deep gaussian crfs. In *European conference on computer vision*, pages 402–418. Springer, 2016.

[8] Gabriel Chartrand, Phillip M Cheng, Eugene Vorontsov, Michal Drozdzal, Simon Turcotte, Christopher J Pal, Samuel Kadoury, and An Tang. Deep learning: a primer for radiologists. *Radiographics*, 37(7):2113–2131, 2017.

[9] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4974–4983, 2019.

[10] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[11] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[12] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[13] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[14] Wuyang Chen, Xinyu Gong, Xianming Liu, Qian Zhang, Yuan Li, and Zhangyang Wang. Fasterseg: Searching for faster real-time semantic segmentation. *arXiv preprint arXiv:1912.10917*, 2019.

[15] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12475–12485, 2020.

[16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] Clément Dechesne, Clément Mallet, Arnaud Le Bris, and Valérie Gouet-Brunet. Semantic segmentation of forest stands of pure species combining airborne lidar data and very high resolution multispectral imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 126:129–145, 2017.

[18] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[19] Ido Freeman, Lutz Roese-Koerner, and Anton Kummert. Effnet: An efficient structure for convolutional neural networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 6–10. IEEE, 2018.

[20] Björn Fröhlich, Eric Bach, Irene Walde, Sören Hese, Christiane Schmullius, and Joachim Denzler. Land cover classification of satellite images using contextual information. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 3(W1), 2013.

[21] Cheng-Yang Fu, Tamara L Berg, and Alexander C Berg. Imp: Instance mask projection for high accuracy semantic segmentation of things. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5178–5187, 2019.

[22] Huazhu Fu, Yanwu Xu, Stephen Lin, Damon Wing Kee Wong, and Jiang Liu. Deepvessel: Retinal vessel segmentation via deep learning and conditional random field. In *International conference on medical image computing and computer-assisted intervention*, pages 132–139. Springer, 2016.

[23] Jun Fu, Jing Liu, Haijie Tian, Yong Li, Yongjun Bao, Zhiwei Fang, and Hanqing Lu. Dual attention network for scene segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3146–3154, 2019.

[24] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589, 2020.

[25] Zhongyi Han, Benzheng Wei, Ashley Mercado, Stephanie Leung, and Shuo Li. Spine-gan: Semantic segmentation of multiple spinal structures. *Medical image analysis*, 50:23–35, 2018.

[26] Shijie Hao, Yuan Zhou, and Yanrong Guo. Bi-direction context propagation network for real-time semantic segmentation. *arXiv preprint arXiv:2005.11034*, 2020.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[30] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. 2016.

[31] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[32] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[33] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[34] Lang Huang, Yuhui Yuan, Jianyuan Guo, Chao Zhang, Xilin Chen, and Jingdong Wang. Interlaced sparse self-attention for semantic segmentation. *arXiv preprint arXiv:1907.12273*, 2019.

[35] Wei Ji, Xi Li, Yueting Zhuang, Omar El Farouk Bourahla, Yixin Ji, Shihao Li, and Jiabao Cui. Semantic locality-aware deformable network for clothing segmentation. In *IJCAI*, pages 764–770, 2018.

[36] Bin Jiang, Wenxuan Tu, Chao Yang, and Junsong Yuan. Context-integrated and feature-refined network for lightweight urban scene parsing. *arXiv preprint arXiv:1907.11474*, 2019.

[37] Feng Jiang, Aleksei Grigorev, Seungmin Rho, Zhihong Tian, YunSheng Fu, Worku Jifara, Khan Adil, and Shaohui Liu. Medical image semantic segmentation based on deep learning. *Neural Computing and Applications*, 29(5):1257–1265, 2018.

[38] Baris Kayalibay, Grady Jensen, and Patrick van der Smagt. Cnn-based segmentation of medical imaging data. *arXiv preprint arXiv:1701.03056*, 2017.

[39] Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[40] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9404–9413, 2019.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[42] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[43] Gen Li, Inyoung Yun, Jonghyun Kim, and Joongkyu Kim. Dabnet: Depth-wise asymmetric bottleneck for real-time semantic segmentation. *arXiv preprint arXiv:1907.11357*, 2019.

[44] Hanchao Li, Pengfei Xiong, Haoqiang Fan, and Jian Sun. Dfanet: Deep feature aggregation for real-time semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9522–9531, 2019.

[45] Hongliang Li, King N Ngan, and Qiang Liu. Faceseg: automatic face segmentation for real-time video. *IEEE Transactions on Multimedia*, 11(1):77–88, 2008.

[46] Qizhu Li, Anurag Arnab, and Philip HS Torr. Weakly-and semi-supervised panoptic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 102–118, 2018.

[47] Weijia Li, Conghui He, Jiarui Fang, and Haohuan Fu. Semantic segmentation based building extraction method using multi-source gis map datasets and satellite imagery. In *CVPR Workshops*, pages 238–241, 2018.

[48] Xiangtai Li, Li Zhang, Ansheng You, Maoke Yang, Kuiyuan Yang, and Yunhai Tong. Global aggregation then local distribution in fully convolutional networks. *arXiv preprint arXiv:1909.07229*, 2019.

