Towards Enhancing Semantic Segmentation in Resource Constrained Settings

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

Master of Science
in
Computer Science and Engineering by Research

by

Ashutosh Mishra
20172087
ashutosh.mishra@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
January 2024

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled '**Towards Enhancing Semantic Segmentation in Resource Constrained Settings**' by **Ashutosh Mishra**, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Dr. Girish Varma
Date	Adviser: Prof. CV Jawahar

To The One

Acknowledgments

I am grateful to many people for providing me with the motivation, support, and confidence in me during the span of this specialization.

First and foremost, I would like to thank Prof CV Jawahar and Dr. Girish Varma who have guided me through the course of this Master's degree by providing consistent support and motivation. They have enabled me to come better both as a student and as a researcher. I would also like to thank Tarun Kalluri (UCSD), Sudhir Reddy (State Univ. of NY), Dr. Anbumani Subramanian (Intel), and Prof Manmohan Chandrakar (UCSD) for the collaboration and for providing their input as and when needed.

I am very grateful to have a friend and brother like Shyam Nandan Rai (pursuing his PhD at University of Torino, Italy and my batch mate from MS and BTech) who has helped me in every aspect of academia. His timely suggestions have also helped me a lot during this journey. I am also thankful to CVIT students and admin staff for making this journey enjoyable. The list is very long but to name a few, Vamsidhar, Siddhant, Abhishek, Avijit Da, Ajoy Da, Ankit Lat sir, Deepayan, Silar sir and so on.

A special thanks to Dr. Anand Mishra (IITJ) who has been a great mentor to me during the course of academia and helped me start my research path.

I am thankful and feel fortunate enough to have friends like Kishan, Anurag, Chris, Chandrakanth, Kabir, and Santhoshini for being there whenever I needed them. I couldn't have asked for anything else.

But most importantly, I would like to thank my family and the family of Shanti Sarovar for their never-ending support throughout. Their love and affection is the reason I have been able to pursue research with such passion and zeal.

Abstract

Understanding the semantics of the scene to automate the decision process for self-driving cars completely is becoming a crucial task to solve in computer vision. Due to the recent progress in the state of autonomous driving, added with a lot of semantic segmentation datasets for road scene understanding being proposed, semantic segmentation of road scenes has recently evolved to be an important problem to tackle. But training semantic segmentation models becomes a resource-intensive task since it requires multi-GPU training and therefore becomes the bottleneck to reproducing results for better understanding quickly. This thesis introduces challenges and provides solutions to reduce the training time of segmentation models by introducing two small-scale datasets. Additionally, the thesis explores the potential of employing neural architecture search and automatic pruning techniques to create efficient segmentation modules in resource-constrained settings. Chapter2 of the thesis introduces the problem of semantic segmentation and discusses some deep learning approaches to solve supervised semantic segmentation. We briefly discuss the different metrics used and also touch upon the statistics of various datasets that are available in the literature to train semantic segmentation models.

Chapter 3 of the thesis explains the need of having a dataset based on the Indian road scenario. Most of the datasets in the literature are captured in Western settings having well-defined traffic participants, delineated boundaries, etc, which seldom mold in the Indian setting. We describe the annotation pipeline, along with the quality check framework used to annotate the dataset. Now, though the IDD dataset [121] caters to the Indian setting, this dataset is still quite resource intensive in terms of GPU computation. Hence, there is a need to have a small resolution, less label-sized dataset for rapid prototyping. We introduce our proposed datasets and provide a detailed set of experiments, and statistical comparisons with the existing datasets to substantiate our claim regarding the usefulness of the proposed solution. We also show through experiments that the models trained using our datasets can be deployed on low-resource hardware such as Raspberry Pi. At the end of this chapter, we also look into the significance of the proposed datasets in facilitating challenges at two prominent conferences: the *International Conference on Computer Vision* (ICCV) and the *National Conference on Pattern Recognition, Image Processing, and Graphics* (NCVPRIPG) in 2019. These challenges aimed to address semantic segmentation in resource-constrained settings, inviting innovative architectures capable of achieving decent accuracy on these proposed datasets. We also discuss the potential application of these datasets in teaching semantic segmentation through a course of notebooks introducing traditional as well as deep learning-based methods to perform segmentation. These notebooks are plug and play, where the first three notebooks can run on laptop CPU, while the fourth notebook requires GPU access.

