On Designing Efficient Deep Neural Networks for Semantic Segmentation

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Electronics and Communication Engineering
by Research

by

Nikitha Vallurupalli
201331220
nikitha.vallurupalli@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
November 2022

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "On Designing Efficient Deep Neural Networks for Semantic Segmentation" by Nikitha Vallurupalli, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. C.V. Jawahar
Date	Adviser: Dr. Girish Varma

To

My Parents

Acknowledgments

Firstly, I would like to thank my advisers, Prof. C. V. Jawahar and Dr. Girish Varma, for their guidance. They have been very patient, allowing me to have second chances, and supporting me through the journey.

I owe special gratitude to Thrupthi Ann John, who has been my incredible mentor and friend throughout this arduous road. She has greatly helped me in every aspect and has been a constant source of inspiration. I am grateful for her valuable suggestions and enormous amount of patience.

I would also like to thank Manu Mathew and Soyeb Nagori from Texas Instruments, Bangalore, who have collaborated with the work done in this thesis.

I would like to thank all my friends at IIIT who have given me great memories and a wonderful learning experience.

Most importantly, I am very grateful to my parents and sister for being my pillar of support and encouragement.

Abstract

Semantic segmentation is an essential primitive in real-time systems such as autonomous navigation, which require processing at high frames per second. Hence for models to be practically applicable, it is essential that they have to be compact, fast as well as achieve high prediction accuracies. Previous research into semantic segmentation has focused on creating high-performance deep learning architectures. Most of the time, these best-performing models are complex, deep, have large processing times, and demand a significantly higher amount of processing capacity. Another relevant area of research is model compression, by which we can obtain lightweight models. Considering that there also have been works that produced mainstream light-weight semantic segmentation models at the expense of performance, we design models that bring a desirable balance between performance and latency. Specifically, methods and architectures that give a *high performance* while being *real-time* and working on *resource-constrained settings*. We identify the redundancies in the existing state-of-the-art approaches and propose compact architecture family called ESSNet with accuracy comparable to the state-of-the-art while utilizing only a fraction of the space and computational power of those networks. We propose convolutional module designs with sparse coding theory as a premise and we also present two real-time encoder backbones employing our proposed modules. We empirically evaluate the efficacy of our proposed layers and compare them with existing approaches. Secondly, we explore the need for optimization during the training phase in the proposed models and present a novel training method called *Gradual Grouping* that results in models with improved implementation efficiency vs accuracy trade-offs. Additionally, we conduct extensive experiments by varying architecture hyper-parameters such as network depth, kernel sizes, dilation rates, split branching and additional context extraction modules. We also present a compact architecture using multi-branch separable convolutional layers with different dilation rates.

Contents

List of Figures

List of Tables

Chapter *1*

Introduction

1.1 Motivation

This thesis is motivated by the practical requirement of semantic segmentation in scene understanding applications on resource-constrained edge devices. Scene understanding is a fundamental computer vision problem. Semantic segmentation is a crucial task since it provides relevant context for the actions to be taken based on comprehending the scene at a pixel level [11]. Despite advances in the hardware, most real world scene understanding applications such as autonomous driving [73, 62], intelligent visual surveillance, driver assistance applications [14], augmented reality [47], intelligent transportation systems, scene tagging, and medical image diagnosis [75] still have the need for computationally cheap, small-sized networks for real-time processing.

In June of 2022, a fleet of self-driving cars by a company stopped moving in the middle of the highway, causing a huge traffic jam[1]. The cars lost contact with company servers for 90 minutes, leading to a dozen vehicles being frozen in the middle of the road, blocking lanes and crosswalks. It is nonviable to rely on remote computing clusters for heavy computation of critical applications due to issues with network connectivity and privacy. This illustrates the need for computing on edge devices in resource-constrained settings. These advanced scene understanding applications operate in resource-constrained environments with limited energy overhead, restricted onboard compute capacity, and memory. They require online processing of data locally on edge devices, and at the same time, the machine learning models deployed need to run in real-time. Therefore there has been a growing emphasis on developing real-time systems with minimal processing needs.

There have been giant strides in the accuracy of semantic segmentation due to the pervasiveness of deep learning [23, 20, 4]. However, as the task of semantic segmentation is computationally intensive, most of the proposed deep learning-based models do not serve real-time needs [11, 18, 23, 42]. The most popular architectures for semantic segmentation uses an encoder-decoder structure [42, 59]. We input the image into the encoder, which produces a compact representation. The decoder then intelligently upsamples the representation until it produces a semantic label for each pixel of the input image.

[1] https://www.wired.com/story/cruises-robot-car-outages/

In this thesis, we discuss other architectures used in literature, such as multi-scale, context module, and feature fusion-based models [49, 9, 13, 22, 39]. These models attempt to produce the most accurate semantic segmentation. Various model compression approaches have been proposed to improve the implementation efficiency of segmentation models. Explicit model compression applies post-hoc techniques to heavy networks after or during training, whereas implicit model compression uses lightweight structures and layers in the network architecture. Broadly they can be grouped under pruning [21, 36], quantization [19, 20], and architecture design [69, 6, 28] approaches. Efficient architecture design involves proposing novel neural network layers with minimal parameters that are expressive [52, 2]. These three approaches are orthogonal i.e. they can be applied together to improve performance. In our work, we use a combination of these approaches to design lightweight, high-performing architectures.

In this thesis, we investigate the feasibility of improving the overall performance of real-time segmentation models.

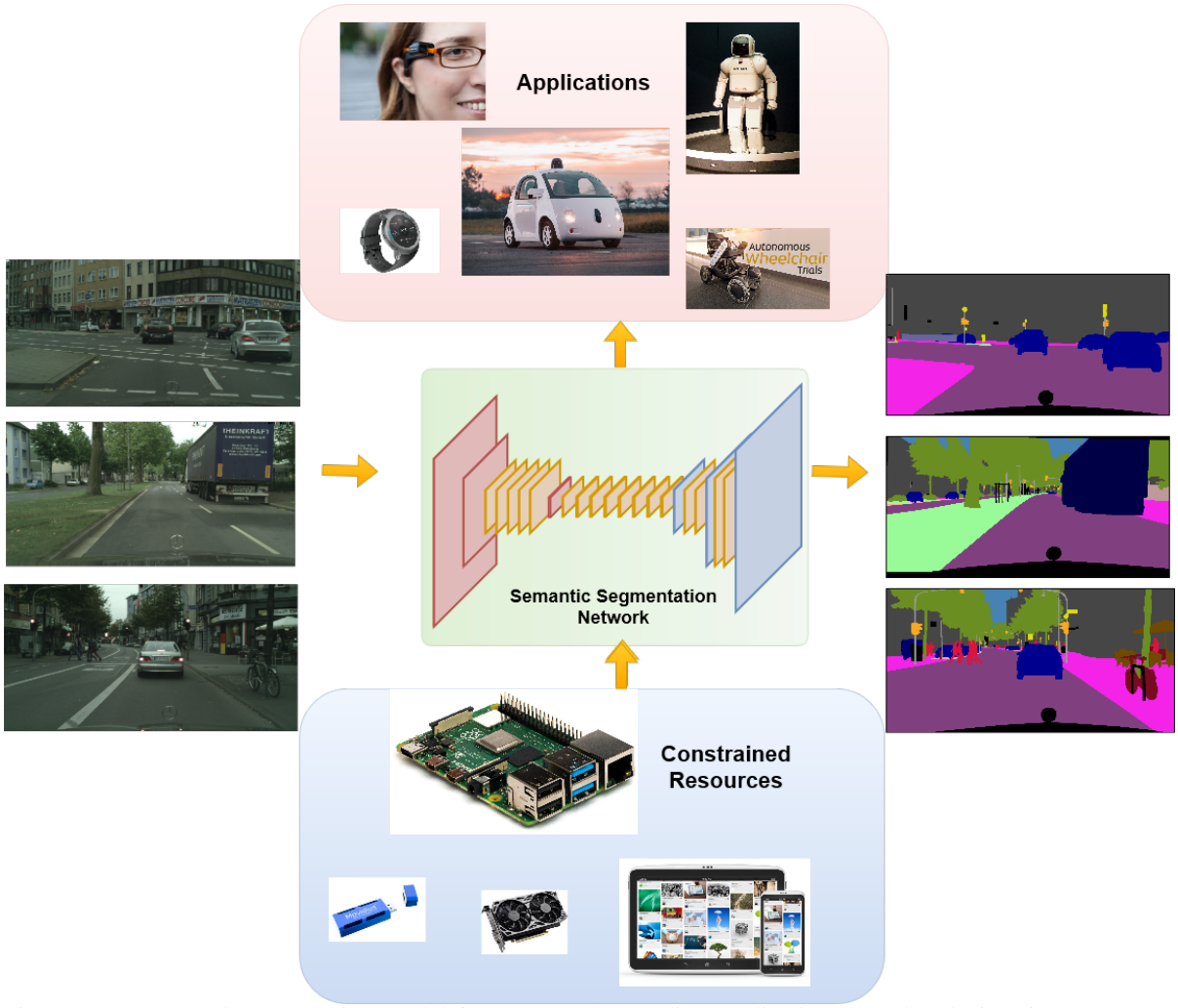

Figure 1.1: Proposed segmentation model for scene understanding applications on edge devices in a resource-constrained setting.

1.2 Contribution

The following are our contributions in this thesis:

- We comprehensively analyze the most effective convolutional design strategies and utilize them to alleviate the efficiency constraints. We present a component-wise analysis of real-time segmentation models, examine the shortcomings, and propose efficient convolutional layer designs that reduce redundancies without compromising on the performance.

- We evaluate the effectiveness of our proposed convolutional layers by implementing them in real-time semantic segmentation backbones ESSNetA, and ESSNetB. Our proposed networks called ESSNet are empirically demonstrated to be highly efficient and equally accurate while learning fewer parameters. Our proposed models run over 40FPS on a single GTX 1080Ti GPU with compact model size making them suitable for real-time applications on edge devices.

- We propose a novel training procedure called Gradual Grouping that optimizes the model at a higher dimensional subspace and transforms a dense convolution to a grouped convolution where the channel grouping is learnt while training. This method overcomes the limitations of leveraging sparse convolutions in the architecture design.

- We derive meaningful insights into the scalability and effectiveness of encoder-decoder style models in the context of semantic segmentation. We comprehend the significance of each design element in segmentation architectures and determine the components that are the best fit to build our proposed models. We conduct extensive ablation experiments on the network architecture by adding new blocks to the models.

- We propose a novel segmentation network WSPD-Net that incorporates a pyramid structure in the convolutional layer with differential dilation rates that is much more effective without increasing computational complexity. In this network, we employ a heterogeneous combination of spatially factorized and depthwise separable convolutional layers. We also exploit different kernel sizes with varying-sized receptive fields, which are proven to better segment diverse-sized objects.

1.3 Thesis Outline

In this chapter, we have introduced the problem statement, our motivation and contribution to the thesis. The remaining thesis is divided into four chapters.

- In Chapter 2, we present an overview of semantic segmentation. We detail the pertinent literature in context to model compression. We introduce the relevant datasets, and evaluation methodologies for semantic segmentation.

- In Chapter 3, we discuss several efficient CNN design strategies that were proven to be successful and present novel convolutional layer designs concerning segmentation architectures. We present various design considerations for segmentation architectures. We extensively examine the choice of each component in the architecture. In addition, we present efficient real-time segmentation networks and a novel training algorithm. We empirically demonstrate the effectiveness of our proposed layers. We benchmark our architectures on the Cityscapes dataset.
- In Chapter 4, we detail the identification of key elements concerning real-time segmentation networks. We discuss the benefits and merits of each architecture design hyper-parameter. Furthermore, we present a novel module design and introduce a real-time architecture that achieves better balance between run-time and performance.
- In Chapter 5, we present the thesis summary, conclusions and the possible future directions.

Chapter *2*

Background and Related Works

2.1 Semantic Segmentation: Preliminary

Semantic image segmentation is the challenge of dividing an image into a group of meaningful sections that are not overlapped and correspond to entities or portions of object classes that may provide semantics or high-level structural information [62, 14]. Classification, detection, and segmentation are visual perception tasks. Semantic segmentation aims to anticipate semantic labels at the pixel level for a image [63].

Image classification task assigns a single label to the whole image; in object detection, the precise location of target needs to be known. Unlike image classification, multiple objects belonging to the same class are treated as if they were a single item in semantic segmentation. On the other hand, instance segmentation considers numerous objects belonging to the same class as if they were separate unique objects (or instances). Figure 2.1 shows an example of semantic segmentation task. The segmentation output generated is a dense prediction, where each pixel of the input image is assigned a unique class label from a predefined set of categories. In order to be effective, semantic segmentation should be able to achieve high similarity within segments while achieving low association between segments. The segmentation border should correspond to human perception, and semantic segmentation should consider key characteristics while disregarding small-scale variation. In other words, semantic image segmentation should deconstruct an image into a limited collection of meaningful areas, each of which is of significant size.

2.2 Overview of Deep Learning Architectures for Semantic Segmentation

In this section, we discuss various deep learning-based approaches towards semantic segmentation, and categorize various models based on their architectures. Before the advent of deep learning, semantic segmentation was accomplished with a classifier that labels each superpixel in an image [16]. The

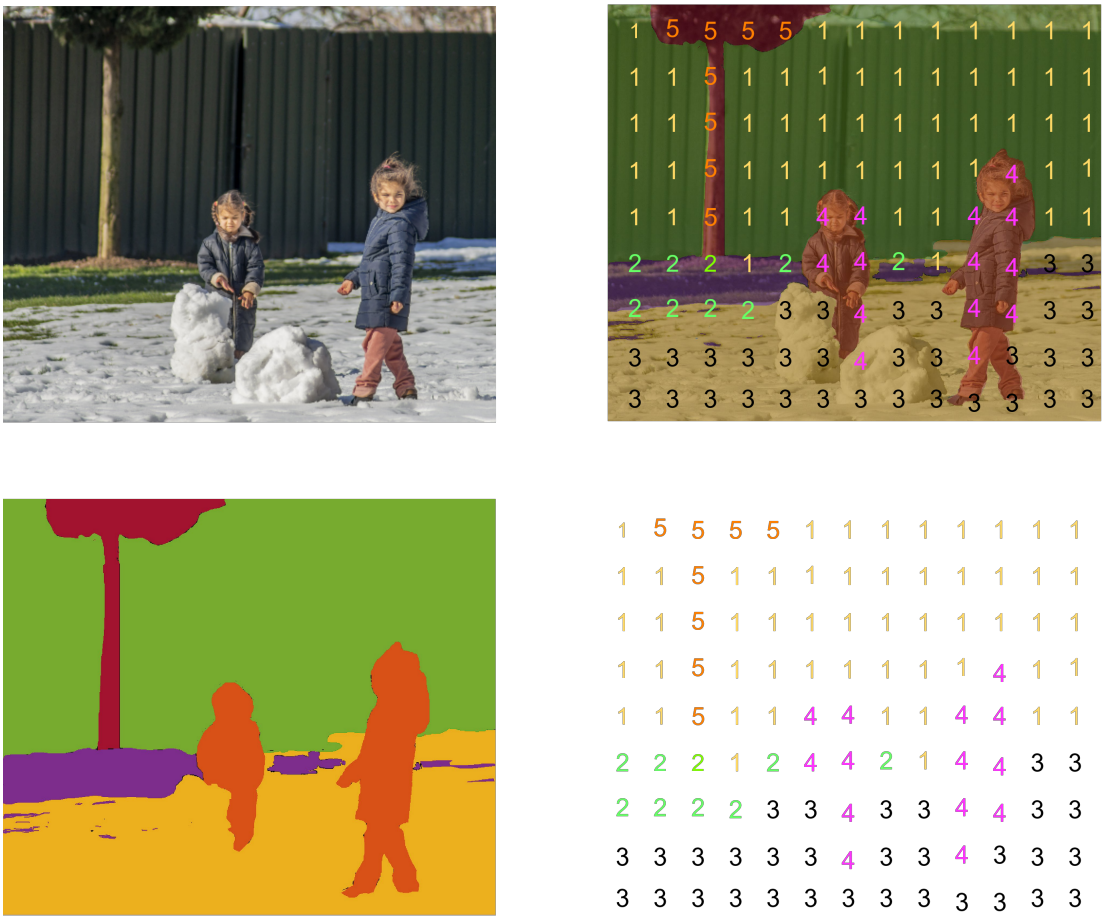

Figure 2.1: This figure shows example of semantic segmentation of an image. Pixels belonging to a particular class are assigned the same label. 1-fence, 2-grass, 3-snow, 4-person, 5-tree.

introduction of deep learning led to the development of algorithms that employed object detection to categorize areas acquired from segmentation based on low and mid-level characteristics [18].

2.2.1 Fully Convolutional Networks

The Fully Convolutional Network (FCN) [42] achieved a major breakthrough in the early deep learning works for semantic segmentation by providing dense class predictions for each pixel. In this work, it was suggested that fully connected layers should be removed from Deep Convolutional Neural Networks (DCNNs) to develop an end-to-end semantic segmentation model. By modifying classification networks so that they only contain fully convolutional layers, an FCN model can generate spatial segmentation maps for images of any resolution. Upsampling the output feature maps to a pixel format of any resolution is accomplished with the help of deconvolutional/transposed convolutional layers of

this model. These layers make up an essential part of the decoding phase of this paradigm. They are responsible for regaining the initial spatial dimensions of the image and communicating the ”where” information contained within it. Aditionally, it suggests making use of skip connections across layers that are not contiguous to one another. To prevent any information from being lost due to max-pooling layers or dropouts, the feature maps of layers that are not related to one another are upsampled and concatenated.

FCNs are one of the early trademark models in semantic segmentation architectures. This approach shows how information from successful classification networks may be applied to other computer vision applications. Fully convolutional layers have fewer parameters and marginal weights than fully connected layers, resulting in faster training and inference time for any image. In practical applications such as medical image segmentation and autonomous driving, more refined localization of class labels is required in addition to accurate pixel-level predictions. However, there have been many subsequent approaches based on the idea of fully connected layers addressing the drawbacks of FCNs, such as high inference times and inefficiency in processing global contextual information generating coarse segmentation maps.

2.2.2 Encoder-Decoder Architectures

The encoder-decoder models were proposed to improve the idea of deconvolutional layers and skip connections in the FCN architecture. In the Encoder-Decoder architectures, the fundamental concept is to combine a contracting route (encoder) that extracts the ”what” information with a laterally applied expanding path (decoder) that retrieves the ”where” information such as object details and image size. Convolutional and downsampling techniques are used to train the encoder-decoder networks on the latent space representations. Following that, these representations are decoded using upsampling and deconvolutional processes.

Deconvnet [46] was designed in response to the inference that, despite the use of transposed convolutional layers for FCN upsampling, there was a limitation in learning true deconvolutions to provide a finer segmentation output. Deconvnet proposes the concept of learning a multilayer deconvolution network with VGGNet serving as the encoder backbone. The input image may be reconstructed from its feature representation using the deconvolution network, and these learned filters enable capturing class-specific shape information. Thus, encoder-decoders generate abstract hierarchical features with high localization precision. Individual object proposals are applied to the trained network to get instance-wise segmentation, concatenated to form the final semantic segmentation.