[49] Yanwei Li, Xinze Chen, Zheng Zhu, Lingxi Xie, Guan Huang, Dalong Du, and Xingang Wang. Attention-guided unified network for panoptic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7026–7035, 2019.

[50] Xiaodan Liang, Liang Lin, Wei Yang, Ping Luo, Junshi Huang, and Shuicheng Yan. Clothes co-parsing via joint image segmentation and labeling with application to clothing retrieval. *IEEE Transactions on Multimedia*, 18(6):1175–1186, 2016.

[51] Guosheng Lin, Chunhua Shen, Anton Van Den Hengel, and Ian Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3194–3203, 2016.

[52] Peiwen Lin, Peng Sun, Guangliang Cheng, Sirui Xie, Xi Li, and Jianping Shi. Graph-guided architecture search for real-time semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4203–4212, 2020.

[53] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[54] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[55] Huanyu Liu, Chao Peng, Changqian Yu, Jingbo Wang, Xu Liu, Gang Yu, and Wei Jiang. An end-to-end network for panoptic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6172–6181, 2019.

[56] Mengyu Liu and Hujun Yin. Feature pyramid encoding network for real-time semantic segmentation. *arXiv preprint arXiv:1909.08599*, 2019.

[57] Wei Liu, Andrew Rabinovich, and Alexander C Berg. Parsenet: Looking wider to see better. *arXiv preprint arXiv:1506.04579*, 2015.

[58] 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.

[59] Haoran Lyu, Huiyuan Fu, Xiaojun Hu, and Liang Liu. Esnet: Edge-based segmentation network for real-time semantic segmentation in traffic scenes. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1855–1859. IEEE, 2019.

[60] Ye Lyu, George Vosselman, Gui-Song Xia, Alper Yilmaz, and Michael Ying Yang. Uavid: A semantic segmentation dataset for uav imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 165: 108–119, 2020.

[61] 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.

[62] Andreas C Müller and Sven Behnke. Learning depth-sensitive conditional random fields for semantic segmentation of rgb-d images. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6232–6237. IEEE, 2014.

[63] Ishan Nigam, Chen Huang, and Deva Ramanan. Ensemble knowledge transfer for semantic segmentation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1499–1508. IEEE, 2018.

[64] Ruigang Niu. Hmanet: Hybrid multiple attention network for semantic segmentation in aerial images. *arXiv preprint arXiv:2001.02870*, 2020.

[65] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[66] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

[67] 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 conference on computer vision and pattern recognition*, pages 12607–12616, 2019.

[68] Hyojin Park, Lars Sjosund, YoungJoon Yoo, Nicolas Monet, Jihwan Bang, and Nojun Kwak. Sinet: Extreme lightweight portrait segmentation networks with spatial squeeze module and information blocking decoder. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2066–2074, 2020.

[69] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[70] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[71] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2017.

[72] Rudra PK Poudel, Ujwal Bonde, Stephan Liwicki, and Christopher Zach. Contextnet: Exploring context and detail for semantic segmentation in real-time. *arXiv preprint arXiv:1805.04554*, 2018.

[73] Rudra PK Poudel, Stephan Liwicki, and Roberto Cipolla. Fast-scnn: fast semantic segmentation network. *arXiv preprint arXiv:1902.04502*, 2019.

[74] Ariadna Quattoni, Michael Collins, and Trevor Darrell. Conditional random fields for object recognition. In *Advances in neural information processing systems*, pages 1097–1104, 2005.

[75] Petia Radeva and Enric Martí. Facial features segmentation by model-based snakes. In *International Conference on Computing Analysis and Image Processing, Prague*, pages 1–5. Citeseer, 1995.

[76] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909*, 2019.

[77] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[78] Eduardo Romera, José M Alvarez, Luis M Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2017.

[79] Bernardino Romera-Paredes and Philip Hilaire Sean Torr. Recurrent instance segmentation. In *European conference on computer vision*, pages 312–329. Springer, 2016.

[80] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[81] Holger R Roth, Chen Shen, Hirohisa Oda, Masahiro Oda, Yuichiro Hayashi, Kazunari Misawa, and Kensaku Mori. Deep learning and its application to medical image segmentation. *Medical Imaging Technology*, 36(2):63–71, 2018.

[82] Jose-Raul Ruiz-Sarmiento, Cipriano Galindo, Javier Monroy, Francisco-Angel Moreno, and Javier Gonzalez-Jimenez. Ontology-based conditional random fields for object recognition. *Knowledge-Based Systems*, 168:100–108, 2019.

[83] Shunsuke Saito, Tianye Li, and Hao Li. Real-time facial segmentation and performance capture from rgb input. In *European conference on computer vision*, pages 244–261. Springer, 2016.

[84] 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.

[85] Mauricio P Segundo, Chaua Queirolo, Olga RP Bellon, and Luciano Silva. Automatic 3d facial segmentation and landmark detection. In *14th International Conference on Image Analysis and Processing (ICIAP 2007)*, pages 431–436. IEEE, 2007.