Chapter 4 of the thesis is about the application of neural architecture search and hyper-parameter tuning in the context of semantic segmentation. *Neural Architecture Search* (NAS) is defined as automating the process of finding novel architecture without a human in the loop. Initially, we discuss the different approaches to performing neural architecture search followed by the description of hyper-parameter tuning. We discuss some cases of architecture search specific to segmentation and classification. With the help of experiments, we then show that the accuracy of different models obtained through training on our proposed datasets correlates well with the accuracy on large-scale datasets, especially in cross-domain settings. This correlation allows for achieving faster results and also paves the way to perform an architectural search for semantic segmentation algorithms.

In the Chapter 5 of this thesis, we delve into the topic of automatic pruning and its application in optimizing segmentation models. Automatic pruning is a technique that aims to improve the efficiency of deep learning models by removing unnecessary connections or parameters without significantly compromising their performance using predefined optimization algorithms. This process is driven by optimization algorithms that analyze the model's structure and weights to determine which connections can be pruned to achieve a more efficient network without any human intervention.

Finally, we conclude the thesis with a summary of the work and also propose future directions.

Contents

List of Figures

List of Tables

Chapter *1*

Introduction

Since the onset of deep learning in 2012, the boundaries of various machine learning tasks have been pushed, and their performances have seen a huge jump. Along with the algorithmic advancements, the rise of high-end computing devices like GPUs and TPUs, added to the ease of availability of datasets, has helped revolutionize the development of computer vision systems.

The first success in deep learning can be attributed to the task of image recognition, where the model was able to perform classifications at a lower error rate when compared to all the previous techniques. Since then, researchers have tried to incorporate the knowledge behind the success of image-based recognition systems into other machine-learning tasks such as machine translation, object detection, etc. A similar trend has been seen for the task of semantic segmentation, in which the idea is to assign a class value to each pixel in the image. The performance of this task with the help of deep learning in the last few years has achieved quite a lot of improvement. Apart from this, there has been a surge in the datasets for training semantic segmentation models consisting of high-resolution images along with other modality data such as GPS information, LIDAR data, etc. The only disadvantage of models trained using these datasets is that the size and scale of such datasets prohibit widespread adoption because of which many university students still don't get the opportunity to train such systems, especially in developing countries such as India. Apart from this, coming up with efficient semantic segmentation models has become a new research area, and therefore, there is a need for navigating the search space of these architectures in order to obtain time and memory-efficient networks that can be deployed on low-end hardware.

The work in this thesis focuses on providing solutions to enable the ubiquitous adoption of semantic segmentation datasets which can help quickly prototype results. The experiments proposed in this thesis substantiate in validating the claim. The initial work presented in the thesis proposes two small-scale datasets which are easy to plug and play and train simple segmentation models for teaching purposes and deploy on edge devices. In the next set of work, we address the need to search for efficient segmentation architecture with the help of the proposed small-scale datasets that can help reduce the computational burden and also help in transferring the results to large-scale datasets. We also discuss the application of

automatic pruning in removing the redundant connections of semantic segmentation networks through the implementation of a unique network design, known as *HyperNetworks* along with a customized optimization algorithm.