U-Net [59], on the other hand, establishes additional shortcut connections between these two modules. Simultaneously, these shortcuts allow the omission of certain layers while back-propagating the errors during training, avoiding issues such as vanishing gradients. This design was developed initially for biomedical image segmentation and worked well even when the training data is scarce. Badrinarayanan et al. proposed SegNet [2], a technique very similar to the U-Net. The primary distinction

is SegNet does not pass whole feature maps from the encoding to the decoding phase. This approach transfers the pooling indices with the maximum value without using deconvolution operations.

2.2.3 Multi-Scale, Context Module, and Feature Fusion Based Models

A different line of the study suggests input multi-scaling to resolve the classification or localization dilemma while capturing both broad context and minute details. Semantic segmentation may be accomplished effectively by combining features extracted from different input scales, enabling the network to access data at various degrees of detail. This strategy is used in Farabet [13] and Lin [38]. However, this sort of Image Pyramid technique in semantic segmentation has a significant disadvantage: it does not scale well for DCNNs owing to issues such as GPU memory constraints.

Numerous strategies in this field have concentrated on enhancing the details of the image at the expense of losing context at various phases. When the global context explains local misunderstanding, a considerable amount of misclassified pixels can be recovered, resulting in a smoother segmentation output. A global contextual module was suggested to be included by ParseNet et al. Wei Liu [41]. This would be accomplished by enhancing the features at each location with the average feature for each layer and then running segmentation on the resulting feature map. Hariharan et al. [22] propose Spatial Pyramid Pooling layers that do the pooling operation using different kernel/stride sizes to the feature maps and then flatten and concatenate to make fixed-length representations. Atrous Spatial Pyramid Pooling module having a pyramid multiple dilation rates is incorporated in the Deeplab architectures [4].

PSPNet [88] uses ResNet with dilations as a feature extractor. The feature maps generated at various scales corresponding to a pyramid level are upsampled and concatenated along the channel dimension. This alleviates the learning ability of a scene's global context representation. Though PSPNet produces good accuracy, it has a very high computational cost, almost as heavy as fully convolutional neural networks. BiseNet [82] uses two branches starting from the encoder with two different resolutions(i.e., high and low) at each branch processed separately. Wider channels and shallower layers of the network process the high-resolution image to capture the spatial details, whereas narrower channels and deeper layers process the low-resolution image to capture high-level or global information. These two branches are connected through attention refinement modules and feature fusion modules.

Recent literature [31, 37, 77, 10], propose adding shortcut connections, dual path decoder with substage aggregation, multi-path refinement, and attention pyramid modules to extract dense features in the network. Other works presented projections from encoder feature space integrating interleaved pyramid fusion modules to improve segmentation results [49, 50]. However, all these networks have complex architectures and are not compressed enough to be deployed on edge devices.

Different classes of deep learning-based semantic segmentation architectures have been proposed, as discussed above. Most of the models follow the fully convolutional networks (FCNs) [42] approach. Early works designing convolutional neural network architectures for semantic segmentation concentrated on accuracy (weighted IOU). Most of the semantic segmentation models follow an Encoder-

Decoder type of architecture. In the encoder part of these networks, the feature extractors are robust object detectors like ResNet, ResNext, etc. PSPNet [88] achieves accuracies above 80%. However PSPNet [88], runs at more than 100 GFLOPs. Our work is more focused on obtaining models with < 20 GFLOPs.

2.3 Model Compression

Deep learning methods produce remarkable results at the cost of enormous size and computation overhead. This means that deep models cannot be trained without considerable computational power and cannot be deployed on low-powered edge devices. Model compression methods try to identify the redundant connections in large neural networks and prune them, resulting in smaller networks with comparable performance. Studies have shown that starting from an over-parameterized model and removing redundant connections gives better results than starting from a small model [15].

Compressing deep models is beneficial for several reasons. They result in small models with low computational requirements, which means they can be run on low-powered devices. This ensures that they can be deployed in applications like mobile phones, self-driving cars, and embedded systems. Smaller networks also allow training at lower costs and without specialized hardware.

Model compression techniques can be divided into two broad areas: explicit model compression and efficient convolutional filter designs. Explicit model compression works on the premise that neural networks are over-parameterized. The process involves finding redundant parameters and pruning them to obtain a smaller model with similar performance as the original one. Another technique to obtain small networks is to redesign the convolutional layers which have fewer parameters carefully. Efficient architecture design aims to make the architecture compact apriori.

In the following section, we delve into the different techniques of model compression and their working.

2.3.1 Efficient Pruning Techniques

Neural network pruning is a model compression technique that optimizes the model for real-time inference for resource-constrained devices. The redundant elements of a network are found and removed to leave a compact and efficient network. The pruning may happen at different scales. In filter pruning [36], entire convolutional filters are removed at once. This is usually achieved by ranking the filters in order of importance according to some measure and removing the least essential filters. This technique can be applied, which results in high compression ratios. Other techniques are to remove either single neurons or weights at a time. This results in better accuracy as we can be more precise with what is removed from the network.

Son Han [21] introduced weight pruning and neuron pruning, where they significantly reduced the parameter count without obtaining a loss in accuracy. In weight pruning, the matrices are made sparse

by dropping the weights which are 0 after training. Pruning low-weight connections involve removing weights below a certain threshold. The outcome of ensuring sparsity in compressing neural networks is that sparse matrices can be stored in an optimized way, i.e., storage in HDD is efficient.

However, sparse matrices take up the same amount of memory in RAM; therefore optimal sparse matrix multiplication algorithms have to be written from scratch for even the most fundamental operations, such as forward pass operations. Wei Pan [51] introduced regularisers to help in the process of dropping neurons during the training of a neural network. The neurons, along with the incoming and outgoing weights, are dropped permanently resulting in drastic neural network compression.

Another way to classify pruning techniques is the time at which pruning is done. Many works take a fully trained model as input and prune it to a desired sparsity and accuracy. Other works change the objective function while training the networks so that it naturally results in a model with sparse weights. We have adopted this approach in this thesis. There have been a few works which prune the model before it has been trained, with data-centric techniques. We discuss these techniques in detail below.

Quantization is another way to handle the model storage problem by further compressing the parameters of CNNs. While pruning algorithms are used to achieve model compression by lowering the total number of weights, the idea of quantization is to decrease the size of the weights that are already present. In Binary Quantization, Gong et al. [19] quantize the sign of the parameter matrix by making them either +1 or -1. This drastically compresses the size by 32x as each neuron is represented by 1 bit. Binary matrix multiplications have a run time that is seven times better, but the accuracy loss that comes with using this approach is unacceptably high. Another method, called 8-bit uniform quantization, splits the maximum and minimum weight values into 256 equally spaced divisions in a uniform manner. Following this step, the weights are rounded to the closest point, and then they are stored as 8-bit integers. This technique may reduce the data size by a factor of four, and its execution time is much less for 8-bit matrices. The decrease in accuracy seen with 8-bit uniform quantization is comparable to the accuracy drop experienced with binary quantization. The K-means clustering algorithm is applied to the weights in non-uniform quantization or weight sharing. It is necessary to maintain a mapping between integers and cluster centres. To code the clusters, we only require log (k) bits resulting in a compression factor rate of 32/log (k). The compression rate in this instance is equal to 4 [20].

2.3.1.1 Pruning Pre-trained Models

These pruning methods start with a pre-trained network, and then identify and remove redundant connections and fine-tune the network. The last two steps are repeated until the desired sparsity and accuracy are reached. There are several methods for detecting redundant parameters. In [36], the magnitude of the filter is used as a measure of filter relevance, and filters with low magnitude are pruned. Other works use the same criterion but for weights [21]. In Play and Prune [66], importance is given to choosing the correct rate of pruning while the criteria for choosing filters is l_1 norm-based. In [55], each convolutional filter is visualized using activation maximization. The redundant filters are detected by comparing the similarity of filter visualizations and removed.

2.3.1.2 Pruning while Training

Pruning while training is also called dynamic pruning. In these methods, the training objective is modified so that the training procedure automatically produces sparse weights. In addition to the training loss, the objective involves a regularization parameter that favours sparse weights. A commonly used regularization function is the l_1 norm of the weights. This function tries to push individual weights to zero. However, they are not very helpful in pruning entire neurons. Thus some works such as [1] and [2] use an additional group lasso [83][64] constraint, which pushes entire groups of weights to zero at once. The group lasso regularizer is defined as $\sum_{g \in G} \sqrt{d_g} \|\beta_g\|_2$ where g represents a group, d_g is the number of elements in the group and G is the set of all groups. β_g is the regression coefficient of the group. [90] uses an $l_{2,0}$ norm-based regularizer instead of an l_1 norm. A different approach to pruning while training is proposed by [12], based on the ideas proposed in [15].

2.3.1.3 Pruning Untrained Models

Some works prune models with completely random weights. The premise of such algorithms is that the performance of a subnet is based on the type of connections and architecture. These algorithms use a single pass through the dataset to identify an optimum subnet for the dataset. The subnet is then trained generally with the given dataset. Three such works are [76] [35] and [74]. In the work "Pruning From Scratch", [76], the authors take an uninitialized model with random weights and add scalar 'gates' to each channel, whose value is multiplied into the layer output. They adopt sub-gradient descent to change the gates while keeping the model weights unmodified. The channels are pruned based on the gate values.

2.4 Need for Efficiency in Semantic Segmentation

Although there have been a plethora of high-performance segmentation models based on deep learning, the necessity for efficiency and the demand for lower inference time is still vital in resource-constrained environments. Real-world applications like intelligent systems [14], autonomous driving [62], and medical diagnosis [75] require an extremely accurate grasp of the semantic information at each pixel level in addition to the ability to make quick decisions.

2.4.1 Real-Time Semantic Segmentation

Real-time semantic segmentation is most essential and valuable for autonomous navigation of vehicles. For self-driving cars, the most important goal is to learn about their environment and adapt accordingly. When operating in an unfamiliar area, the ability of a vehicle to comprehend the situation in real-time is critical. For this problem, real-time semantic segmentation presents a solution that

uses image pixel-level categorization in many semantic categories, such as cars, pedestrians, and traffic signals to meet the majority of vehicle demands in an integrated manner.

There are many challenges on the road to achieving the target of deploying neural networks on scale. First and foremost, we need to make them power efficient. Models like AlexNet use tremendous amounts of power for simply a forward pass. There is a real need to develop efficient deep networks, which are extremely compact (under 10MBs at runtime) and will ensure low power consumption. They also have to be very fast, and essentially requiring a very meager amount of FLOPs per forward pass. Also, these features are essential since any image-based application, unlike Alexa etc., cannot leverage cloud services for multiple reasons. Requirements for processing are often real-time; collecting images is a major issue since it violates the privacy of nearby people, and it is simply impossible currently to provide a large enough bandwidth to everyone that is capable of transferring image data with deficient power consumption on-the-fly. Hence, compressing neural network models to run locally, power efficiently, and in real-time has been an important objective pursued as a research problem.

PSPNet which produces high accuracy has 65.7 million parameters and runs at 1FPS which is extremely inefficient. More recent works that focus on real-time efficient segmentation are ERFNet [58], ENet [52], ICNet [87], SegNet basic [2] and Clockwork FCNs [63]. All these networks propose different architectural modifications targeting the optimization of different network aspects. Enet [52] is one among the first to show efficient the design principles for designing real-time segmentation networks. However, ENet has only 0.4 million parameters and runs in real-time, but the accuracy is significantly low compared to PSPNet.

2.5 Datasets and Evaluation Metrics

2.5.1 Urban Street Semantic Image Data Sets

Citiscapes [8] is one of the most popular datasets for segmentation. It is large-scale image dataset which consists of scenes of streets and urban landscapes. Citiscapes includes detailed semantic segmentation annotation for each image. Around 25,000 images are collected from 50 cities and taken at different conditions of lighting and seasons. The images are annotated with 30 classes, of which class labels exist for 19 classes. The rest have instance annotations. These classes are organized into the following eight categories: flat, building, nature, vehicle, sky, object, person, and void. The dataset provides fine annotation for 5000 images and coarse annotation (category-level) for the rest of the images. As there are two distinct semantic granularities, namely classes and categories, we present two distinct mean performance scores and refer to them as IoUcategory (8 categories) and IoUclass (19 classes) respectively.

CamVid, KITTI, and SYNTHIA are a few examples of other image data sets that may be used for the semantic segmentation of urban street data. These tend to be dominated by the Cityscapes picture collection for a variety of reasons, the most important being that the Cityscapes set is far larger. It is

one of the most extensively used datasets, where state-of-the-art semantic segmentation architectures benchmark themselves against it.

The only image collection that can be termed largescale is the SYNTHIA image set, which has more than 13,000 annotated photos. However, the SYNTHIA image set was created artificially, a significant constraint for safety-critical systems such as autonomous vehicles.

2.5.2 Performance Evaluation

In this section, we discuss the various metrics to evaluate the performance of semantic segmentation models. The following is a concise description of the primary measurements that are most often used in assessing the efficiency of semantic segmentation models during test and validation.

2.5.2.1 Accuracy

Accuracy is the measure of how faithful the model prediction is to the ground truth, usually provided by a human. To evaluate semantic segmentation, we commonly use mean pixel accuracy and intersection over union, which we define below.

Pixel accuracy (PA), is the ratio of the number of correctly identified pixels to the total number of pixels. Here, x_{ab} stands for a pixel that belongs to class a but is predicted as belonging to class b. Thus, x_{ij} stands for the false positives, x_{ji} stands for the false negatives and x_{ii} are the correctly classified pixels. c is the number of semantic classes.

$$PA = \frac{\sum_{i=1}^{c} x_{ii}}{\sum_{i=1}^{c} \sum_{j=1}^{c} x_{ij}}$$

The Mean Pixel Accuracy(mPA) determines the ratio of accurate pixels on a per-class basis and then takes that ratio and averages it out across the whole number of classes.

$$mPA = \frac{1}{c} \sum_{i=1}^{c} \frac{x_{ii}}{\sum_{j=1}^{c} x_{ij}}$$

When the classes are severely unbalanced, one or more classes predominate the image, while other classes comprise just a minor fraction of the whole image. Pixel accuracy is a relatively simple technique, but it is also quite susceptible to being skewed by classes that occupy a significant amount of the image.

A more popular evaluation measure for segmentation is Intersection over Union (IoU). It does not suffer from the drawbacks of pixel accuracy. IoU is calculated per class by counting the number of 'true positive' pixels (those pixels correctly predicted to be in a class c) and dividing it by the number of pixels in class c as well as all pixels predicted to be in class c.

$$IoU = \frac{\sum_{i=1}^{c} x_{ii}}{\sum_{i=1}^{c} \sum_{j=1}^{c} x_{ij} + \sum_{i=1}^{c} \sum_{j=1}^{c} x_{ji} - \sum_{i=1}^{c} x_{ii}}$$

Mean IoU (mIoU) is calculated as the overall class weighted average of the IoU. mIoU is the most efficient metric used to evaluate the models performance reflecting how well the model performs across all the classes.

$$mIoU = \frac{1}{c}\sum_{i=1}^{c}\frac{x_{ii}}{\sum_{j=1}^{c} x_{ij} + \sum_{j=}^{c} x_{ji} - x_{ii}}$$

As seen from the above definitions, the Pixel Accuracy does not take into account the false positives and hence, the IoU metric provides more accurate scores than the pixel accuracy.

2.5.2.2 Computational Complexity

The computational efficiency of networks is measured primarily based on two metrics: the speed at which the deep neural network runs and the amount of memory that the model occupies.

- **Inference time**: Execution time for a frame is assessed as the total amount of time required to process an image. This time is referred to as the inference time. The performance of the inference time metric is heavily dependent on the hardware that is being used for each deep neural network architecture. Every measure of an algorithm's execution time (i.e the model of GPU's used while running the network) is listed with a comprehensive explanation of the hardware being used for state-of-the-art comparison.

- **Frames per second**: This is an industry-standard statistic used to evaluate the amount of time required for a neural network to evaluate a sequence of image frames from the given test data source. It is the inverse of inference time. It is essential to have a precise understanding of the amount of frames that a model can handle in less than one second for applications requiring real-time processing. Most real-time segmentation networks report their performance based on frames per second (FPS). To calculate the FPS for our models, we set the batch size to 1 during test time.

- **Measures of efficiency**: The memory of the model may be calculated as the disk space, which is the amount of parameters in the model or the space the network takes up when loaded. In this work, we use the first definition, i.e. parameter count in the model to quantify the level of computational burden. Another related measure is the computational complexity of the model, measured in FLOPS (floating-point operations). We present the computational complexity of our proposed model in giga-FLOPS (GFLOPS).

2.5.3 Loss Function for Semantic Segmentation

In order to obtain pixel-wise categorization in semantic segmentation, we employ the use of categorical cross-entropy loss function between predicted and target class distribution of pixels. The pixel-wise loss is determined by adding up the log losses of all of the relevant classes. This score is iterated through all of the pixels, and then the average is calculated. This is different from L1, L2 loss functions that most regression models use.

$$-\sum_{\text{classes}} y_{\text{true}} \log\left(y_{\text{pred}}\right)$$

Because the pixel vector predictions for each class are analyzed by the cross entropy loss on an individual basis before averaging the results across all pixels, we are effectively claiming that each pixel in the picture has access to an equal amount of learning. Because of this, training may be dominated by the class that is the most frequent in the picture, which may be an issue if the representation of various classes in the image is not balanced. Long et al. [42] proposes the strategy of assigning weights for individual output channels in the loss function to resolve the issue of disparity amongst the classes within the dataset. Ronneberger et al. [59] provide a loss weighing strategy for each pixel in U-Net where they give a precalculated weight map. This system gives a more significant weight to the pixels that are located around the borders of the segmented objects. Dice loss [68] is another loss function used in combination with the pixel-wise cross entropy loss as solution for the class disproportion issue in semantic segmentation.

Chapter *3*

Designing Efficient Convolutional layers for Semantic Segmentation

Our work mainly focuses on designing efficient semantic segmentation architectures and training methods that enhance computational efficiency. There are two significant contributions that we introduce in this chapter. Firstly, we introduce three new convolutional modules using the concept of convolutional factorization. Based on these proposed modules, we achieve compressed segmentation network structures. We design compact neural network architectures for real-time semantic segmentation from scratch that are end-to-end trainable, achieving an optimal balance between accuracy and computational complexity. We evaluate the effectiveness of our proposed convolutional layers by implementing them in two distinct encoder backbones: ESSNetA and ESSNetB. Secondly, we present a new methodology to compress a pre-trained network distinct from the conventional model compression approaches. We propose to obtain highly efficient semantic segmentation architectures by devising specialized training techniques for grouped convolutions.

Segmentation architectures in the era of deep learning can be divided into high accuracy networks and high inference-speed networks. Most performance-oriented, highly accurate architectures such as Unet[59], DeepLab architectures[4][7], and DeConvNet[46] are ponderous, have compound architectures with heavy computational overhead. Whereas high inference-speed architectures such as ENet[52], ICNet[87] and SQ Net[72] are highly efficient in terms of size and frames per second (FPS), but they sacrifice performance. A significant amount of work has been put into enhancing the accuracy of conventional models with substantial computational requirements even in the testing stage. Nevertheless, the networks themselves need to be simplified for these models to be implemented into compact devices and in various portable devices for scene understanding applications. Such efficacy can be accomplished by using more concise models; by applying efficient convolutional blocks and optimizing the training of a complex model. Optimization during training is achieved by compressing the learned information into a more compact model; achieving a memory-efficient network during the test time that reproduces the complex model.