[86] Maurício Pamplona Segundo, Luciano Silva, Olga Regina Pereira Bellon, and Chauã C Queirolo. Automatic face segmentation and facial landmark detection in range images. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(5):1319–1330, 2010.

[87] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[88] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016.

[89] Haiyang Si, Zhiqiang Zhang, Feifan Lv, Gang Yu, and Feng Lu. Real-time semantic segmentation via multiply spatial fusion network. *arXiv preprint arXiv:1911.07217*, 2019.

[90] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[91] Amber L Simpson, Michela Antonelli, Spyridon Bakas, Michel Bilello, Keyvan Farahani, Bram Van Ginneken, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, et al. A large annotated medical image dataset for the development and evaluation of segmentation algorithms. *arXiv preprint arXiv:1902.09063*, 2019.

[92] Paul Suetens, Erwin Bellon, Dirk Vandermeulen, M Smet, Guy Marchal, Johan Nuyts, and Luc Mortelmans. Image segmentation: methods and applications in diagnostic radiology and nuclear medicine. *European journal of radiology*, 17(1):14–21, 1993.

[93] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[94] Andrew Tao, Karan Sapra, and Bryan Catanzaro. Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*, 2020.

[95] Han Vanholder. Efficient inference with tensorrt, 2016.

[96] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[97] Raviteja Vemulapalli, Oncel Tuzel, Ming-Yu Liu, and Rama Chellapa. Gaussian conditional random field network for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3224–3233, 2016.

[98] Michele Volpi and Vittorio Ferrari. Semantic segmentation of urban scenes by learning local class interactions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–9, 2015.

[99] Panqu Wang, Pengfei Chen, Ye Yuan, Ding Liu, Zehua Huang, Xiaodi Hou, and Garrison Cottrell. Understanding convolution for semantic segmentation. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 1451–1460. IEEE, 2018.

[100] Sy Bor Wang, Ariadna Quattoni, L-P Morency, David Demirdjian, and Trevor Darrell. Hidden conditional random fields for gesture recognition. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1521–1527. IEEE, 2006.

[101] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2606–2615, 2017.

[102] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[103] Yang Wang and Greg Mori. Max-margin hidden conditional random fields for human action recognition. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 872–879. IEEE, 2009.

[104] Yang Wang, Kia-Fock Loe, and Jian-Kang Wu. A dynamic conditional random field model for foreground and shadow segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 28 (2):279–289, 2005.

[105] Yu Wang, Quan Zhou, Jia Liu, Jian Xiong, Guangwei Gao, Xiaofu Wu, and Longin Jan Latecki. Lednet: A lightweight encoder-decoder network for real-time semantic segmentation. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1860–1864. IEEE, 2019.

[106] Tianyi Wu, Sheng Tang, Rui Zhang, and Yongdong Zhang. Cgnet: A light-weight context guided network for semantic segmentation. *arXiv preprint arXiv:1811.08201*, 2018.

[107] Michael Wurm, Thomas Stark, Xiao Xiang Zhu, Matthias Weigand, and Hannes Taubenböck. Semantic segmentation of slums in satellite images using transfer learning on fully convolutional neural networks. *ISPRS journal of photogrammetry and remote sensing*, 150:59–69, 2019.

[108] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. Upsnet: A unified panoptic segmentation network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8818–8826, 2019.

[109] Maoke Yang, Kun Yu, Chi Zhang, Zhiwei Li, and Kuiyuan Yang. Denseaspp for semantic segmentation in street scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3684–3692, 2018.

[110] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[111] Wei Yang, Ping Luo, and Liang Lin. Clothing co-parsing by joint image segmentation and labeling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3182–3189, 2014.

[112] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 325–341, 2018.

[113] 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. *arXiv preprint arXiv:2004.02147*, 2020.

[114] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[115] Yuhui Yuan, Jingyi Xie, Xilin Chen, and Jingdong Wang. Segfix: Model-agnostic boundary refinement for segmentation. *arXiv preprint arXiv:2007.04269*, 2020.

[116] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7151–7160, 2018.

[117] Zhanpeng Zhang and Kaipeng Zhang. Farsee-net: Real-time semantic segmentation by efficient multi-scale context aggregation and feature space super-resolution. *arXiv preprint arXiv:2003.03913*, 2020.

[118] 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*, pages 2881–2890, 2017.

[119] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnet for real-time semantic segmentation on high-resolution images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 405–420, 2018.

[120] Zilong Zhong, Zhong Qiu Lin, Rene Bidart, Xiaodan Hu, Ibrahim Ben Daya, Zhifeng Li, Wei-Shi Zheng, Jonathan Li, and Alexander Wong. Squeeze-and-attention networks for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13065–13074, 2020.

[121] Lingyu Zhu, Tinghuai Wang, Emre Aksu, and Joni-Kristian Kamarainen. Cross-granularity attention network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[122] Zhen Zhu, Mengde Xu, Song Bai, Tengteng Huang, and Xiang Bai. Asymmetric non-local neural networks for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 593–602, 2019.

[123] Juntang Zhuang, Junlin Yang, Lin Gu, and Nicha Dvornek. Shelfnet for fast semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.