1.1 Motivation

The presence of large-scale annotated datasets has helped deep networks in achieving human-comparable performance in many tasks. Be it image-related tasks such as classification or recognition or speech/language-related works such as translation, machine learning models have gained accuracy due to readily available large-scale datasets. Though these large-scale datasets have helped in pushing the numbers, the scale and size of the datasets, especially for the task of semantic segmentation, have restricted their widespread adoption. A few of the practical challenges are listed below:

- Firstly, compared to classification datasets that vary from a few megabytes (for e.g. MNIST [63], CIFAR10 [60]) or to few gigabytes (e.g. Imagenet [61]) require a single label per image, curation and annotation of semantic segmentation datasets constitute a large proportion of time in the overall training/deployment process. For instance, in Cityscapes [25], a widely adopted and bench-marked dataset for urban semantic segmentation, it took more than 1.5 hours to annotate and quality check a single image. For IDD [121] dataset, the annotators were first asked to re-annotate the Cityscapes images until the error difference is below 5% and then annotate images sequentially which takes around an average of 1.75 to 2 hours, which is time exhaustive process.

- Secondly, since these images are captured at a very high resolution, the dataset size grows exponentially, limiting the widespread adoption compared to classification datasets. For instance, Cityscapes [25] requires around 12 GB of hard disk space with an average image resolution of 1024×2048, whereas IDD [121] requires 18 gigabytes of hard disk space with an average image resolution of 968×1678. Due to the size factor, deep semantic segmentation models require multi-GPU training and consume a lot of GPU memory indicative of the fact that such heavy datasets can't be used either for rapid prototyping of results which in turn can be readily applied for educational purposes. Moreover, it limits the use of such high-resolution datasets for training or deploying models onto resource-constrained hardware real-time applications.

- There have been many works in classification settings where efficient models are searched on small-scale datasets such as CIFAR [60] to show that best performance also applies on bigger datasets like Imagenet [15, 73, 142]. This is also known as *architecture search*. In contrast to the classification paradigm, finding the best model for semantic segmentation is a demanding task due to the resource needs involved in the same. There have been some works in semantic segmentation like [19, 72] but due to the resource intensiveness of the task, it may take several GPU days to search for an efficient model catering to the needs of the user.

- Lastly, in order to find efficient architecture in a resource-constrained environment, automatic pruning can be considered as an alternative with the advent of neural architecture search and AutoML [18, 144]. Automatic pruning strives to compress the target network using algorithmic techniques to obtain novel sub-networks. Prior research has explored automatic pruning within resource-constrained settings in the fields of classification and restoration [67], but the application to semantic segmentation remains unexplored.

This thesis focuses on a novel endeavor, where two small-scale datasets (characterized by lower resolution and label size) are introduced, paving the way for a wide range of applications as discussed above.

1.2 Contribution

The work of this thesis can be defined using two major points-

- **Setting Up Annotation Pipeline for Semantic Segmentation Dataset for Indian Roads:** Although high-resolution datasets exist for training semantic segmentation models, these datasets are highly sophisticated in terms of scenes and the participants present. These trained models do not perform well on Indian roads since most of the assumptions of Western geographies aren't met. Hence, in order to come up with solutions for road scene understanding in the Indian setting, we require a dataset containing dynamic, unstructured scenes existing in the road network of India. Hence, we set up a pipeline for annotating and quality checking the recently proposed Indian Driving Dataset (IDD) [121].

- **IDD-mini & IDD-lite for Resource Constrained Segmentation and Challenge:** Although the annotation pipeline was setup to guide the process for annotating the IDD dataset, it didn't comply with the scenario of running the models in a resource-constrained environment. In other words, the model trained either on an Indian or any Western dataset cannot be implemented in real-time since it takes a multi-GPU training regime to get decent enough accuracy. Hence, there is a need to have a low-resolution, label-space dataset meant for running resource-constrained models that can even be trained on a laptop CPU. The proposed datasets namely *IDD-mini* and *IDD-lite*, were used as a part of the ICCV 2019 and NCVPRIPG 2019 challenges for resource-constrained semantic segmentation to come up with efficient models that give decent accuracies on these datasets. Our experiments show that models trained on these datasets with minimal GPU computing can still give decent enough accuracy when trained for 15-20 minutes and can be deployed on low-resource hardware for inference.