The holy grail of semantic segmentation research is to attain a better balance between model size and accuracy. In this chapter, we analyze the different architecture components used in various state-of-the-art real-time semantic segmentation models, and we further validate the effectiveness of each design

component. In addition, we propose network design variants that minimize the model size without degradation in accuracy along with inference times feasible for portable embedded real-time solutions.

We design a highly compact model ESSNet having 5.78 GFLOPs and 0.43 million parameters, resulting in 68.4% segmentation accuracy. We also propose two other variants of the real-time segmentation networks that give 1.5x, 2x reduction in FLOPs compared to the baseline architecture with only 0%, 2% reduction in accuracies, respectively. The experiments demonstrate that our approach delivers cutting-edge results in terms of parameters and accuracy trade-off for real-time semantic segmentation on the Cityscapes dataset. In this chapter, our effort is more oriented towards designing efficient and effective CNN modules in alignment with model compression methodologies by fixing the macro architecture design parameters. In the next chapter, we venture with architecture design elements such as additional context extraction modules, varying filter kernel sizes in each layer, symmetry of the architecture, and effective receptive field of the network.

We propose a novel training methodology called Gradual Grouping, which prunes the superfluous connections between layers of a dense network. In this method, we propose to gradually compress a model during training, and the resultant sparse model is implemented efficiently at test time. Our approach is inspired by lifting methods in linear programming, where better optimization is feasible in a higher-dimensional representation. We choose ERFNet [58] as our baseline architecture which is already an efficient and real-time model, and we draw comparisons to our proposed models concerning this architecture.

In section 3.2, we describe the efficient CNN layer designs that we use. Finally, in Section 3.5, we describe our novel training procedure.

3.1 Design Elements for Efficient CNN Design

In order to lessen the burden of extensive computation of a standard convolutional layer, many heuristics have been incorporated which are discussed in this section. A more recent approach to model compression is to design the architecture with specific insights about the information flow required to give accurate predictions. We leverage the efficient convolutional design techniques that were proven to be successful for image classification and incorporate them into segmentation networks. In this section, we explain all these approaches in detail.

3.1.1 Filter kernel factorization

Filter kernel factorization transforms conventional convolutional layers, which are full-rank matrices into more computationally efficient low-rank matrices that can be approximated as linear summation of basis vectors [29]. Learning models with these efficient representations and avoiding redundant parameters helps to reduce over-fitting, boost generalization, and ultimately increase accuracy.

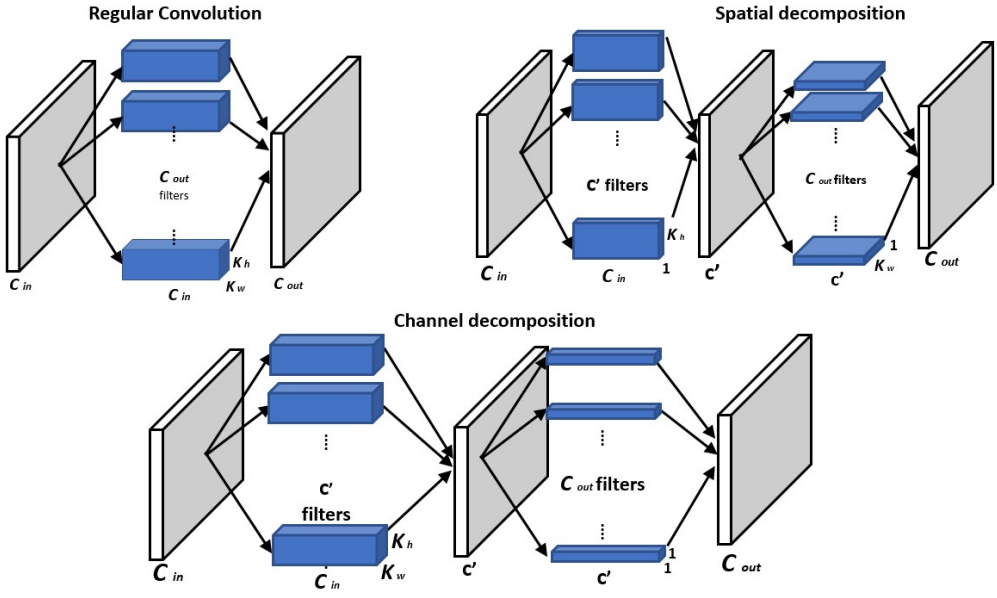

Figure 3.1: Figure shows two factorization methods: spatial decomposition and channel decomposition. The feature maps are shown in gray, whereas the convolutional weight matrix (filter kernel) is shown in blue. The first figure shows a regular convolution, where the kernel is 3-dimensional. The 2-D filter kernel is factorized in spatial decomposition into two 1-D kernels. For channel decomposition, the weights tensor is decomposed into depth-wise convolutional weights and point-wise convolutional weights respectively as shown in the bottom part of figure [24].

In this section, we discuss different types of kernel factorizations that provide the basis for optimizing computational efficiency of various semantic segmentation architectures. We calculate the number of parameters and FLOPs for each type of convolution and draw a comparison between each of them: regular convolutions, depthwise-separable convolutions and spatially separable convolutions.

At the beginning of the deep learning era, many deep networks used large-size filters in the initial layers. Simonyan et al. [65] demonstrated that a convolutional layer with 7x7 filters could be replaced with sequential stacking of three smaller-sized 3x3 filters along with adding three ReLU layers in between them which uses only half the computational parameters in comparison to larger kernels. Jaderberg et al. [30] have proposed spatial decomposition of a regular convolution by exploiting the redundancy between different channels and filters. They use optimization to reduce the reconstruction error of the full rank filters that are already learned. By taking advantage of channel-wise feature map redundancy, Zhang et al. [85] have proposed channel decomposition to decompose a regular convolution by low-rank matrix decomposition, as seen in Figure 3.1. Ioannou et al. [29] provide an approach to learning convolutional filters that may be formulated as linear combinations of basis filters and training these low-rank filters from scratch, unlike [30].

Let us take $\mathbf{X} \in \mathbb{R}^{H_{in} \times W_{in} \times C_{in}}$ as the incoming feature map and $\mathbf{Y} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}}}$ as the outgoing feature map, where C_{in} and C_{out} are the number of input channels and output channels, H_{in}, W_{in} are the input feature maps spatial dimensions, and H_{out} , W_{out} are the spatial sizes of the output feature maps. $\mathbf{W} \in \mathbb{R}^{C_{in} \times K_w \times K_h \times C_{out}}$ is the weights tensor of the convolutional layer, and the bias is $\mathbf{b} \in \mathbb{R}^{C_{\text{out}}}$. The jth filter in the convolutional layer with index i is $\mathbf{f_i^j} \in \mathbb{R}^{K_w \times K_h}$. A regular convolution layer and the output of filter can be written as:

$$z(u,v)_i^j = \varphi_i \left[b_i^j + \sum_{c=1}^{C_{in}} \sum_{m=-\alpha}^{\alpha} \sum_{n=-\beta}^{\beta} \bar{f}_i^j \cdot \chi(u+m, v+n)_c^T \right] \tag{3.1}$$

$$Y_i = \sum_{j=1}^{C_{out}} H_{out} \times W_{out} \times z(u,v)_i^j \tag{3.2}$$

where $\varphi(\cdot)$ is the non-linearity, α, β are scalar values where $\alpha = (K_w - 1)/2, \beta = (K_h - 1)/2$, $\bar{f}_i^j$ is the 2- dimensional matrix. $\chi(u+m, v+n)_c$ is the analogous 2-dimensional matrix where (u, v) is the center of the feature map with index c and the same filter sizes. Based on equations 3.1 and 3.2, the number FLOPs can be computed as:

$$C_{\text{in}} \times C_{\text{out}} \times K_h \times K_w \times H_{\text{out}} \times W_{\text{out}} \tag{3.3}$$

The conventional convolutional layer complexity quadratically depends on size of the kernel, number of incoming and outgoing channels, spatial extent of the output. Hence, in our architectures, whenever the feature map size reduces by half, the number of channels increases proportionally to maintain the cost. For simplicity, we set $K_w \equiv K_h \equiv K, H_{in} \equiv W_{in} \equiv H_{out} \equiv W_{in} \equiv HW, C_{in} \equiv C_{out} \equiv C$ and we omit the influence of bias. Hence, a regular convolution has $HW^2 \times K^2 \times C^2$ FLOPs and $C^2 \times K^2$ parameters.

3.1.1.1 Spatial Kernel Factorization/Asymmetric Convolution

This method factorizes a two-dimensional kernel into two asymmetric one-dimensional kernels which approximates the original kernel. For example, consider a 3×3 kernel, which requires $9 \times HW^2 \times C^2$ FLOPs. We show that this filter can be decomposed into two filters of dimension 3×1 and 1×3, which requires only $6 \times HW^2 \times C^2$ FLOPs.

Mamalet et al. [43] propose learning separable filters by training networks from scratch that are comprised of consecutive convolutional layers of horizontal and vertical 1-dimensional filters. It has been proposed by Alvarez and Petersson et al. [1] that each N-dimensional kernel may be broken down into N layers of successively smaller 1-dimensional kernels along with non-linearity between the 1D kernels.

In this case, we show decomposition for 2-dimensional kernels implying that any 2D convolution (we take equal kernel size K for simplicity, $\boldsymbol{W}_{\mathbf{2}D} \in \mathbb{R}^{C_{in} \times K \times K \times C_{out}}$) can be factorized into a pair of

2 1-dimensional convolutions($\boldsymbol{W}_{1D} \in \mathbb{R}^{C_{in} \times K \times C_{out}}$). As discussed in the previous section most low-rank approximations [30] are constructed on already trained matrices. Nevertheless, this strategy calls for further fine-tuning, and the filters that are produced may not be separable. By relaxing the rank-1 constraint of the full rank convolutional filter, f_i can be written as a linear combination of a sequence of separable 1D basis filters [67] as shown below.

$$f_i = \sum_{r=1}^{r} \alpha_i^r \bar{h}_i^r \left(\bar{w}_i^r\right)^T \tag{3.4}$$

where $\bar{h}_i^r$ and $(\bar{w}_i^r)$ are vectors of length K, α_i^r is a scalar weight, and r is the rank of f_i. In continuation to the above equations, the factorized layer can be expressed as:

$$z(u,v)_i^j = \varphi_i \left[\sum_{c=1}^{C_{in}} \sum_{-\alpha}^{\alpha} \bar{w}_i^j \varphi_i \left(\sum_{-\beta}^{\beta} \chi(u+m, v+n)_c^T \bar{h}_i^j \right) \right] \tag{3.5}$$

where m is in the range $[-\alpha, \alpha], n$ is in $[-\beta, \beta]$ and the bias is omitted [39]. From the above equations, it turns out that parameter count for asymmetric factorization is $2 \times K \times C^2$ and FLOP count is $2 \times K \times HW^2 \times C^2$. Szegedy et al. [71] in the inception module makes use of the separable asymmetric 3x3 convolutions and demonstrates that the computational cost is 33% cheaper.

3.1.1.2 Depthwise Separable Convolutions

In this method, the cross-channel and spatial correlation are subsequently calculated separately, resulting in a significant reduction in the number of parameters, which in turn results in fewer floating point operations and a faster execution time. Using this factorization we show that compared to a regular convolution, in the case of a 3x3 kernel, there is 9x reduction in FLOPs and the number of parameters.

Depth-wise separable convolutions are a crucial component of interest in many recent efficient deep neural networks like mobilenets[26] and shufflenets[84]. In a standard convolution, features are filtered and combined to form new representations implying that the interchannel and intra-channel computations are carried out in a single step with highly redundant parameters. Depthwise separable convolutions perform the same operation in two phases: spatial convolution, where a single filter is convolved with a single input channel along the spatial dimensions (intra channel) subsequently followed by point-wise convolutions, which are linear channel projections as seen in figure 3.1. Depthwise separable convolutions [57, 6], as seen in Xception network, comprises of a depthwise convolution performed over each channel of an input layer and followed by a 1×1 convolution.

Based on 3.1 and 3.2 we see that in depth-wise phase $C_{\text{in}} \equiv C_{\text{out}} \equiv 1$ with $C_{\text{in}} \times Y_i$ number of outputs, and in point-wise phase f_i^l and χ are both 1 D vector and $K_{\text{w}} \equiv K_{\text{h}} \equiv 1$. The number of FLOPs in the depth-wise separable convolutions are calculated to be:

$$H_{in} \times W_{in} \times C_{in} \times K_w \times K_h + H_{out} \times W_{out} \times C_{in} \times C_{out} \tag{3.6}$$

Hence, this form of factorization needs $K^2 \times HW^2 \times C + HW^2 \times C^2$ FLOPs and $K^2 \times C + C^2$ parameters in total. Compared to a regular convolution, using depthwise separable convolution, we can reduce the computational cost by $1/C + 1/K^2$ [26]. A series of these layers result in a drastic reduction of multiplications as the depth of the network increases. The cardinality of $HW^2 \times K^2$ is in the order of 10^6; hence the reduction factor plays a huge difference in the network.

3.1.2 Grouped Convolutions

Grouped convolution is another way of building structured sparse convolutions. A grouped convolution layer with a group parameter g decreases the parameter and FLOPs of the layer by a factor of g. Grouped convolutions were first used by Alexnet [44] [34]. ResNext[80] uses grouped convolutions in the ResNet convolutional blocks, which otherwise consume a huge number of parameters in the network architecture.

When filter groups are applied, the input feature map channels are divided into g independent groups, and C'/g number of filters are applied to each group, as seen in the figure 3.2. Each filter operates on C_{in}/g portion of the input channels, thus filter dimensions in the channel space reduce from $K_h * K_w * C_{in}$ to $K_h * K_w * C_{in}/g$[3]. Applying filter groups does not change the size of the input and output feature maps, but it does make the model much easier to run and reduce the number of parameters. Depthwise convolutions can also be seen as a particular case of grouped convolutions where the number of groups equals to the number of input channels.

3.1.3 Channel Shuffling

If multiple group convolutions are stacked together, outputs from a certain channel are only derived from a small fraction of input channels. It is clear that outputs from a certain group only relate to the inputs within the group. This property blocks information flow between channel groups and weakens representation. By grouping in 1x1 point-wise convolution, we could reduce the number of parameters consumed, but the information flow across groups is blocked. The input and output channels will be entirely related if we allow group convolution to obtain input data from different groups (see Figure 3.3). Specifically, for the feature map generated from the previous group layer, we can divide the channels in each group into several subgroups, then feed each group in the next layer with different subgroups. This can be efficiently implemented by a channel shuffle operation [84]. The output channel dimension vector is first reshaped, a transpose operation is applied, and the resultant is flattened before passing it to the succeeding layer. Channel shuffling operation is differentiable and can be included in end-to-end training, enabling cross-group information flow for multiple group convolutions.

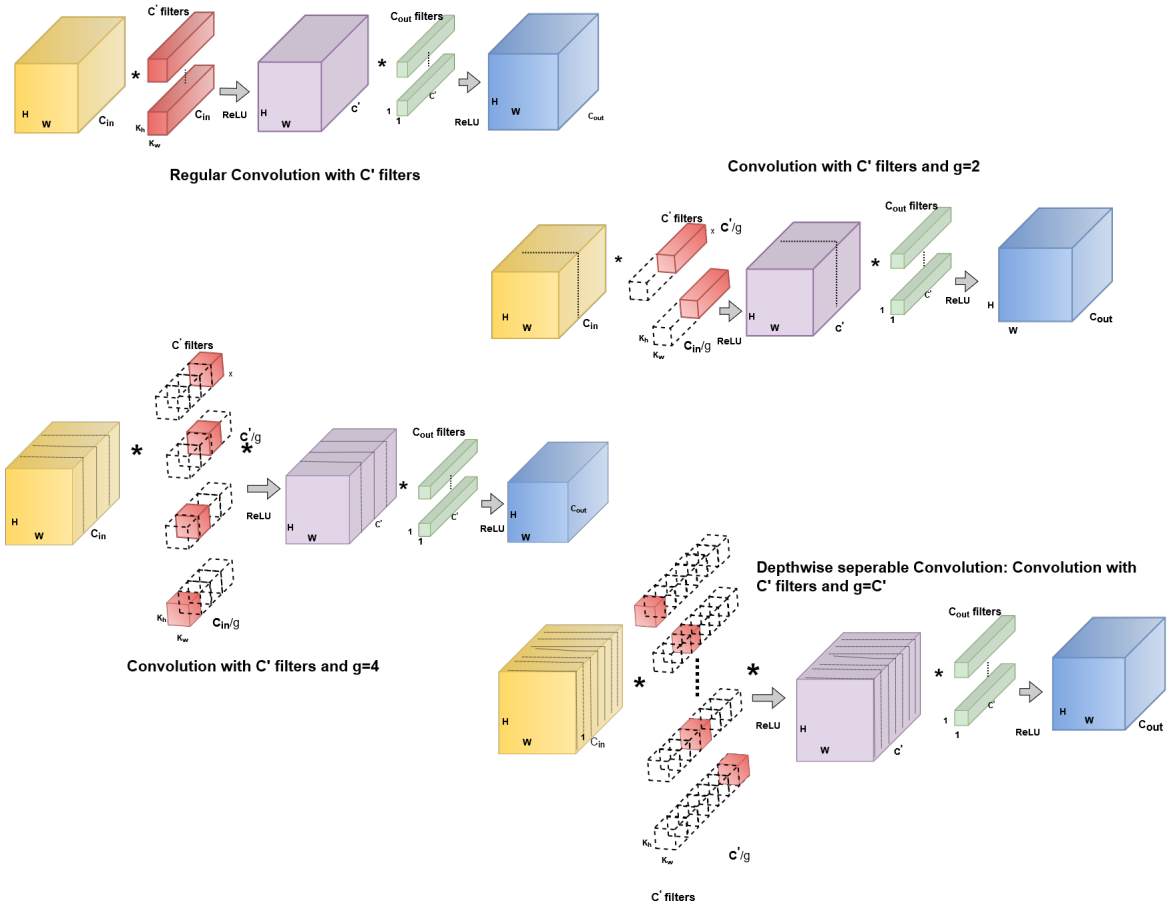

Figure 3.2: Figure demonstrates grouped convolutions with different group sizes. The first part shows regular convolution, and the second part shows when number groups are 2, C' filters are divided into C'/g number of filters and each filter operates on C_{in}/g channels. The third part shows when group size is 4 and the last one is a case of depthwise separable convolutions($g = C_{in}$) where each channel operates on a single filter separately.

3.1.4 1x1 Convolutions

Efficient layer designs started with GoogLeNet [69], which proposed to reduce the input channels to 3x3 convolutions. Xception [6] took it further by using 1x1 convolutions after depth-wise 3x3 separable convolutions. 1x1 convolutions are seen as low dimensional embeddings [40], which compute cross-channel correlation. The new feature maps have lesser channels than the input. This technique is proven to be successful to reduce the computational burden in deeper networks where the number of filters and filter sizes are more [23], [70],[26]. Integrating these blocks reduces the amount of computation to be done by successive spatial filters as they operate on lower dimensional input space.