We then show the applicability of these datasets for teaching, neural architecture search, and automatic pruning.

- **Teaching:** In the third part of the thesis, we demonstrate the usefulness of the proposed datasets as a part of some course or workshop in teaching the community the associated concepts of semantic segmentation since these datasets can be run in real-time, giving decent accuracy.
- **Architecture Search:** Finding the optimum configuration of parameters/layers to get an efficient architecture in the case of semantic segmentation is a challenging one because of the costs involved in the training procedure. The second part of the thesis builds on the experiments which show that the proposed datasets can be very helpful in quick prototyping for hyperparameter search and help in replicating the results on larger datasets.
- **Automatic Pruning:** In order to remove redundant connections, pruning is employed with the objective of reducing the model size while maintaining accuracy. But with the advent of neural architecture search, techniques involving automatic pruning have become prominent. The fourth part of the thesis demonstrates that the proposed datasets not only facilitate quick prototyping for hyperparameter search but also serve as a valuable foundation for implementing automatic pruning techniques. By leveraging these datasets, we explore how automatic pruning can effectively optimize segmentation models, leading to enhanced model efficiency without compromising segmentation accuracy.

1.3 Thesis Outline

The thesis is organized as follows:

- Chapter 2 details an overview of the task of semantic segmentation and explains our motivation for introducing small-scale datasets despite the presence of large datasets. It also discusses supervised methods for semantic segmentation and different metrics used to evaluate different segmentation models. The chapter also gives a brief overview of the existing large-scale datasets used for training semantic segmentation models. Lastly, the chapter ends with a discussion of practical challenges related to training semantic segmentation models.
- Chapter 3 discusses details of our approach to solving some of the practical challenges presented in the previous chapter. We discuss the need for a dataset pertaining to Indian road conditions for developing technological solutions for Indian-specific road scenarios. We set up the annotation and quality check pipeline for annotating the IDD dataset [121]. We also address the need for small-scale datasets in resource-constrained settings for rapid prototyping on low-resource hardware since large-scale datasets require multi-GPU training. The utility of these datasets with proper benchmarking, statistical comparison to other datasets, and real-time deployment results is also explored. The proposed datasets have also been a part of the ICCV and NCVPRIPG 2019 challenges to find segmentation models that give good accuracies on the test set. Finally, we also showcase that the proposed datasets can be used as a part of a workshop/curriculum for teaching

semantic segmentation. We discuss our effort in this space by introducing four notebooks that cover wide aspects of semantic segmentation from traditional to deep learning methods.

- In Chapter 4, this thesis explores the utilization of neural architecture search and hyperparameter tuning techniques in various vision tasks, including classification and segmentation. The chapter presents detailed experiments conducted using the proposed datasets, demonstrating compelling results that support the effectiveness of the proposed approach for identifying efficient architectures within specific resource constraints.
- In Chapter 5, automatic pruning is carried out as an important aspect of this research on the proposed datasets. The chapter focuses on applying automatic pruning techniques to optimize segmentation models, emphasizing the reduction of model complexity and computational demands while preserving segmentation accuracy. Extensive experiments are conducted to evaluate the effectiveness of automatic pruning, and the outcomes are thoroughly analyzed, shedding light on its potential benefits for resource-constrained scenarios and real-world applications.
- In Chapter 6, we summarize our findings, conclude the thesis, and also present some possible future directions to explore.

Chapter *2*

Background

In this chapter, we will be going through an overview of semantic segmentation which forms the basis for solving the general problem of scene understanding. It is one of the most challenging and widely studied problems because the algorithm/model needs to correctly predict the class at each pixel location of the image. Though there are several multi-scale datasets available to train semantic segmentation models (as discussed in this chapter), this thesis builds up the understanding of the need for small-scale datasets ideally for two reasons. Firstly, there is a need of having small scale segmentation datasets (similar to CIFAR10, and MNIST in classification setting) in terms of size, and resolution for educational/teaching purposes that can be trained on CPUs and also deployed on low-resource hardware like Raspberry Pi for real-time projects. Secondly, finding efficient architectures using search-based methods on these small-scale datasets can result in good performance on bigger datasets such as Cityscapes [25], IDD [121]. We start the chapter by discussing the problem of semantic segmentation, more specifically supervised semantic segmentation in Section 2.1. In Section 2.2, we explore the various deep-learning approaches to perform semantic segmentation. Subsequently, we discuss the evaluation strategies/metrics used to evaluate the segmentation models in 2.3. In Section 2.4, we see various datasets available in the literature to train deep segmentation models. We conclude this chapter by summarizing all the datasets discussed in a form of table (see Table 2.1) with number of training, validation and test images, number of labels etc.

2.1 Semantic Segmentation: Overview

The task of semantic segmentation involves assigning a pre-defined class label to each pixel in the image. Unlike other computer vision tasks such as classification, where the model needs to output the majority class in the image, the task of semantic segmentation involves solving the dual problem of recognizing and understanding the contents of the image at the pixel level. Finding the spatial relationship between the pixels belonging to a particular class and assigning them the same label is what makes this problem complex. It is also known as dense prediction because it predicts the essence of each pixel in the image. An example of the task is shown in Figure 2.1. One thing to differentiate here is that there is another

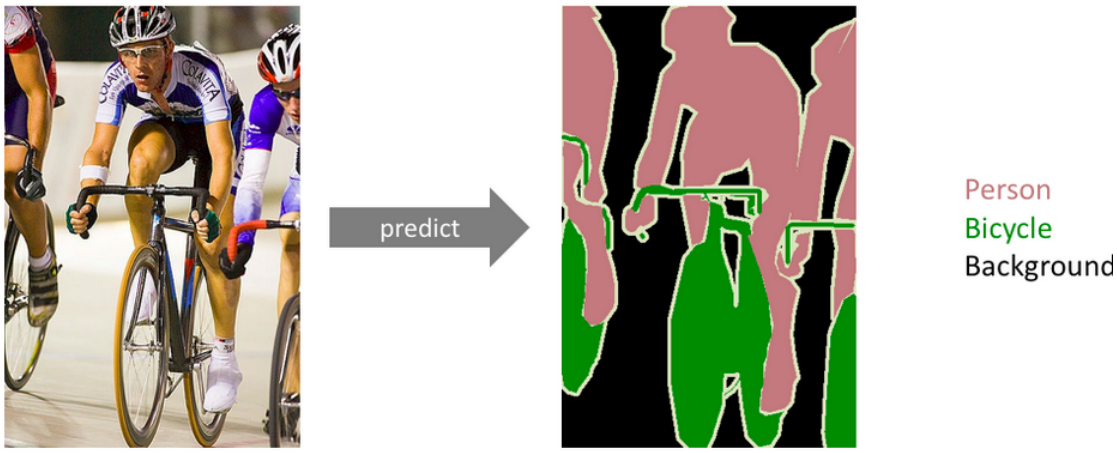


Figure 2.1 An example of the output of a semantic segmentation algorithm. The classes are encoded in the form of color such that similar pixels are clubbed together. This image has been taken from PASCAL VOC [30] dataset that will be discussed in the later section.

class of algorithms that separates the individual instances of a class present in the image and is commonly known as instance segmentation. The difference between the two algorithms is depicted in Figure 2.2.