In SqueezeNet [28] Iandola introduces the design strategy to maintain minimum parameters in each layer, and minimize the number of large-size filters. They introduce squeeze layers to decrease the input channels to more extensive filters and subsequent expand layers. Mobilenet[26] uses 1x1 convolutions to expand the feature maps after the feature transformation step. Inception[71] module uses

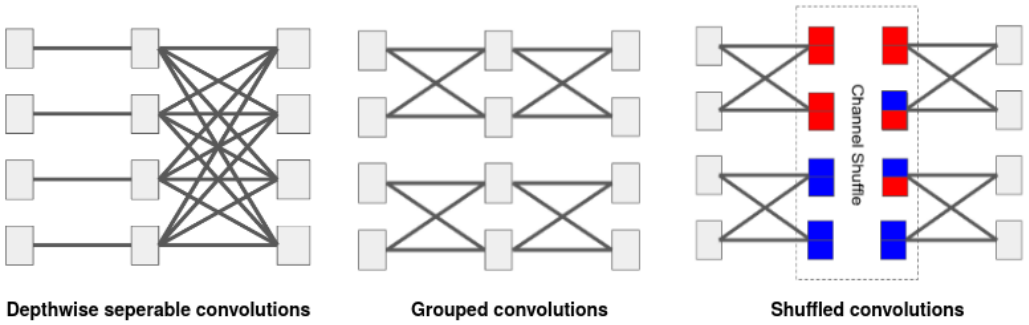

Figure 3.3: Depthwise separable convolutions in the figure show a depthwise convolution performed over each channel of an input layer followed by a 1×1 convolution, grouped convolutions can be thought of as a dense convolution with certain weights zeroed out which is a simple way of having structured sparsity in convolutions, and channel shuffling operation enables cross-group information flow for multiple group convolutions.

the strategy to split, reduce, transform and merge. This module uses 1x1 convolutions in the reduction step. ResNext[80] module performs split, reduce, transform, expand and merge operations. In these redesigned CNN modules, 1x1 convolutions are used in the reduction step or expansion steps. In the later sections, we utilize the functionality of these operations in our convolutional layers and upsampling blocks.

3.1.5 Residual Convolutional Layer

If x denotes the input to a convolutional layer, in a standard convolution, a mapping function $F(x)$ has to be learned to find the appropriate weights (W_c) such that

$$F(x) = M\left(x, W_c\right)$$

whereas in a residual layer, the skip connection optimizes this mapping better where (W_r) are the parameters to be learned and $F(x)$ is written as

$$F(x) = MR\left(x, W_r\right) + x$$

It was shown by [23] that stacking convolutional layers without residual connections cause degradation in deeper networks. Hence we choose to add residual connections in all our proposed layers, as the network configurations have more than 20 layers.

The residual block choice can be either a bottleneck design or a non-bottleneck design, as seen in ResNet[23]. In the bottleneck layer design, the 1x1 convolutions are deployed in the reduce and merge phase. α is the scaling factor by which the number of channels are reduced in the middle 3x3 layer. Hence, in a bottleneck layer, $C_{in} = C_{out} = C_{mid} \times \alpha$. ResNet shows that number of parameters is quadratically proportional to the width(controlled by α) and directly proportional to the depth of the

network. Decreasing the width results in a loss of accuracy, which is compensated by increasing the depth at a lower parameter cost[81]. ENet[52] chooses a bottleneck layer, whereas ERFNet[58] chooses a 1D factorized version of the non-bottleneck layer.

3.2 Proposed Efficient Architecture Design Pipeline

In the above discussion, we have seen various approaches toward efficiency. With the help of those design principles, we make design choices that are suitable for efficiency-oriented architecture implementations. We have designed three convolutional layers: D, DGC, and DGCS as seen in Figure 3.4.

- **D**: We choose a non-bottleneck design[23], having a shortcut between the input and the output of the second 1x1 convolution block. We do a depthwise convolution factorization of the regular 3x3 convolution blocks. There is a significant reduction in computation burden because sharing weights make these separable convolutions quite efficient with improved runtime performance. This convolution factorization, not only helps in model compression but also regularize the network and promote generalization.

- **DGC**: Inside depthwise separable convolutions, most of the FLOPs are found within the 1x1 convolutions making them computational bottlenecks. As we explore the possibilities to compress the model further, we discover that merging grouped convolutions inside the 1x1 convolutions yields a more effective architecture for us. Hence we group the convolutions in a point-wise stage after the depthwise convolution, and the number of groups is a controllable parameter C.

- **DGCS**: We further add channel shuffling [84] to facilitate information flow after filter groups in 1x1 convolutions in the DGC module. In our proposed layer architecture (see DGCS in Figure 3.4), we do a channel shuffle operation after a grouped 1x1 point-wise convolution before passing information to the next convolution block.

Based on these proposed convolutional layers, we introduce three encoder-decoder backbone variants of efficient network architectures: ESSNetA, ESSNetB, and ESSNet .

We study the effectiveness of our proposed residual layers intending to achieve real-time performance, and we validate the placement of these layers. A detailed discussion of these proposed architectures is done in the following sections. We present a comprehensive assessment of efficiency vs performance for all the proposed designs.

3.2.1 Encoder Architecture

In the encoder design, many architectures deploy more convolutional layers after the input image to extract suitable feature vectors with more information. Nevertheless, processing large-size input images with convolutional operations are costly in terms of parameters and FLOPs. In this architecture, we use

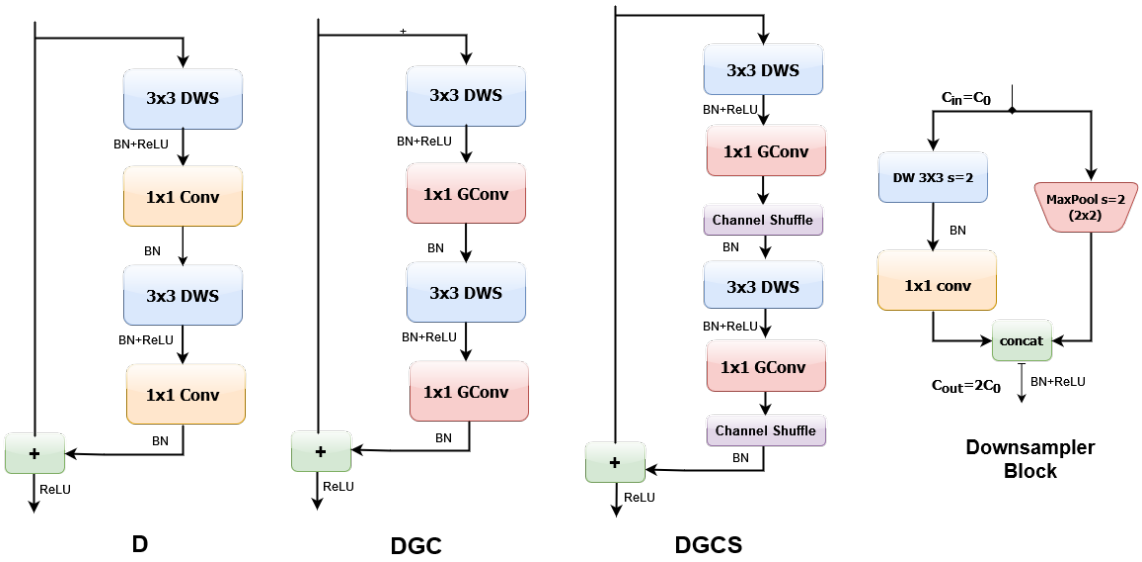

Figure 3.4: Different types of residual layers used in the proposed architecture. D, DGC and and DGCS are our proposed layer architectures and downsampler block.

two downsampling blocks initially to downsample the image before using the convolutional layers and also have small feature map sizes for convolutional operations to improve efficiency.

The downsampling operation in the encoder is the concatenation of the parallel outputs: max pooling layer and a strided convolution operation, as seen in Figure 3.4. As the initial layers are primary feature extractors and visual information processing is done in the later layers, early downsampling plays a key role in achieving an efficient encoder. However, a reduction in feature map resolution would also incur a loss in spatial details that would be difficult to recover in the decoder and thus lower accuracy. We found that using three downsampler blocks in our architecture strikes a good place. The output spatial dimensions at the end of the encoder are one eighth the size of the input image spatial dimensions. We also used dilated convolutions in the convolutional layers with changing dilation factors for gathering more context information in the later part of the encoder.

The encoders of compact semantic segmentation networks are feature extractors that resemble robust object detectors [23, 80]. PSPNet, a popular segmentation network, achieves accuracies above 80% but has more than 100 GFLOPs. Our work is more focused on obtaining models with $<$ 20 GFLOPs. Our encoder in semantic segmentation should be similar to a lightweight image recognition backbone architecture because the increasing the depth of the network increases the cost burden proportionally.

3.2.2 Decoder Architecture

In an Encoder-Decoder architecture, the number of upsampling and downsampling operations should be the same because the final output is a pixel-wise classification map from each input image pixel to a

certain class. The conventional method to increase the spatial dimensions of the feature maps is upsampling through interpolations and unpooling operations. DeconvNet[46], SegNet[2] and ENet[52] use max-unpooling operations in the decoder for upsampling, where the pooling indices are shared from the encoder to the decoder. The drawback of the max unpooling operation is that all the max activation indices from pooling operations must be stored at each level. However, we use transposed convolutions with a stride factor as discussed below, which does not require sharing the pooling indices, simplifying the memory and computational operations[58].

Upsampling in neural network pipeline can be made learnable. This learnable upsampling is known as deconvolution operation or transposed convolution[42]. Compared to a conventional convolution matrix, the relationship between the input and the output is processed in the opposite direction (single input activation to multiple outputs). If the upsampling has to be done by a factor x, then it implies that the convolution operation in the backward direction has input stride 1/x. The regular convolution and the transposed convolution operations create the same connectivity; however with the transposed convolution, the connection is formed reversely. Unlike a predetermined interpolation technique, the weights used in the transposed convolution are learned through end-to-end network training guided by the loss function. The input feature map is upsampled by inserting zeros in between the values. This is done so that the direct convolution achieves the same result as the transposed convolution operation. The stride factor can control the amount of spatial change reversion by the deconvolution operation. However, owing to the need to add zeros in order to up-sample the input before the convolution is still not very efficient.

The prominent role of the decoder is to upsample the output of the encoder by fine-tuning the details and recover the final resolution of the image. Unlike semantic segmentation architectures like SegNet[2] and UNet[59], which have a symmetric encoder-decoder architecture, we use a smaller decoder than the encoder. Nevertheless, Enet[52] is extremely compact but has a lower accuracy because the decoder size is too small, and because of the small size of the receptive field, large objects are not segmented accurately. We have also conducted experiments with the conventional setting of directly upsampling the encoder output using bilinear interpolation instead of a decoder(see section 4.1). As the predictions are at a shallow resolution directly after the encoder and loss function, small objects are not recognized, and the receptive field is not large enough to classify large objects. We do not use skip connections from the encoder to the decoder in our architecture unlike UNet[59], as adding skip connections did not show much improvement in our experiments.

3.3 Baseline Architecture

Our baseline segmentation architecture is inspired by ERFNet[58], considering the balance between FLOP count and accuracy. This model runs on a single GPU with frames per second (FPS) greater than

30 FPS at 0.5 Mpx resolution on the Cityscape dataset, which is good enough for scene understanding applications. ERFNet proposes an efficient convolutional block called Non-Bottleneck-1D layer as the core of the architecture. Most architectures use bottleneck layers because they are more efficient. However, ERFNet chooses to use the non-bottleneck layer, as shown by Szegedy et al. [71] as non-bottleneck layers show performance improvements in shallow architectures like ResNet. This architecture utilizes the concept that the learning capacity of the model increases with wider networks. As seen in the Inception modules [71], changing the bottleneck layers to non-bottleneck increases the parameters by 16x and increases the filters by 4x. Hence, they spatially factorize the non-bottleneck layer resulting in a 33% reduction of parameters, which is a final 11x increase compared to the bottleneck layer. This architecture was proposed as an improvement over the ENet [52], which is highly efficient (runs at $<$ 2GFLOPS) but has low accuracies (57% IOUs). ERFNet model obtains 70% accuracy at 27.7 GFLOPs.

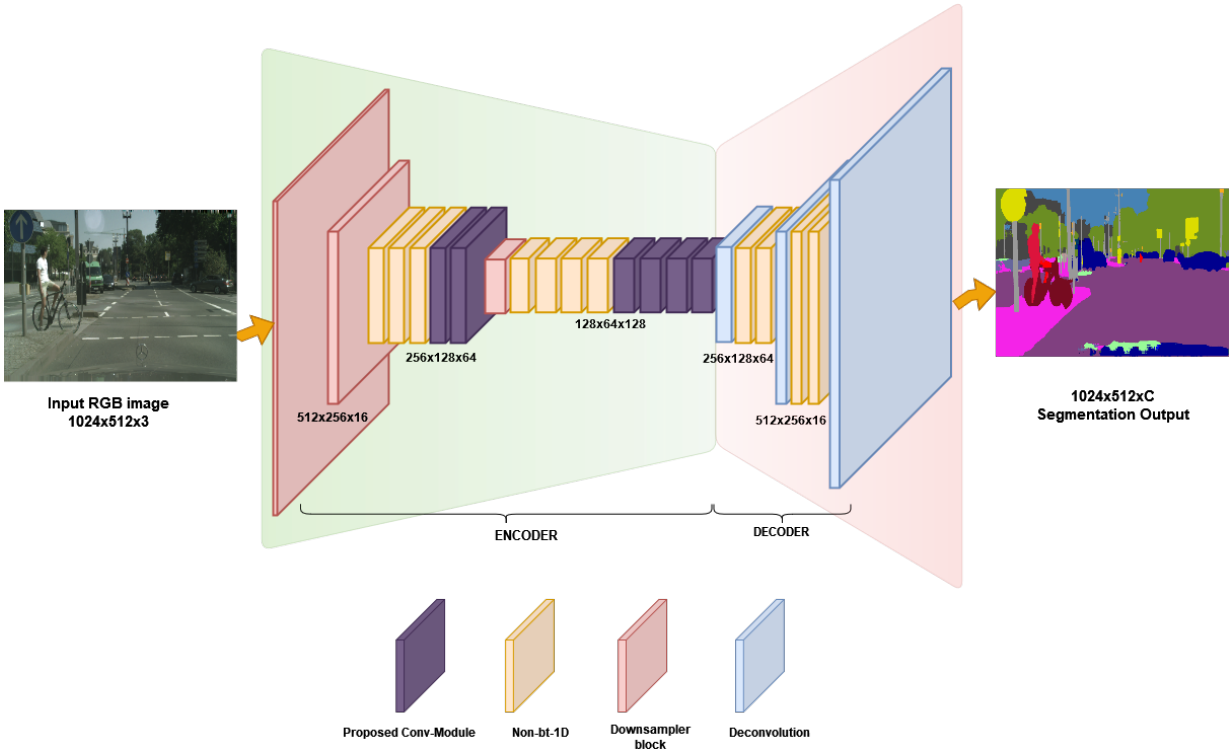

Figure 3.5: Our proposed backbone architecture ESSNetAwith a combination of Non-Bt-1D, and Conv-module layers in the encoder.

3.4 Experiments and Results

Our proposed architectures along with their layer-wise description and the segmentation results are discussed in the following sections. We assess the efficiency of our proposed convolutional layer designs through various experiments.

3.4.1 Implementational Details

We use Cityscapes Dataset [8] as described in the previous chapter in all our experiments. All the models are trained only using the train set. To asses the performance of the architecture, we use Intersection over Union (IoU) scores as an accuracy metric. We report meanIoU, which is the validation accuracy on all the 19 classes of the Cityscapes dataset. We use the Weighted Cross-Entropy Loss function [86][5], which weighs each class to reduce class imbalance, as in equation 3.7. $\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top$, where x is the input, y is the target, w is the weight, C is the number of classes, and N is the batch size.

$$l_n = -\sum_{c=1}^{C} w_c \log \frac{\exp\left(x_{n,c}\right)}{\sum_{i=1}^{C} \exp\left(x_{n,i}\right)} y_{n,c} \tag{3.7}$$

Here, we initialize our weights w_c with a specifically designed class weighing technique as used in [52] to correct the imbalance in data between the various classes, according to the equation 3.8. Unlike the inverse class probability weighing technique, this technique uses weights that are bounded as the probability approaches 0 and the value of γ restricts the class weights. We set the hyper parameter γ to 1.10.

$$w_c = \frac{1}{\ln\left(\gamma + p_c\right)} \tag{3.8}$$

The experiments are done in PyTorch with CUDA 9.0 and CUDNN back ends. Training is done with a batch size inversely proportional to the size of each of the proposed compressed models. L2 regularization is used to avoid over-fitting with a weight decay rate of $2\mathrm{e}^{-4}$. Learning rate of $5\mathrm{e}^{-4}$ and momentum of 0.9 are given as inputs to the Adam optimizer[33]. We use the reduce on plateau learning rate scheduler with a factor of 0.5 in order to accelerate convergence. We use batch normalization between each convolutional layer and following the non-linearity. The two hyper parameters for batch normalization: ϵ and b_{momentum}, are set to $0.001, 0.1$ respectively. We use ReLU[45] non linear activation function in the convolutional modules and downsampler blocks. We use dropout[25] as the regularization while training with a constant of 0.3. As a part of data augmentation strategy at the time of training, we perform random horizontal and vertical flips. We also perform small translations around both the axes and scaling with random factors between 0.5 to 2. The data set is at a resolution of 2048x1024. Our trained model, gives the output class probability map at 1024x512 resolution, but we rescale the output by interpolation to the original dataset resolution in order to validate the proposed models. All the proposed models were implemented on Nvidia GTX 1080Ti GPU.

3.4.2 Encoder-Decoder Training

In order to determine the full extent of our architecture's capabilities, we train the encoder and the decoder in two different processes. First, we train the encoder in its unique way, either from scratch using only the Cityscapes dataset or using the pretrained method and then connecting the decoder so that we may continue training the whole architecture. In the pretrained training approach, the weights of the encoder are initialized from a more extensive dataset such as ImageNet[60]. We achieve this by modifying the encoder's last layers to generate a single prediction instead of multiple classification outputs (1000 classes) as in ImageNet. This is done by adding additional pooling layers and a fully connected layer. After we have finished training this modified encoder, we will get rid of the additional layers, connect the decoder, and do a combined training. In training from scratch approach, we use only the Cityscapes dataset to train the encoder. Since the encoder output is 1/8th the size of the final mapping, we train the encoder using segmentation annotations from Cityscapes that have been downsampled to 1/8th size by attaching a convolutional layer that produces output channel dimension equal to the number of classes. Subsequently, we remove the extra layers and attach the decoder so that we can train end-to-end on the Cityscapes dataset.