Semantic segmentation has a lot of applications, for example, in the medical domain such as detecting tumors in the brain [89], tracing the path of medicinal instruments during operations [124]. As mentioned earlier, for the problem of scene understanding, semantic segmentation has been used heavily for ADAS (*Advanced Driver Assistance Systems*) or for self-driving applications for automotive scenarios [34, 84]. Nowadays, semantic segmentation has its application for land cover classification using satellite imagery [52]. In the next section, we are going to ponder over different approaches used to perform semantic segmentation in the images through supervised training.

2.2 Deep Learning approaches for supervised semantic segmentation

Prior to deep learning, traditional methods in segmentation initially applied thresholding-based techniques to segment out the required areas. This technique was common in the medical domain since they used grayscale images [128, 141]. The community also used KMeans, which is an unsupervised algorithm to perform clustering of similar regions and assign them one label. The number of cluster centers donated by k should be fixed before hand [40]. Researchers have also tried to solve it by posing the problem as an energy model derived from a compression-based technique [86]. Edge and region growth based approaches [71, 93, 94, 97], superpixel based technique [1], random forest [100] have also been explored for the task of segmentation. The community has also tried energy-based models like *Conditional Random Fields* (CRF) and *Markov Random Field* (MRF) to perform segmentation [45, 62, 110]. MRF defines random variables which learn the joint distribution of output class and input pixels by modeling

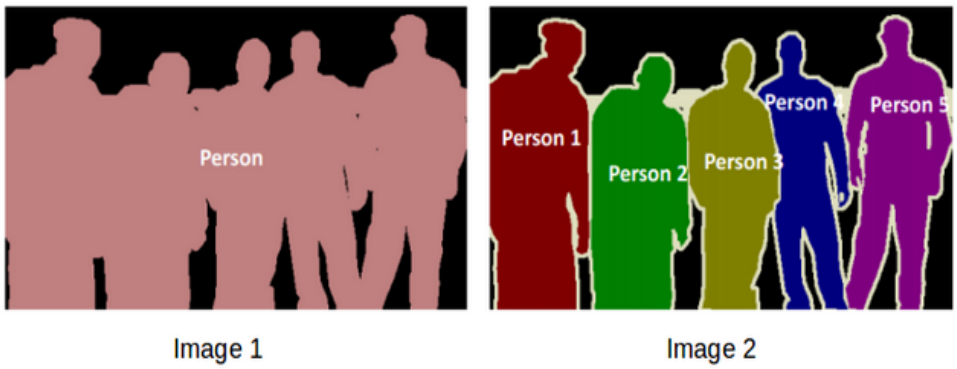


Figure 2.2 An example indicating the difference between semantic and instance segmentation. In instance segmentation, we go a step beyond and also label the individual instances in the example. As shown, all the individual instances of the person class have been given a separate label in the form of color coding.

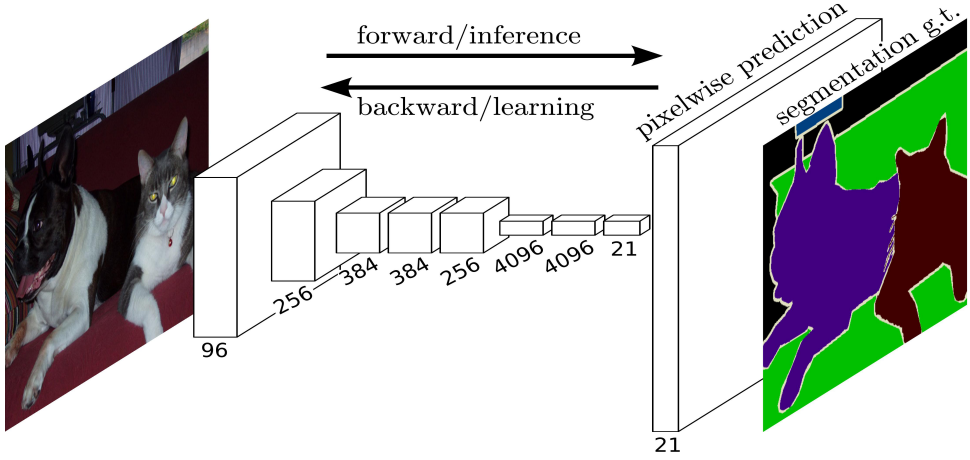


Figure 2.3 The diagram depicts *Fully Convolutional Network* [80] for semantic segmentation. This is the first end-to-end network proposed consisting of convolutional layers replacing the standard FC layer to allow input of variable size in order to pixel classifications.

an undirected graphical model. CRF, on the other hand, tries to model the conditional distribution of output label given input features.

2.2.1 Fully Convolutional Networks

From the onset of deep learning in 2012, following the success of Alexnet [61] in the ILSVRC competition [107], there have been a lot of deep learning models that have been proposed for various computer vision problems. For instance, networks such as VGGNet [111], GoogleNet [115], InceptionNet [116], ResNet [44] etc have been proposed for classification task. Similarly, for the problem of object detection where the task is to output a boundary for a particular class object, networks such as RCNN [38], FastRCNN [37], FasterRCNN [102] and SSD [75] have been proposed. These networks have not only

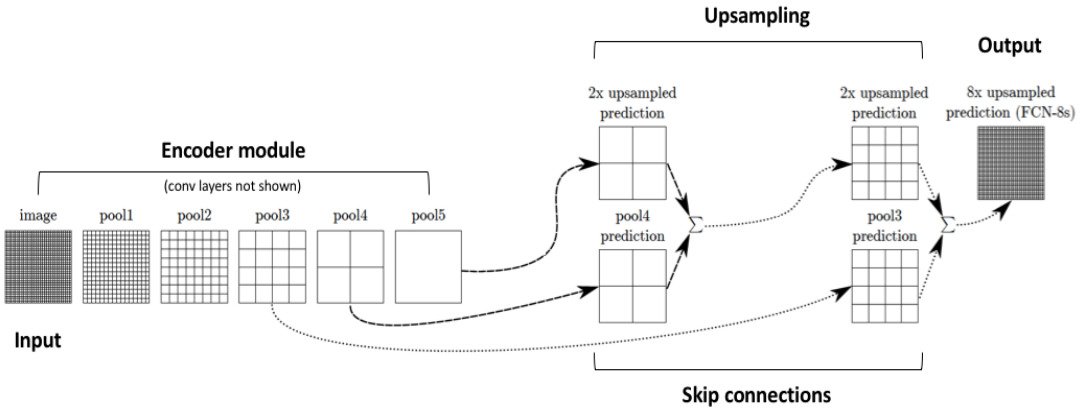


Figure 2.4 The skip connections used in the *Fully Convolutional Network* [57, 80]. To improve the prediction information in the up-sampled layers, information from the pool layers is also added. The up-sampled features are simply concatenated with the preceding pool features to get better predictions.

surpassed the performance of the traditional approaches but in some cases have also outdone human performance in these tasks.

The first deep semantic segmentation model, also known as *Fully Convolutional Network* (FCN) [80] was proposed in 2014. FCN tweaked a naive classification network by replacing all the fully-connected layers with convolution layers. Upsampling layers are used to scale the output size the same as the input for dense predictions of each pixel through the use of transposed convolutional layers. These layers form the basis for decoding the high-dimensional features. In order to achieve obtain finer segmentation maps, skip connections were added to generate crisp boundary maps for the object of interest. This network can take arbitrary size input and can be trained in an end-to-end fashion. Figure 2.3 and Figure 2.4 depict the FCN architecture and the skip connections employed.

FCNs have been a pioneering model in semantic segmentation architectures, showcasing how successful classification networks can be adapted for other computer vision tasks. By employing only convolutional layers, FCNs offer advantages over traditional fully connected layers, including reduced parameters and weights, leading to faster training and inference times for images. Nonetheless, in practical applications like medical image segmentation and autonomous driving, higher precision in localizing class labels is often required, in addition to accurate pixel-level predictions. Subsequent advancements in the field have sought to address FCNs' limitations, such as longer inference times and limitations in processing global contextual information, resulting in more refined segmentation maps.