3.4.3 Network Configuration

Table 3.1: Proposed Backbone Architecture ESSNetB

	Layer	Type	out-chann	out-Res
ENCODER	1	**Downsampler block**	16	512x256
	2	**Downsampler block**	64	256x128
	3-7	**5 x Conv-module**	64	256x128
	8	**Downsampler block**	128	128x64
	9	**Conv-module**(dilated 2)	128	128x64
	10	**Conv-module**(dilated 4)	128	128x64
	11	**Conv-module**(dilated 8)	128	128x64
	12	**Conv-module**(dilated 16)	128	128x64
	13	**Conv-module**(dilated 2)	128	128x64
	14	**Conv-module**(dilated 4)	128	128x64
	15	**Conv-module**(dilated 8)	128	128x64
	16	**Conv-module**(dilated 16)	128	128x64
DECODER	17	**Deconvolution**(upsampling)	64	256x128
	18-19	**2 x Non-bt-1D**	64	256x128
	20	**Deconvolution**(upsampling)	16	512x256
	21-22	**2 x Non-bt-1D**	16	512x256
	23	**Deconvolution**(upsampling)	C	1024x512

The layer-wise disposal of our proposed backbone ESSNetB is seen in Table 3.1. ESSNetB is constructed by progressively stacking the proposed convolutional layers to optimize their learning performance and efficiency. All the residual layers in the encoder of this network are configured with our

proposed convolutional layers, and downsampler blocks as seen in Figure 3.4. Whereas, the decoder is built with Non-bt-1D layers and upsampling blocks. The Conv-module seen in Table 3.1 is configured with either D, DGC or DGCS layers as described in section 3.2.

ESSNetB-D architecture comprises of proposed layer D throughout the encoder. When the Conv-module in ESSNetB architecture is configured with DGC layer, the network is named as ESSNetB-DGC where C is the value of the number of groups. If Conv-module has DGCS configuration, then the network is named ESSNetB-DGCS, where C indicates the number of groups and S indicates that a shuffle operation is being done. A more detailed architecture of this network describing all the layers as seen in Figure 3.6. Dilation is combined with depthwise separable convolutions, making the resulting layer sparse and increasing the effective receptive field. Each residual layer has a single discrete dilation rate. As these layers with dilation rates 2, 4, 8, and 16 are sequentially stacked, the encoder learns the representations from a large effective receptive field. Upsampler block architecture includes deconvolution layers, simplifying memory and computation requirements. The final layer of the decoder has a volume with the number of channels equal to the number of classes in the dataset, and each channel 1D map in the final layer is per-pixel probability of that respective class. All the results are reported and compared using this nomenclature.

Table 3.2: Results of our proposed convolutional layers applied to ESSNetB backbone architecture

Models	**IOU**	**Params**	**GFLOPs**
ERFNet (Baseline)	70.45	2038448	27.705
ESSNetB-D	68.55	683568	10.597
ESSNetB-DG2	65.35	395568	8.852
ESSNetB-DG4	61.42	319792	7.980
ESSNetB-DG8	59.15	281904	7.543
ESSNetB-DG2S	65.36	395568	8.852
ESSNetB-DG4S	61.27	319792	7.980
ESSNetB-DG8S	59.89	281904	7.543

3.4.4 Comparison with Depthwise Separable, Groups and Shuffle Layers

The performance results along with model size of each network with ESSNetB backbone is seen in Table 3.2. ESSNetB-D model gives an accuracy of 68.55%, having 0.55 million parameters and 10.6 GFlops. This model achieves 3X compression with an accuracy degradation of 2% compared to the baseline model. This model attains a competent balance between model size and accuracy, achieving efficiency.

We further use grouped convolutions to decrease the FLOP count and number of parameters. We use filter groups with varying sizes in the subsequent 1x1 pointwise convolutions as they are the performance bottlenecks with most parameters after the 3x3 depthwise convolutions. This results in the models

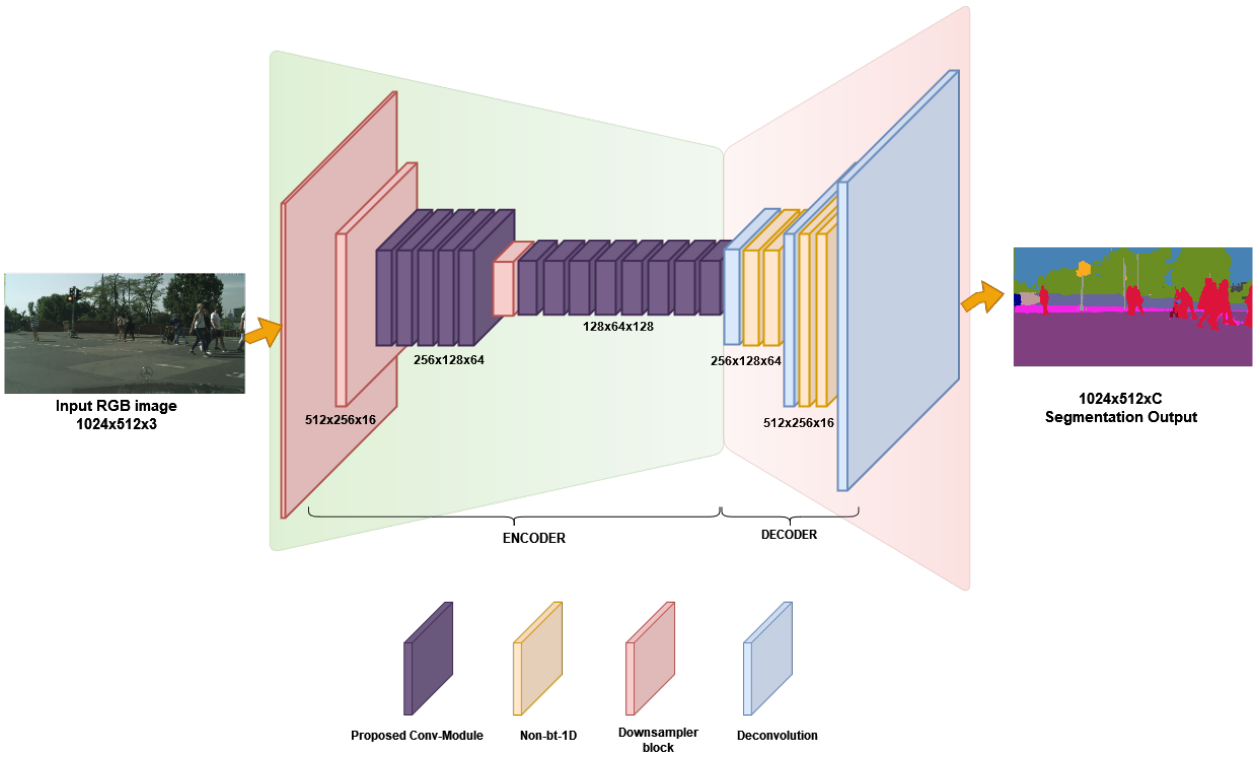

Figure 3.6: Figure shows our proposed backbone architecture-ESSNetB with the convolution module applied throughout the encoder. Our various experiments demonstrate the effectiveness of our proposed Conv-module and the resulting network performance.

ESSNetB-DG2, ESSNetB-DG4, ESSNetB-DG8 with group sizes 2, 4 and 8 respectively. As seen in Table 3.2, we observe the effect of group size on the model performance by keeping the other parameters constant. These models are sufficiently compressed in terms of FLOPs, with the ESSNetB-DG8 model being only 7.5 GFLOPs. However, the accuracy degradation for ESSNetB-DG8 is over 10%, which is likely to be unacceptable. Figure 3.7 shows the segmentation output of our lightweight model ESSNetB-DG8.

We now look at the effect of shuffling on the performance. Shuffled convolutions were affixed to improve the accuracies of grouped convolutions. They have the same parameters and FLOPs as grouped convolutions since shuffle operation is essentially rearranging of the channels. The models ESSNetB-DGCS (where C=2,4,8) are also mentioned in Table 3.2. Nevertheless, we observe that shuffling operation is not affecting the accuracy eminently in our case.

These proposed networks achieve a competitive balance between efficiency and prediction accuracy compared to other real-time semantic segmentation models such as ENet and ERFNet. Besides employing efficient CNN designs, we attain our goal through limiting macro-architecture level design hyperparameters such as network depth, channels per layer, input image size and size of the decoder.

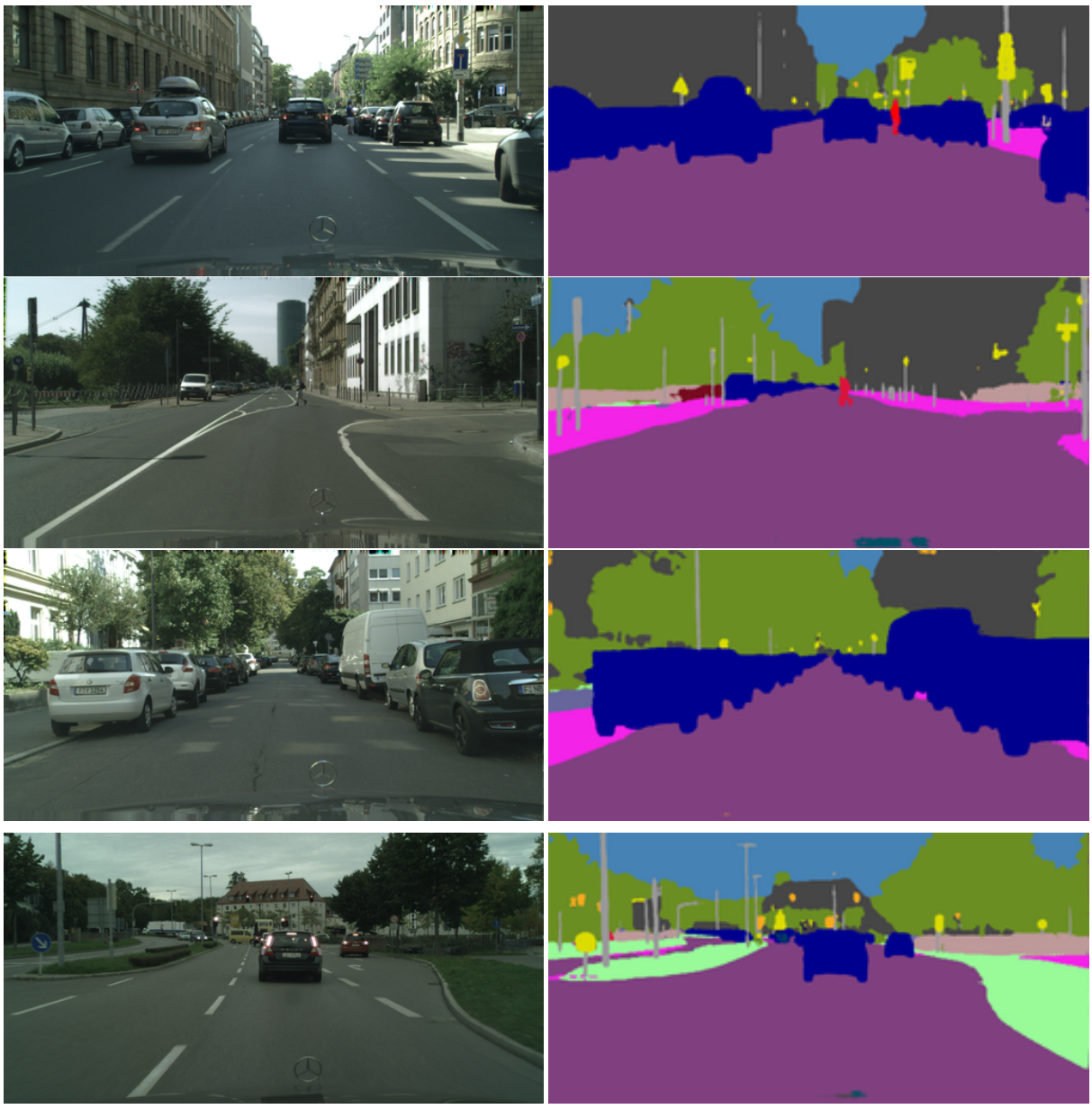

Figure 3.7: Qualitative examples of the validation set and their segmented output from our model ESSNetB-DG8 with group size equal to 8. We see that the segmentation output is not well defined and reflects the accuracy drop. Though the model is lightweight, the disconnectivity between input feature maps and output feature maps increases as the group size increases.

3.4.5 Selective Application of CNN Modules

We devise yet another network architecture, as shown in Figure 3.5, based on the selective application of proposed layers, namely ESSNetA . ESSNetA encoder backbone has a selective fusion of Non-bt-1D layers and our proposed convolutional layers, while the decoder is same as ESSNetB .

As discussed earlier, in ESSNetB networks incorporating compressed layers throughout the encoder results in an accuracy drop. From conventional model compression techniques like pruning and quantization, we adapt the idea to apply compression techniques only to the later layers in the network. In this network architecture ESSNetA seen in Table 3.3, we selectively incorporate our proposed Conv-module layers in the encoder, leaving a few initial layers after the Downsampler block to be Non-bt-1D layers.

Table 3.3: Proposed Backbone Architecture ESSNetA

	Layer	Type	out-chann	out-Res
ENCODER	1	**Downsampler block**	16	512x256
	2	**Downsampler block**	64	256x128
	3-5	**3 x Non-bt-1D**	128	128x64
	5-7	**2 x Conv-module**	64	256x128
	8	**Downsampler block**	128	128x64
	9	**Non-bt-1D**(dilated 2)	128	128x64
	10	**Non-bt-1D**(dilated 4)	128	128x64
	11	**Non-bt-1D**(dilated 8)	128	128x64
	12	**Non-bt-1D**(dilated 16)	128	128x64
	13	**Conv-module**(dilated 2)	128	128x64
	14	**Conv-module**(dilated 4)	128	128x64
	15	**Conv-module**(dilated 8)	128	128x64
	16	**Conv-module**(dilated 16)	128	128x64
DECODER	17	**Deconvolution**(upsampling)	64	256x128
	18-19	**2 x Non-bt-1D**	64	256x128
	20	**Deconvolution**(upsampling)	16	512x256
	21-22	**2 x Non-bt-1D**	16	512x256
	23	**Deconvolution**(upsampling)	C	1024x512

Table 3.4: Results of our proposed convolutional layers applied to ESSNetA backbone architecture - Selective Application of depthwise separable convolutions, grouping and shuffling

Models	IOU	Params	GFlops
ESSNetA-D	69.26	1291648	19.025
ESSNetA-DG2	69.71	1238960	18.998
ESSNetA-DG4	68.98	1202096	18.595
ESSNetA-DG8	69.57	1183664	18.394
ESSNetA-DG2S	70.62	1238960	18.998
ESSNetA-DG4S	69.59	1202096	18.595
ESSNetA-DG8S	69.57	1183664	18.394

When the Conv-module layer is modeled with DGC and DGCS layers (see Fig 3.4), the network is named ESSNetA-DGC, and ESSNetA-DGCS, respectively. Table 3.3 shows the specific positioning of convolutional layers used. Table 3.4 contains the results of these experiments. The networks ESSNetA-DGC refer to models where groups = C is used for selected 1x1 convolutions. We also conducted experiments with the corresponding shuffle convolution versions.

We observe that the model ESSNetA-DG2S without a decoder incurs only a 1% reduction in accuracy while giving a 2X improvement in FLOPs. Also, the model ESSNetA-DG2S gives a 1.5X improvement in FLOPs without incurring any loss in accuracy. As discussed in section 3.1.1, our proposed convolutional layers D and DGC are nearly 3X, 3CX times more efficient than the Non-bt-1D layer. As seen from tables 3.4 and 3.2, ESSNetB has significantly fewer parameters and GFLOPs than ESSNetA because the encoder is entirely configured with our proposed efficient convolutional layers.

We empirically demonstrate the effectiveness of our proposed efficient convolutional layers in real-time semantic segmentation backbones. These proposed CNN modules have shown to be effective and can be generalized to other segmentation networks with residual convolutional layers. We experiment with different group numbers and shuffle operations. We observe that extreme sparsity induced by these techniques can reduce the FLOPs significantly but incurs as much as 10% degradation in accuracies.

3.5 Gradual Training of Grouped Convolutions

Although dense connectivity in non-compressed models allow feature re-use, there are redundant connections where early features are not required in the later layers. Grouped convolution is an effective technique to induce sparsity in the network and thereby significantly reduces the FLOPs. Nevertheless, in highly sparse networks, there is not enough information flow leading to unacceptable reduction in accuracy as seen in section 3.4.4. We propose reducing the redundant weights during training, which enables the resultant model to be implemented efficiently at test time. We introduce a novel training procedure that can be easily implemented, specifically targeting grouped convolutions. In this training process, the model starts as a dense model and gradually evolves towards a lighter model with larger group size. This method allows the gradient descent to happen initially at a higher dimensional model space and gradually evolve towards a lower dimensional subspace of grouped convolutions.

We have reviewed several model compression methods such as pruning, vector quantization, hashing, and shrinking to compress the size of heavy pre-trained networks in the previous chapter. For quite a while in the past, the focus has been on pruning and quantization of networks. Quantizing networks have been using low precision weights; XNORNet[56] uses 1-bit quantization (binary weights) to train networks. By leveraging this, convolutions can be done by fast XNOR-Popcount operations and networks can be accelerated by multiple orders of magnitude occupying significantly less space. Pruning tries to only keep important parts of the network by removing unimportant connections and neurons in

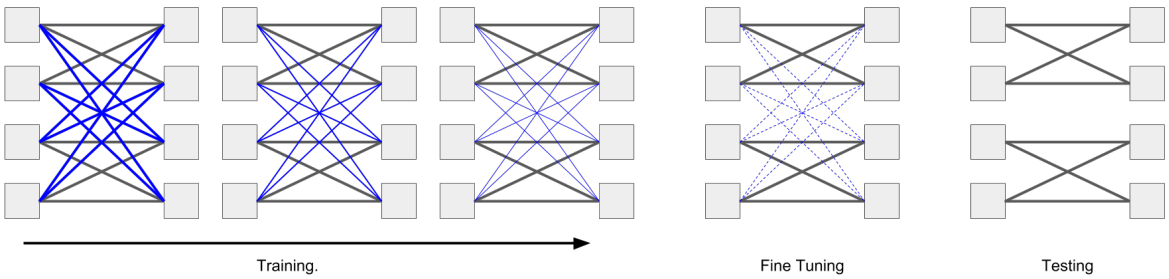

Figure 3.8: Our proposed training procedure for obtaining improved accuracy in grouped convolution-based architectures. A crucial observation is that grouped convolution can be thought of as a dense convolution with multiple weights being 0 (the blue edges). Note that here each edge represents a convolutional filter of $w \times w$. In our method, we start with a dense convolution and multiply the blue edges by a mask matrix. As the training progresses the group structures are dynamically optimized according to the objective loss function of the entire network. We also have a fine-tuning phase where pruned weights remains 0. Finally, at test time, the convolutions can be implemented as a grouped convolution, giving better efficiency. Since the optimization happening at training time is in the higher dimensional space of dense convolutions, we can obtain better accuracy than traditional training for grouped convolutions.

the network, making it compact and fast. These two are the remedial approaches, which take a network design and try to compress it.

There have been works on architecture search [53, 92, 91] where a separate machine learning algorithm is used to drive a heuristic search procedure to pick an efficient architecture. However, these methods require large server farms and have not yet proved their utility for a dense prediction task like semantic segmentation. CondenseNet [27] proposes a simple architecture search procedure integrated with the training of the base network focusing on classification benchmarks. In our work, the architecture is fixed beforehand, unlike [27]. However, we propose novel training algorithm which overcomes the limitation of sparsity, and lack of information flow between the layers giving improved accuracies.

Figure 3.8 shows our proposed training procedure with a special focus on grouped convolutions, which can improve the accuracies. We first observe that grouped convolutions can be considered as dense convolutions with certain weights zeroed out. Hence the space of grouped convolutions is nothing but a linear subspace of dense convolutions. Traditional training procedures start out with a grouped convolution model; hence the gradient descent optimization will only happen in the low dimensional subspace of grouped convolutions. In this training method, we gradually evolve a dense convolution towards a grouped convolution. It is a well-known result in linear programming lifting that optimization in a higher-dimensional space can often lead to convergence towards better minima.

In this proposed training procedure, the network starts out as a model with no groups, which is equivalent to saying that the total number of groups is equal to one and gradually evolves to a model with the number of groups equal to targeted group number. In this training process, the dense connections which we had initially gradually reduce to sparse group connections. At the time of test, the model has connections only within groups and can be implemented as grouped convolution. This reduces the number of FLOPs significantly at the time of validation and testing.

3.5.1 Proposed Training Protocol

In the proposed training procedure, a ESSNetA-D (see Table 3.3), ESSNetB-D encoder model (see Table 3.1) evolve into a ESSNetA-DGC, ESSNetB-DGC encoder model (see Sec 3.4.3), respectively. C is the targeted number of groups. This process trains the encoder and decoder in two phases. The meanIoU value is calculated only after the model settles to a ESSNetA-DGC, ESSNetB-DGC model respectively. The training is done using a controllable parameter alpha. As the value of alpha changes with the increasing number of epochs, the connections become sparse (see Figure 3.8). When the value of alpha is 1, it is a ESSNetA-D, ESSNetB-D model correspondingly. In the initial epochs, alpha gradually decrements from 1 to 0. When the alpha value becomes zero, it is ESSNetA-DGC, ESSNetB-DGC model respectively. In the last few epochs, the model is fine-tuned keeping the alpha value zero, and the model converges. As the grouping techniques are applied only in the encoder, the controllable parameter alpha is used only in training the encoder.

Using these pretrained encoder weights, the encoder decoder architecture is trained. We remove the last layer from the encoder and attach the decoder to train the full network as discussed in Section 3.4.2. Since the encoder model uses pretrained gradual grouping weights, the encoder is well initialized, but decoder weights are not trained. The initialization gained through gradual grouping is lost when the encoder is trained again along with decoder. To overcome this, we almost freeze the encoder, which is equivalent to giving a significantly low learning rate of $5 * e^{-20}$ to the encoder for a few initial epochs. Whereas, the decoder will have a learning rate of $5 * e^{-04}$ in the initial epochs. We start with a learning rate of $5 * e^{-04}$, and a learning rate scheduler is used to decrease the learning rate, so that the convergence is accelerated. In the later epochs, the encoder and decoder start training together with the same learning rate. The proposed novel training procedure can be easily implemented, specifically targeting grouped convolutions.

3.5.2 Validation of Our Proposed Designs

In this section, we validate our proposed method with ESSNet architecture. We further try to reduce the number of FLOPs by attaching a light-weight decoder to the existing encoder model proposed in ESSNetB . In ESSNet backbone, all the residual layers in the encoder and decoder are configured with our proposed convolutional layers. We have applied compression to only the encoder part in ESSNetA and ESSNetB networks. In ESSNet, we apply compression on the decoder by utilizing the efficient Conv-module layers in place of Non-bt-1D layers. We also alter the 3x3 deconvolution (upsampling) operation to 1x1 upsampling. As discussed in Section 3.1.4, we use 1×1 convolutions to generate new features by calculating linear combinations of preexisting ones in the residual layers and also, to generate lower dimensional feature maps in our upsampling blocks. For the predictions to retain detail, ideally, the decoder should restore the resolution of encoded features. We tend to make the upsampling procedure as simple as possible to maintain real-time processing speeds. Thereby, our

proposed ESSNet backbone network is extremely compact compared to the baseline architecture. We train ESSNet encoder using gradual grouping as described in Sections 3.4.2 and 3.5.1.

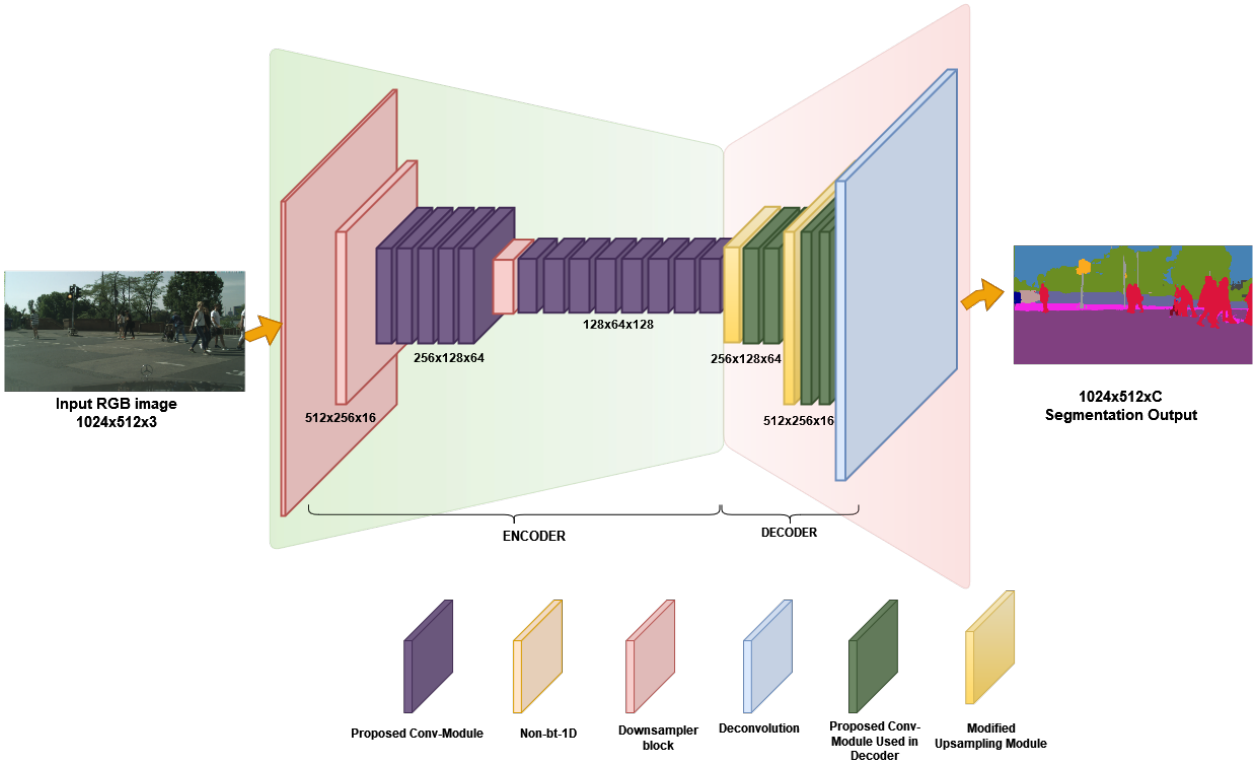

Figure 3.9: Figure shows the application of the proposed Conv-module both in the encoder and the decoder along with the modified upsampling block in our final proposed model: ESSNet

The results of the gradual grouping training are given in Table 3.5. As it can be seen, the FLOPs vs. accuracy trade-off has decreased significantly with our proposed method on the ESSNet architecture. Specifically, the ESSNet-DGC models, with varying group sizes, when trained with gradual grouping, significantly improve accuracy over the usual training. Our proposed model gives accuracies of 68% while having only 5.77 GFLOPs (5X reduction in FLOPs). By implementing this proposed training procedure in ESSNet models, the accuracy degradation reduces, resulting in performance efficient compact networks. ESSNet-DG2 has 6X reduction in model size and 66% mIOU as seen in Table 3.5.

Our first approach is to employ the proposed compact layers in all the residual blocks of the encoder, while the decoder is still not optimized. In the second approach, we design architecture variants that achieve substantial compression compared to various other models with insignificant loss in accuracy by exquisitely applying the proposed CNN blocks. In this chapter, we stick to a constant macro architecture. We closely examine the trade-offs for each design choice and thereby choose our implementation techniques.

Figure 3.10: Performance trade-off graph for all the models studied. Note that the green points representing models trained by gradual grouping give the best performance trade-offs. Also, the selective application of proposed layers (orange points) hardly degrades the accuracy while still giving a reasonable reduction in GFLOP of 1.5X over the baseline ERFNet, which runs at 27.7 GFLOPs.

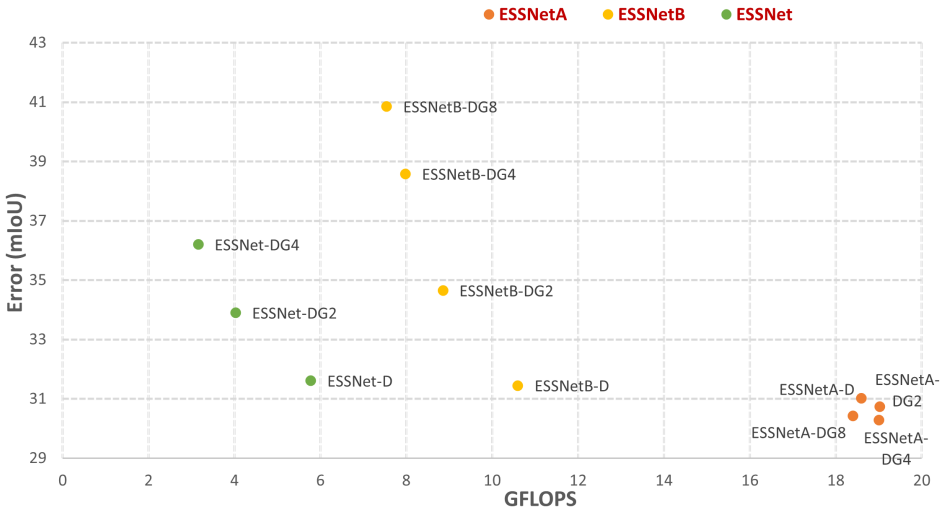

Table 3.5: Gradual Training of Grouped Convolutions. As can be seen, our proposed models have FLOPs ranging from 5.77 GFLOPs to 3.15 GFLOPs, while the best accuracy is around 68%. Improvement in accuracy is seen due to gradual training from the traditional training method (reported in Table 3.2)

Models	IOU	Params	GFLOPs
ERFNet (Baseline)	70.45	2038448	27.705
ESSNet-D	68.39	431312	5.773
ESSNet-DG2	66.10	279760	4.029
ESSNet-DG4	63.80	203984	3.156

3.6 State of the Art Comparison

We quantitatively compare our proposed models with other state-of-the-art methods. Table 3.6, shows the results on the Cityscapes dataset in terms of FLOPs, Parameters, FPS and accuracy (mIoU). Our proposed models give consistent results in terms of all the metrics. Our proposed model ESSNet-DG2 is almost 100x compressed than PSPNet. Although ENet is heavily compressed in terms of model size, the performance degradation is almost 30% compared to PSPNet. Our proposed models achieve comparable compression with ENet while having nearly 10% improvement in terms of accuracy. Our most compact model ESSNet-DG4 achieves 49FPS, and 63.8% mIoU, which is superior to ENet in both the metrics. Figures 3.13, 3.14 show class-wise performance in comparison to the state-of-art models (Per-Class IoU(%) for each of the 19 classes).

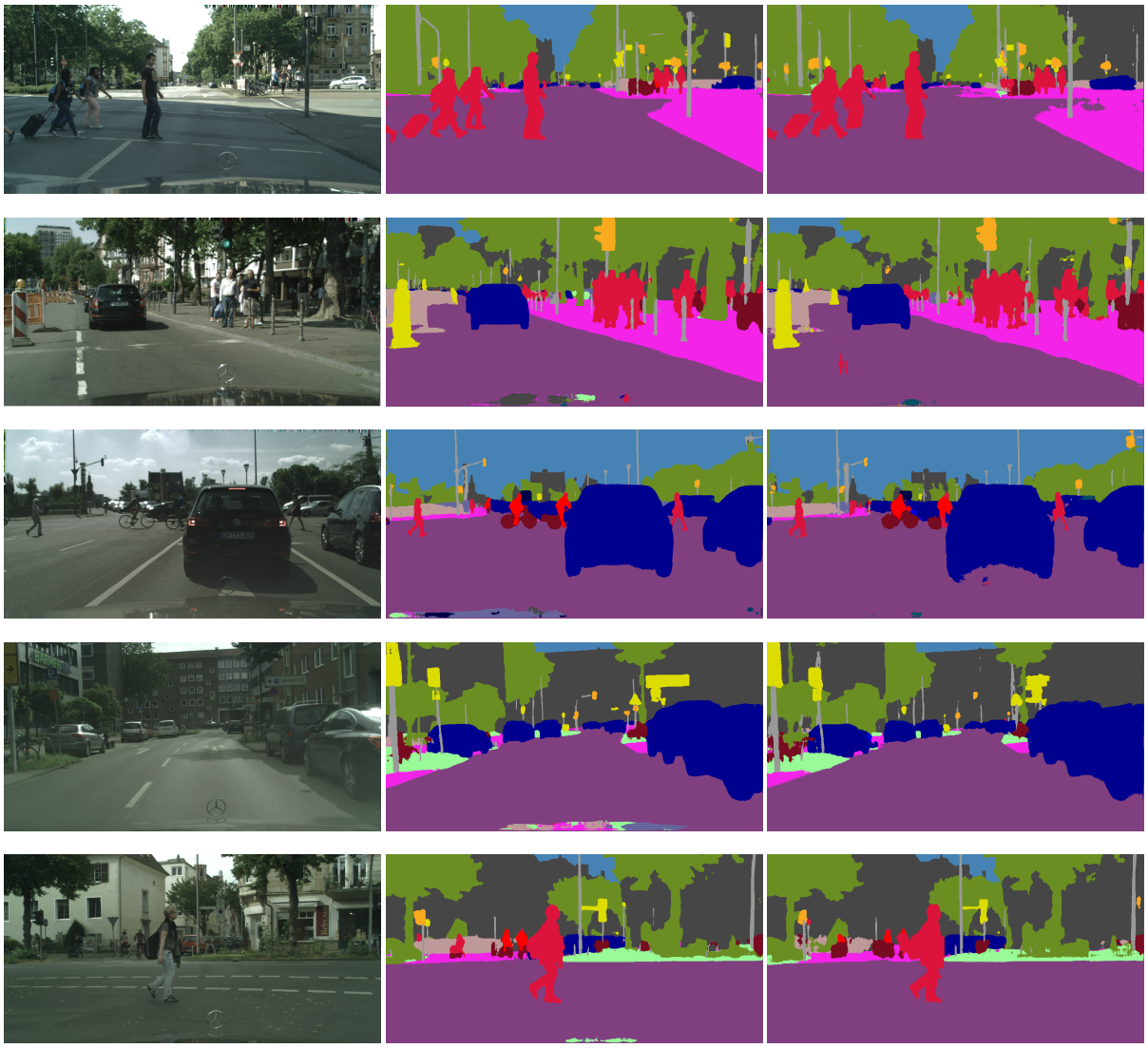

Figure 3.11: Examples of qualitative results demonstrating input images from the Cityscapes dataset, ERF Net output and our proposed ESSNet-DG2 model with gradual grouping segmentation output. We observe that our proposed model with **7x** lesser parameters gives a segmentation output that yields consistent results for all the classes, and the segmentation is qualitatively good. Both the networks accurately predict the road and objects in the scene. However, we observe that the reduction in accuracy reflects in the prediction of small size objects such as "bicycles","fence", and "riders", which are at far distances in the scene. We also observe that the segmentation boundaries are slightly not well defined when objects of the same class like "car" are at a closer distance than the ground truth. All of our proposed network variants have significantly lesser parameters and memory footprint compared to other high-performance segmentation models. These proposed models are meant to be used in real-time applications with system-on-chip (soc) devices that can be mounted on vehicles in numerous scene understanding applications.

Our proposed models achieve an optimal balance between model size and performance making them an ideal choice for the task of real-time semantic segmentation. Figure 3.11 shows a qualitative comparison of our proposed ESSNet-DG2 model output with ERFNet output and groundtruth.

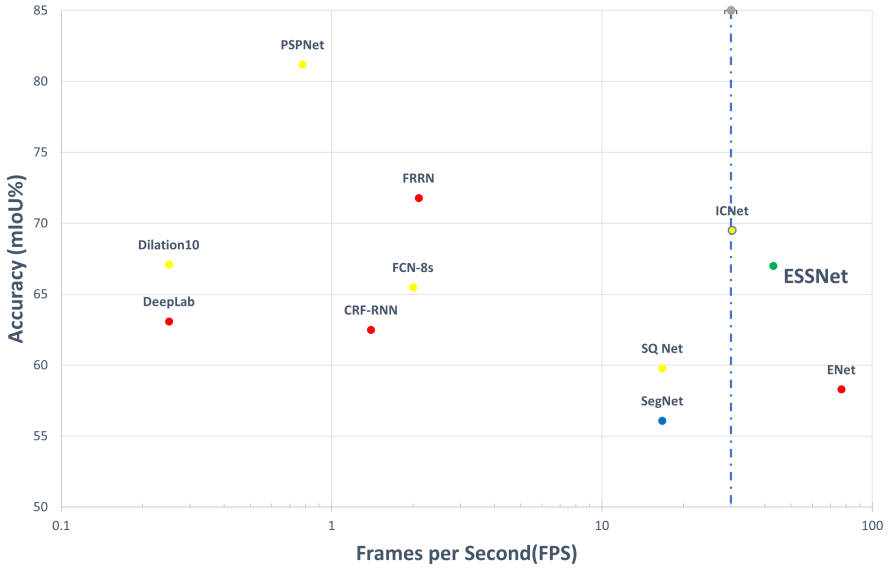

Figure 3.12: The accuracy (mIoU) and inference speed (FPS) trade-off for different state-of-the-art semantic segmentation approaches on Cityscapes test set. Each colour represents the input image size at the time of test as listed in table 3.6. Models that run at FPS $>$ 30 are considered to be real-time models.

3.7 Summary

We approach the problem of designing highly efficient CNNs for semantic segmentation, specifically focusing on autonomous navigation. For this purpose, we study the ERFNet model, which is already among the most efficient, real-time models on the Cityscapes dataset. We comprehensively analyze the most effective design methodologies and determine the components that are the best fit to build our proposed models. We obtain our lightweight networks benefiting from residual blocks, spatial kernel factorization, depthwise separable convolutions, and grouped convolutions. The core residual blocks help eliminate the degradation problem in deep neural networks and also help in speeding up the training time.

Through thorough experimentation, we empirically establish the effectiveness of our proposed models. Depth-wise separable convolutions used in classification models like MobileNet [26], Xception [6] and ResNeXt [80] are proved to be efficient. Our main idea was to translate this efficiency into segmentation models. Though grouped convolutions are a great way to induce structured sparsity, we apprehend that grouped convolutions coupled with depthwise separable convolutions can significantly

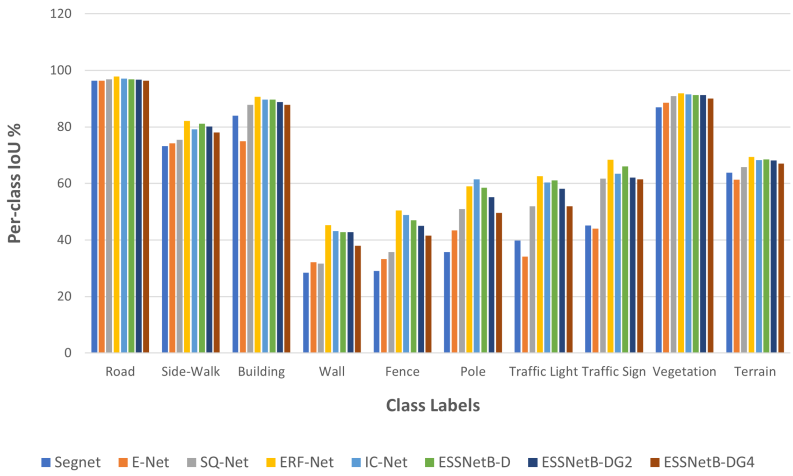

Figure 3.13: Performance of Per-Class IoU of our proposed models on each class in the Cityscapes dataset compared to other state-of-the-art real-time segmentation models

.

compress the network but leads to accuracy degradation. We infer that the decoder is not meant for further feature extraction, but its function is only to recover the feature representations to meaningful probability maps. Apart from initializing the training from scratch, we also experiment with the pretrained approach. In this process, we take advantage of the regularization opportunity offered by knowledge transfer[48] from large recognition datasets such as ImageNet[60].

We propose segmentation frameworks ESSNetA , ESSNetB , and ESSNet that are competitively accurate, lightweight, and fast enough for real-time applications. Our main result is to obtain a semantic segmentation model with 5.8 GFLOPs running time with IOU scores of 68%. We propose a novel training procedure that can be easily implemented, specifically targeting grouped convolutions. The procedure starts with dense convolutions and gradually evolves toward grouped convolutions as the training progresses, allowing the optimization to be done in a higher-dimensional space. We empirically show that this procedure on our proposed efficient architecture results in a model running at 4.1 GFLOPs while giving accuracies equitable to other state-of-the-art networks.

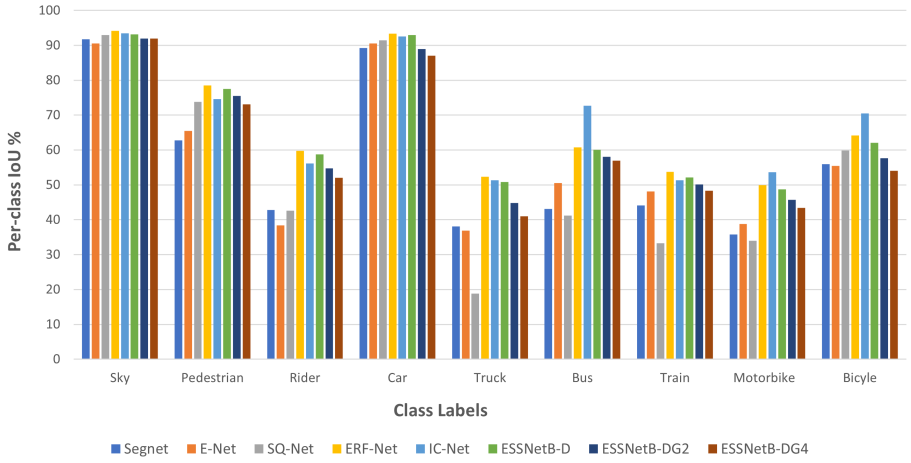

Figure 3.14: Performance of Per-Class IoU of our proposed models on each class in the Cityscapes dataset compared to other state-of-the-art real-time segmentation models.

Table 3.6: Our proposed models benchmark on the Cityscapes dataset compared to various other state-of-the-art approaches.

Models	**InputSize**	**GFLOPs**	**Params(M)**	**Frame(FPS)**	**mIoU(%)**
FCN-8S[42]	512×1024	136.2	134	2	63.1
PSPNet[88]	713×713	412.2	250.8	0.78	81.2
DeepLab[23]	512×1024	457.8	262.1	0.25	63.1
ICNet[87]	1024×2048	28.3	26.5	30.3	69.5
ESSNet -D	512×1024	5.78	0.43	47	68.39
ESSNet -DG2	512×1024	4.03	0.28	49	66.10
ESSNet -DG4	512×1024	3.16	0.21	49	63.80
FRRN[54]	512×1024	235	-	0.25	71.8
SegNet[2]	640×360	286	29.5	16.7	57
ENet[52]	1280×720	3.8	0.4	46.8	58.3
SQNet[72]	1024×2048	270	-	16.7	59.8
CRF-RNN[89]	512×1024	-	-	1.4	62.5

Table 3.7: Comparison between different type of convolutions. Here, $k \times k$ is the kernel size, $k_d = (k-1) \cdot d+1, d$ is the dilation rate, c and $\hat{c}$ are the input and output channels respectively, and g is the number of groups.

Convolution type	Parameters	Eff. receptive field
Standard	$k^2 c\hat{c}$	$k \times k$
Group	$\frac{k^2 c\hat{c}}{g}$	$k \times k$
Depth-wise separable	$k^2 c + c\hat{c}$	$k \times k$
Depth-wise dilated separable	$k^2 c + c\hat{c}$	$k_d \times k_d$
Asymmetric	$2kc\hat{c}$	$k \times k$
Dilated	$k^2 c\hat{c}$	$k_d \times k_d$
Bottleneck	$cc' + k^2 c' + c'\hat{c}$	$k \times k$

Chapter *4*

Design Considerations for Efficient Architectures

In the previous chapter, we analyzed the efficacy of sparse convolutions, and evaluated the difference in the performance of various cost-cutting design methods of convolutional layers. In this chapter, we propose architectures that are sufficiently accurate compared to the other state-of-the-art approaches while using notably less number of network parameters and producing substantially faster segmentation outputs. In the previously proposed frameworks, our overall design strategy was lightweight convolutions in deep convolutional neural networks (DCNNs). We aimed to enlarge the receptive field, which is crucial to obtain accurate semantic information by deepening the network, downsizing feature maps, and using dilated or atrous convolutions.

In this chapter, we perform experiments with varying macro architecture hyper-parameters such as depth of the network, size of decoder and additional context extraction modules. The concept behind the design technique employed and its benefits, drawbacks, and implications on the segmentation output are discussed in this chapter. Furthermore, we propose a novel segmentation network WSPD-Net that incorporates a pyramid structure in the convolutional layer with differential dilation rates that is much more effective without increasing computational complexity. In this network, we exploit different kernel sizes with varying-sized receptive fields, which are proven to better segment diverse sized objects [32],[78]. This proposed framework is also end-to-end trainable but the design methodology is shallow and symmetric as opposed to ESSNet models which employ sparse and deep convolutional layers.

4.1 Variations in the Decoder

In this section, we explore the instrumentality of the decoder in a real-time segmentation network, and discuss our proposed techniques to improve the results. To make the model lighter, we experiment with various versions of the decoder and compare the computational operations vs accuracy to attain a suitable balance between both. As discussed in section 3.5, ESSNet-D architecture has a lightweight decoder employing 1x1 convolutions in the upsampling(trans2x) block along with proposed convolutional layers-D. Optimizing only the upsampling block in ESSNetB-D decoder results in 7.4 GFlops, and the accuracy is 66.51%. We present exploratory work on lightweight decoders. We study decoder variants

with stride factors of 4x and 8x, respectively named ESSNet-trans4x and ESSNet-trans8x. We also experimented by completely removing the decoder (i.e., deconvolution operation) and used bi-linear interpolation by a factor of 8, which is a straightforward technique for restoring the original image resolution. Transposed convolutions accomplish upsampling by adding blank spaces between successive pixels in the original feature maps, much like dilated convolutions, and executing conventional convolution operations on the upsampled feature maps. ESSNet-trans4x has two deconvolution operations, one with a upsampling factor of 4 followed by the two proposed convolutional layers and the final deconvolution layer.

As seen in Table 4.1, ESSNet-trans4x has 5.3GFLops and gives an accuracy of 62.12%, ESSNet-trans8x has only one upsampling operation with an upsampling rate of 8 and gives an accuracy of 60.39% with 4.9 GFlops. ESSNet-nodec model completely removes the decoder and bilinearly interpolates the output of the encoder by a factor of 8 having 4.7GFlops. Although utilising transposed convolution with a high upsampling rate, such as 4x or 8x results in a decoder that is much lighter than the original decoder and the accuracy drops substantially lower than the baseline model. Therefore, the poor performance of transposed convolution with a high upsampling rate is attributable to the excessive insertion of blank spaces, degrading the high-level feature representations produced during the encoder phase. ESSNet-no decoder model has only 1/6 GFlops and 1/2 the number of parameters compared to the baseline model. For performance reasons, we always have the last convolutional layer as a full convolution without any modifications.

Table 4.1: We evaluate the variations in decoder as against the upsampling rate with the number of GFlops

Models	**IOU**	**GFLOPs**
ESSNet-enc-UPS(proposed)	66.51	7.41
ESSNet-trans4x	62.12	5.31
ESSNet-trans8x	60.39	4.89
ESSNet-nodec	57.33	4.70

4.2 Pyramid Pooling Module

We experiment with improvising on the architecture by adding a pyramid pooling module. This module is devised to incorporate the global contextual prior[88]. However, PSPNet has a heavy backbone, a ResNet feature extractor that uses huge computational resources. In this network, we utilize our proposed lightweight encoder as the backbone and add the Pyramid Pooling module on top of the encoder's final layer feature map. We utilize four pyramid levels which are 1/8x, 1/4x, 1/2x, and 1x size scale blocks to the size of the final convolutional block of the encoder. The number of channels of these pyramid blocks are 128 which is same as the last encoder block. The number of channels of the pyramid blocks is reduced to 1/4 (i.e., 32) by a 1x1 convolution operation. Followed by the low-dimesional projection, these feature maps are bi-linearly upsampled to the same size as the original feature map taken from the encoder. Thereafter, all these five maps are concatenated along the channel dimension (256

channels), resulting in a robust feature map containing local and global information, as seen in fig 4.1. This is further attached to the upsampling block with a stride factor of 4, followed by convolutional layers and the final class probability layer. We skip one upsampling block and the following convolutional blocks (i.e., layers 17-19) from the ESSNetB architecture as seen in table 3.2 to reduce the size of the decoder.

The encoder is trained from the down-sampled annotations as ground truth for the first 95-100 epochs, and in the second step, we train the complete architecture end-to-end, including the pyramid module to get the segmentation output. The training protocol is the same as the earlier architectures using Adam optimizer and weighted cross-entropy loss function. We do additional data augmentation with random cropping for better invariance of the model to aspect ratio and scale changes.

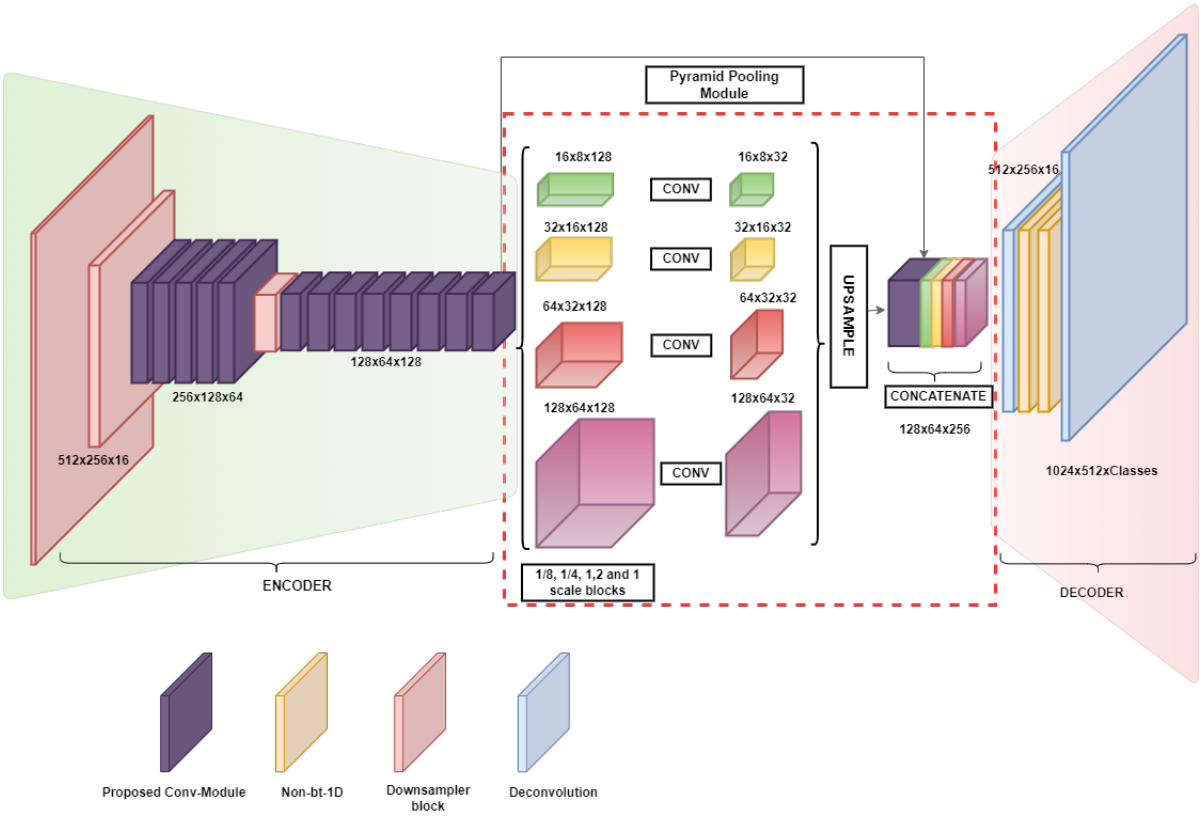

Figure 4.1: This figure shows the addition of pyramid pooling block before the decoder

All the layers in the encoder are our proposed convolutional modules with 3x3 depthwise separable convolutions followed by 1x1 grouped convolutions. However, the additional 1x1 convolutional projections in the pyramid pooling module outweigh the lightweight decoder. Thus, we do not see a significant computational gain in the network. For the architecture, as seen in figure 4.1, the proposed convolutional modules with group size 2, 4 and 8 give accuracies 65%, 62.7% and 59.4% respectively. Although the mean IoU is not as expected, we see improvement in the per-class IoU for classes like

Fence, Traffic Light, Truck, and Bicycle. The spatial details contained in low-level features are still ignored in these models and leading to incorrect identifications of small objects and boundary information. The multi-level context extraction module is more effective in the networks with deeper layers as the high-resolution features are captured better. Additional pretraining on the ImageNet dataset has been shown to improve accuracy by 2% in each case. Though the resultant advantages do not directly translate as much as for the architectures designed for competitive ImageNet performance, the models based on ImageNet pretrained encoders are observed to be benefited from the regularization induced by transfer learning. We also overcome the risk of over-fitting with this method. Whenever we use ImageNet pretraining, the pretrained parameters are updated by a learning rate of $1.25\mathrm{e}^{-4}$ and a weight decay rate of $0.5\mathrm{e}^{-4}$, which is 4x smaller than the previous learning rate and weight decay parameters. Due to additional computations, the forward pass time is higher than the previous ESSNetmodels. We observe that an additional hierarchical global prior did not show much improvement in the mIoU results using our light-weight architecture on the Cityscapes dataset. We further experiment by providing additional 20K coarse data annotations from the Cityscapes dataset in the training phase. Using this data augmentation with coarsely annotated data and using cropped images for training further improves the model accuracy by 1.2%. However, the pyramid pooling block is proved to improve accuracy on datasets with distortions, varying brightness, and different focal angels like fish-eyed images, as shown in [61],[79],[39].

4.3 Efficient Dilated Convolution Module

Unlike the previous Encoder-Decoder architectures we have seen, in this section, we explore a different direction of designing the network. SegNet and UNet have shown symmetric architectures for semantic segmentation. However, in the previous chapter, we introduced architectures with a larger encoder and a light-weight decoder. We also used the same kernel size throughout. In this design, we explore ways to reduce the computational complexity by widening the filter kernels in some layers, and at the same time, reducing the number of convolutional layers. The idea behind WSPD-Net is to better encode the spatial information by learning representations from a larger effective receptive field with minimal additional computation cost. In DWS-Dil block, the idea is to leverage the representation capacity of large and dense layers at a lower cost.

In addition to the Non-Bt-1D layers as seen in the previous chapter, that are factorized spatially as 3x1 and 1x3, we also use 5x1 and 1x5 convolutions that are larger kernels and help extract broader scale information like in ENet[52]. We propose to stack the smaller kernels in the early layers and larger kernels in the encoder's deeper layers to capture low scale feature maps from the initial layers. Likewise, our proposed convolutional layer in the previous chapter has 3x3 depthwise convolutions to make the connections sparse, and the layers are subsequently stacked with different dilation rates of 2,4,8,16 in each layer. A filter of size K x K having dilation rate d, with an expansion of $(d-1)*(d-1)$ will cover $(d-1)*K*(d-1)*K$ pixels. Dilated convolutions are of great use for semantic segmentation networks

because the receptive field increases exponentially, implying a better grasp of contextual details without extra calculations[78].

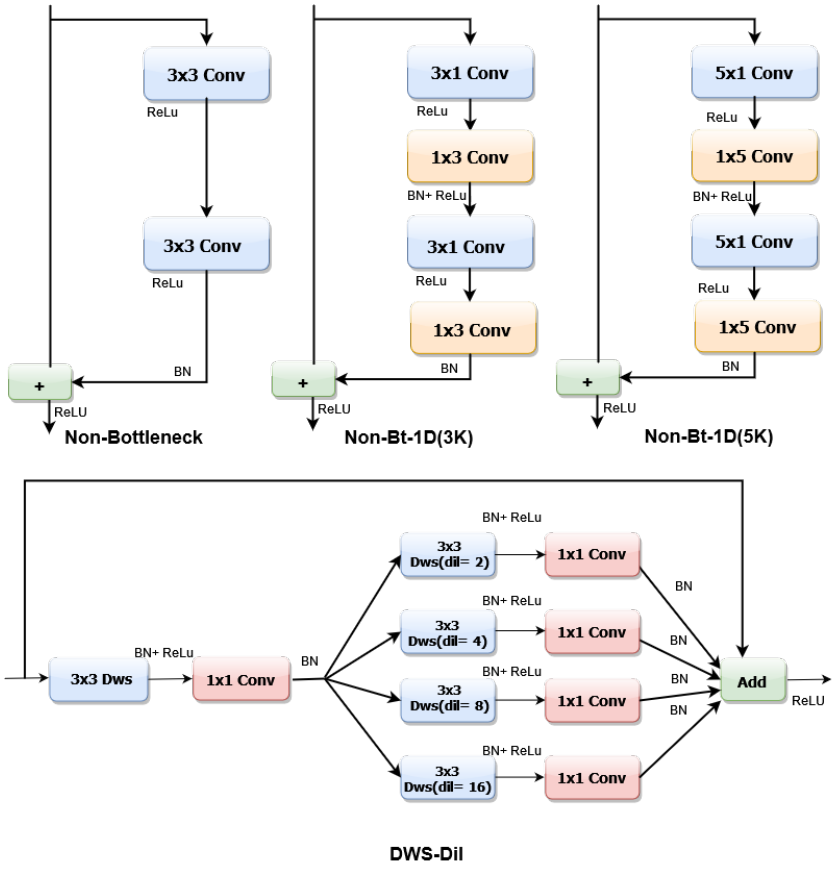

Figure 4.2: Different convolutional layers used in our proposed WSPD-Net architecture. Non-Bt-1D(3K), Non-Bt-1D(5K) are spatially factorized residual layers with kernel sizes 3,5 respectively. DWS-Dil is our proposed layer with four different dilation rates used parallely in the same residual layer. This proposed layer not only reduces the complexity of the computations but also makes it possible for the network to learn representations from a greater effective receptive field.

However, we observe that using dilation with depthwise factorization as a direct combination leads to extreme sparsity in the feature maps implying significantly less connectivity to the neighboring pixels, which causes loss of information. From this inference, we propose to design a novel block (DWS-Dil) that effectively integrates dilated convolutions in the efficient convolutional layer in a pyramid fashion. The cross-channel information loss is unavoidable given that the dilated convolutional operation is spatially discontinuous according to the dilation rate. Hence, we place the depthwise separable convolutions after the first set of asymmetric convolutions to reduce the effect of excessive sparsity and poor information flow in the encoder.

Considering the design choices we made in this network, i.e., using wider spatially factorized kernels and dense dilated blocks, we reduce the number of layers (i.e., shallower architecture) in comparison to the previous networks. Similar to the previous architectures i.e., ESSNetA , ESSNetB , we use three

Table 4.2: Proposed WSPD-Net(symmetric and parallely dilated network) architecture

	Layer	Type	out-chann	out-Res
ENCODER	1	**Downsampler block**	16	512x256
	2-4	**3 x Non-bt-1D(3K)**	16	512X256
	5	**Downsampler block**	64	256x128
	6-7	**2 x Non-bt-1D(5K)**	64	256x128
	8	**Downsampler block**	128	128x64
	9-11	**2 x DWS-Dil**	128	128x64
DECODER	12	**Deconvolution**(upsampling)	64	256x128
	13-14	**2 x Non-bt-1D(5k)**	64	256x128
	15	**Deconvolution**(upsampling)	16	512x256
	16-17	**2 x Non-bt-1D(3K)**	16	512x256
	18	**Deconvolution**(upsampling)	C	1024x512

downsampling blocks with the exact spatial sizes of feature maps at each downsampling stage and three upsampling blocks. We lowered the number of encoder layers by five in the encoder's later stages while retaining all of the decoder's layers from the baseline architecture.

Rather than using two downsampling blocks subsequently, we apply convolutional layers after the first downsampling block allowing superior feature extraction. Hence, this is more of a symmetric structure like SegNet with fewer parameters. The experimental setup is the same as seen in ESSNetA and ESSNetB architectures, and the network is trained end-end on the Cityscapes dataset. The detailed network configuration is listed below. This architecture is compact and has 18 layers, whereas the baseline architecture has 23 layers.

The design choices in this architecture are different from the previous trend of choices in terms of size, number of layers, and direction of expansion. In the previous chapter, we have seen variations in the convolutional layers keeping the macro architecture the same. We experiment with design parameters such as the number of layers, kernel dimensions, size of encoder-decoder, dilation rates, and combination of factorized filter kernels.

The significant contribution to this design is our novel DWS-Dil block reduces the direct loss of sequential processing of dilated feature maps but rather parallelly does the same in a concentrated manner. This architecture performs better in terms of processing time and has fewer parameters than our baseline architecture ERFNet. This model has 0.67 million parameters, whereas the baseline model has 0.72 million parameters. This model runs at 51FPS on Nvidia GTX 1080Ti GPU, which is faster than the previous models and gives an accuracy of 69.72%. Figure 4.3 shows the segmentation output of this network. We observe that the classes that are under represented have lower per-class mIou than classes that are well represented. Pretraining the network with coarse annotation helps overcome this problem by giving a slight enhancement in accuracy on classes having a lesser representation than others in the dataset [17]. This architecture produces near state-of-the-art performance with better inference time and fewer parameters.

Figure 4.3: Segmentation Output of WSPD-Net Architecture

4.4 Summary

We explore various design guidelines from previous literature to tackle the problem of designing efficient segmentation architectures that retain the capacity of deep networks without its limitations. We also validated the effect of various transposed convolutions with 2x, 4x, and 8x upsampling rates in the upsampling block of the decoder. We also experimented by removing the decoder, resulting in the smallest model with 4.7GFlops, but we observed a significant drop in accuracy. We proposed an architecture design incorporating the pyramid pooling module to integrate the fine-grained local information with global contextual information.

Using the results, we propose a novel segmentation network WSPD-Net. In this network, we employ a heterogeneous combination of spatially factorized and depthwise separable CNN layers. Additionally, we employ an aggregated structure of dilated convolutions in a subset of the encoder's blocks to boost contextual extraction capability without negatively impacting the encoder's efficiency. Our designed DWS-Dil layer generalizes the use of factorized dilated convolutions in an efficient manner. Our proposed WSPD-Net achieves the goal of a real-time segmentation model attaining near state-of-the-art accuracy while maintaining a low level of latency.

Chapter *5*

Conclusions and Future Work

This chapter presents the summary of the thesis and the conclusions drawn from our work. In addition, a brief discussion of the future work is presented.

5.1 Summary

This thesis addresses the problem of real-time semantic segmentation and presents solutions that enhance the model efficiency while maintaining the state-of-the-art accuracy level. We present simple yet efficient architectures for the task of image semantic segmentation. Our proposed networks are designed to achieve high accuracy and processing speeds for practical applications in resource-constrained environments.

In Chapter 2, we introduced the semantic segmentation task and presented a detailed discussion of various deep learning approaches to this problem. A study of existing literature for relevant model compression techniques was presented. We discuss in detail the significance of real-time semantic segmentation in light of the increasing popularity of autonomous vehicles. We detailed how various state-of-the-art methods have approached this problem to overcome the computational burden in traditional semantic segmentation networks. We have discussed the merits and demerits of existing model compression approaches.

In Chapter 3, we discussed how previous literature for image classification tasks had approached the individual components in deep convolutional neural networks. A fruitful discussion is presented on the encoder and decoder components in the proposed architecture pipeline providing vital insights into network design. Various strategies were introduced, such as spatial factorization, depth-wise separable convolutions, grouped convolutions, and channel shuffling. We identified various computational bottlenecks in the existing architectures and adapted the most suitable techniques for boosting the efficiency of the models while preserving accuracy. A detailed calculation of the computational burden for each design choice was provided. The performance of the proposed models was assessed by weigh-

ing the trade-off between the accuracy metric and the efficiency metric. A novel training framework using model compression technique for grouped convolutions, called gradual grouping, was presented. The procedure starts with dense convolutions and gradually evolves toward grouped convolutions as the training progresses, allowing the optimization to be done in higher-dimensional space. We successfully obtained an optimal trade-off between performance and efficiency for real-time segmentation architectures. A quantitative comparison of our proposed models with various state-of-the-art methods is presented. To the best of our knowledge, this type of compression approach was never used in the previous works for semantic segmentation, and this is the first such effort. We thoroughly examine the aspects of each design choice and discuss their impacts on qualitative and quantitative performance. We also validate the benefit of pretraining the encoder on Imagenet and using a custom decoder.

Chapter 4 empirically analyzes the fundamental network design strategies in the existing literature. We derive valuable insights into the efficacy of the decoder, and we incorporate a pyramid structure in the added design components to improve the quality of segmentation. Each section details several experiments and results of the proposed meta-architecture changes to the networks. Our proposed architecture WSPD-Net details the benefits and merits of parallelly using different dilation rates in the same convolutional layer. Our proposed module DWS-Dil, not only reduces computational complexity by kernel factorization but also improves the effective receptive field. We carefully design our encoder and decoder blocks in a way that maximizes the trade-off between low-resolution features, which are more effective and include more context, and the high-resolution features, which have better feature localization at the pixel level but are more expensive to compute.

5.2 Conclusion

Real-time semantic segmentation is a crucial part of the early pipeline of several critical computer vision applications, such as autonomous vehicles, ADAS systems, or robot-assisted surgery. In these applications, time is of the essence, and any delays may result in disastrous consequences. Thus, improvements in real-time semantic segmentation affects downstream benefits for many computer vision tasks. In this work, we propose methods and architectures that give *high performance* in *real-time* and working on *resource-constrained settings.*

This thesis comprehensively analyzes the various key components in designing real-time semantic segmentation architectures. We overcome the limitations of increased network complexity and redundant computational overhead by designing efficient convolutional layers and strategically placing these layers in the network. We choose the suitable design elements that result in a beneficial compromise between latency and accuracy. With the help of these design choices, we propose end-to-end trainable lightweight architectures. Secondly, we explore the need for optimization during the training phase in the proposed models. We devise a novel training method derived from the model compression tech-

niques, resulting in a highly compact deep learning model, achieving improved results. A substantial amount of work has been done in comparing our proposed models with the existing real-time semantic segmentation architectures. We also exhaustively explore the essential components unique to semantic segmentation tasks that affect the performance and segmentation quality and take them into account in our proposed architecture designs.

5.3 Future Directions

Although our proposed models achieve an excellent accuracy/efficiency balance with limited computational budgets, there are still some areas for improvement. In this section, we discuss potential directions in which the proposed models could be extended for further improvement.

Our proposed networks attain significant compression with minor compromise in performance. Attaining best accuracy in constrained budgets is still an open research problem to pursue. A significant percentage of information is lost during the encoding process, which results in the low performance of the semantic segmentation network. A possible direction of research to solve this problem is to reintroduce the input information in the network by adding a low-resolution version of the input to intermediate areas or by using a parallel sub-network.

Our encoder network relies primarily on the dilated convolutions, which cause gridding effects in the output. Further research is required to design architectures without this drawback. In this work, we propose a convolutional layer that incorporates distinct dilation rates in a spatial direction. An interesting way would be to integrate distinct dilation rates in the channel direction at a low cost. Another possible focus is to streamline the decoder block, as the compressed upsampling operation does not show satisfactory performance in recovering the input information. Further research should be undertaken to explore how the decoder can be efficiently designed such that performance is not impacted.

Related Publications

- **Efficient Semantic Segmentation using Gradual Grouping**

Nikitha Vallurupalli [1], Sriharsha Annamaneni [1], Girish Varma [1], C V Jawahar [1], Manu Mathew [2], Soyeb Nagori [2]

IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) 2018 [**Best Runner Up Award**]

[1] Center for Visual Information Technology, Kohli Center on Intelligent Systems, IIIT-Hyderabad, India

[2] Texas Instruments, Bangalore, India

Bibliography

[1] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *NIPS*. 2016.

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

[3] A. Briot, P. Viswanath, and S. K. Yogamani. Analysis of efficient cnn design techniques for semantic segmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 776–77609, 2018.

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. 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, 2018.

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. P. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2015.

[6] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CVPR*, 2017.

[7] N. Cohen, O. Sharir, and A. Shashua. Deep simnets. In *CVPR*, pages 4782–4791, 2016.

[8] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[9] H. Ding, X. Jiang, B. Shuai, A. Q. Liu, and G. Wang. Context contrasted feature and gated multi-scale aggregation for scene segmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2393–2402, 2018.

[10] G. Dong, Y. Yan, C. Shen, and H. Wang. Real-time high-performance semantic image segmentation of urban street scenes. *IEEE Transactions on Intelligent Transportation Systems*, 22:3258–3274, 2021.

[11] A. Ess, T. Mueller, H. Grabner, and L. V. Gool. Segmentation-based urban traffic scene understanding. In *BMVC*, 2009.

[12] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. *ArXiv*, abs/1911.11134, 2020.

[13] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:1915–1929, 2013.

[14] F. B. Flohr and D. M. Gavrila. Pedcut: an iterative framework for pedestrian segmentation combining shape models and multiple data cues. In *BMVC*, 2013.

[15] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv: Learning*, 2019.

[16] B. Fulkerson, A. Vedaldi, and S. Soatto. Class segmentation and object localization with superpixel neighborhoods. In *2009 IEEE 12th International Conference on Computer Vision*, pages 670–677, 2009.

[17] M. E. Gamal, M. Siam, and M. Abdel-Razek. Shuffleseg: Real-time semantic segmentation network. *ArXiv*, abs/1803.03816, 2018.

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[19] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev. Compressing deep convolutional networks using vector quantization. *ArXiv*, abs/1412.6115, 2014.

[20] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[21] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *ArXiv*, abs/1506.02626, 2015.

[22] B. Hariharan, P. Arbeláez, L. D. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. *2011 International Conference on Computer Vision*, pages 991–998, 2011.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[24] Y. He, J. Qian, and J. Wang. Depth-wise decomposition for accelerating separable convolutions in efficient convolutional neural networks, 2019.

[25] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, abs/1207.0580, 2012.

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[27] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *arXiv preprint arXiv:1711.09224*, 2017.

[28] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡1mb model size. *ArXiv*, abs/1602.07360, 2016.

[29] Y. A. Ioannou, D. P. Robertson, J. Shotton, R. Cipolla, and A. Criminisi. Training cnns with low-rank filters for efficient image classification. *CoRR*, abs/1511.06744, 2016.

[30] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[31] S. Jégou, M. Drozdzal, D. Vázquez, A. Romero, and Y. Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1175–1183, 2017.

[32] J. Kim and Y. S. Heo. Efficient semantic segmentation using spatio-channel dilated convolutions. *IEEE Access*, 7:154239–154252, 2019.

[33] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[35] N. Lee, T. Ajanthan, and P. Torr. Snip: Single-shot network pruning based on connection sensitivity. *ArXiv*, abs/1810.02340, 2019.

[36] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2017.

[37] H. Li, P. Xiong, H. Fan, and J. Sun. Dfanet: Deep feature aggregation for real-time semantic segmentation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9514–9523, 2019.

[38] G. Lin, C. Shen, A. van dan Hengel, and I. D. Reid. Efficient piecewise training of deep structured models for semantic segmentation. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3194–3203, 2016.

[39] J. Lin, W. Jing, H. Song, and G. Chen. Esfnet: Efficient network for building extraction from high-resolution aerial images. *IEEE Access*, 7:54285–54294, 2019.

[40] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2014.

[41] W. Liu, A. Rabinovich, and A. C. Berg. Parsenet: Looking wider to see better. *ArXiv*, abs/1506.04579, 2015.

[42] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[43] F. Mamalet and C. Garcia. Simplifying convnets for fast learning. In *ICANN*, 2012.

[44] M. Mathew, K. Desappan, P. K. Swami, and S. Nagori. Sparse, quantized, full frame cnn for low power embedded devices. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 328–336, July 2017.

[45] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[46] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, 2015.

[47] M. Oberweger, P. Wohlhart, and V. Lepetit. Hands deep in deep learning for hand pose estimation. *ArXiv*, abs/1502.06807, 2015.

[48] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1717–1724, 2014.

[49] M. Orsic, I. Kreso, P. Bevandic, and S. Segvic. In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12599–12608, 2019.

[50] M. Orsic and S. Segvic. Efficient semantic segmentation with pyramidal fusion. *Pattern Recognit.*, 110:107611, 2021.

[51] W. Pan, H. Dong, and Y. Guo. Dropneuron: Simplifying the structure of deep neural networks. *ArXiv*, abs/1606.07326, 2016.

[52] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[53] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[54] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe. Full-resolution residual networks for semantic segmentation in street scenes. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3309–3318, 2017.

[55] Z. Qin, F. Yu, C. Liu, and L. Zhao. Functionality-oriented convolutional filter pruning. In *BMVC*, 2019.

[56] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[57] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua. Learning separable filters. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2754–2761, June 2013.

[58] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2018.

[59] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[60] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2015.

[61] Á. Sáez, L. M. Bergasa, M. E. L. Guillén, E. Romera, M. Tradacete, C. G. Huélamo, and J. del Egido. Real-time semantic segmentation for fisheye urban driving images based on erfnet †. *Sensors (Basel, Switzerland)*, 19, 2019.

[62] A. Sagar and R. Soundrapandiyan. Semantic segmentation with multi scale spatial attention for self driving cars. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 2650–2656, 2021.

[63] E. Shelhamer, K. Rakelly, J. Hoffman, and T. Darrell. Clockwork convnets for video semantic segmentation. In *European Conference on Computer Vision*, pages 852–868. Springer, 2016.

[64] N. Simon, J. H. Friedman, T. Hastie, and R. Tibshirani. A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22:231 – 245, 2013.

[65] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[66] P. Singh, V. Verma, P. Rai, and V. P. Namboodiri. Play and prune: Adaptive filter pruning for deep model compression. In *IJCAI*, 2019.

[67] A. Sironi, B. Tekin, R. Rigamonti, V. Lepetit, and P. V. Fua. Learning separable filters. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2754–2761, 2013.

[68] C. H. Sudre, W. Li, T. K. M. Vercauteren, S. Ourselin, and M. J. Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. *Deep learning in medical image analysis and multimodal learning for clinical decision support : Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, held in conjunction with MICCAI 2017 Quebec City, QC,...*, 2017:240–248, 2017.

[69] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, June 2015.

[70] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[71] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[72] M. Treml, J. A. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, B. Nessler, and S. Hochreiter. Speeding up semantic segmentation for autonomous driving. 2016.

[73] Y.-H. Tseng and S.-S. Jan. Combination of computer vision detection and segmentation for autonomous driving. In *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 1047–1052, 2018.

[74] C. Wang, G. Zhang, and R. B. Grosse. Picking winning tickets before training by preserving gradient flow. *ArXiv*, abs/2002.07376, 2020.

[75] G. Wang, W. Li, S. Ourselin, and T. K. M. Vercauteren. Automatic brain tumor segmentation using convolutional neural networks with test-time augmentation. *ArXiv*, abs/1810.07884, 2018.

[76] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu. Pruning from scratch. *ArXiv*, abs/1909.12579, 2020.

[77] Y. Wang, Q. Zhou, J. Liu, J. Xiong, G. Gao, X. Wu, and L. J. Latecki. Lednet: A lightweight encoder-decoder network for real-time semantic segmentation. *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1860–1864, 2019.

[78] Y. Wang, Q. Zhou, and X. Wu. Esnet: An efficient symmetric network for real-time semantic segmentation. In *PRCV*, 2019.

[79] W. Xiang, H. Mao, and V. Athitsos. Thundernet: A turbo unified network for real-time semantic segmentation. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1789–1796, 2019.

[80] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[81] Z. Yang, H. Yu, Q. Fu, W. Sun, W. Jia, M. Sun, and Z.-H. Mao. Ndnet: Narrow while deep network for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 22(9):5508–5519, 2021.

[82] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *ECCV*, 2018.

[83] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of The Royal Statistical Society Series B-statistical Methodology*, 68:49–67, 2006.

[84] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[85] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:1943–1955, 2016.

[86] Z. Zhang and M. R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *NeurIPS*, 2018.

[87] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia. Icnet for real-time semantic segmentation on high-resolution images. *arXiv preprint arXiv:1704.08545*, 2017.

[88] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.

[89] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional random fields as recurrent neural networks. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1529–1537, 2015.

[90] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J.-H. Zhu. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, 2018.

[91] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[92] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.