2.2.2 Encoder-Decoder Architectures

FCNs are an example of encoder-decoder architecture. The basic methodology behind encoder-decoder architectures is to combine two sub-networks, (a) Encoder, which extracts high-level features from the

input through a series of convolutional layers that successively down-sample the spatial dimensions while increasing the number of feature channels. The encoder learns to encode hierarchical and abstract representations of the input image, capturing the global and local features, (b) Decoder takes in the encoded features and aims to recover the spatial resolution of the input image while refining the feature representations. The up-sampling of the encoded representations to the original resolution happens either through learnable layers such as transposed convolutions or through simple interpolation methods. The decoder also employs skip connections that concatenate or add feature maps from the encoder to the corresponding layers in the decoder. These skip connections help in preserving fine-grained spatial information and assist in the segmentation process. Some examples of encoder-decoder architectures are mentioned below:

UNet [104] uses skip connections connecting the corresponding encoder and decoder layers allowing them to pass the gradient directly. This structure helps in avoiding vanishing gradient problems. The decoder uses concatenation in the channel dimension to leverage both the low-level details from the encoder and the high-level context learned during the encoding process. This architecture was proposed for the application of medical imaging however it has been used in several other domains such as satellite imaging [82], automatic crop investigation [76]. Another architecture based on similar techniques was proposed by Badrinarayan et al. known as SegNet [4]. One of the key innovations in SegNet is the use of max-pooling indices during the encoder phase. While down-sampling the feature maps, SegNet keeps track of the indices of the maximum values at each pooling operation. These indices are then used in the decoder phase for up-sampling. Instead of learning the up-sampling process, SegNet uses these stored indices to efficiently perform the up-sampling, which significantly reduces the computational burden. *Efficient Residual Factorized ConvNet* (ERFNet) [103] uses factorized convolution splitting the standard 3×3 convolution filter into two separate 1×3 and 3×1 convolutions, reducing the computations while capturing important spatial information. It also uses a non-bottleneck module comprising of series of one-dimensional convolutional layers to capture both horizontal and vertical contextual information efficiently. Figure 2.5 pictorially depicts the difference between different kinds of bottleneck and non-bottleneck modules.

2.2.3 Atrous Convolution, Multi-Resolution, Multi-Scaled Architectures

There is another class of semantic segmentation architectures that employ atrous/dilated convolutions specifically aiding in widening the filter field of view. This design helps remove the down-sampling operation needed in the encoder-decoder type of architectures keeping the parameter count but increasing the training and evaluation times. For instance, *Dialated Residual Network* (DRNet) [137] uses dilations using different atrous rates to increase the receptive field size keeping the parameters same and balancing the removal of down-sampling layers, capturing a broad range of contextual information. Similarly, some networks employ spatial pyramids using dilated convolution nearly at the end of the network which helps in getting multi-scale context by performing pooling operations with different kernel sizes and strides on

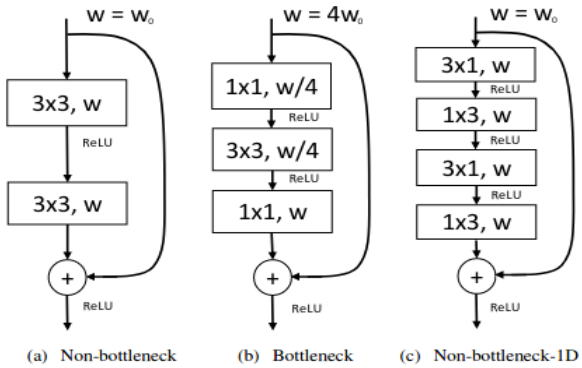


Figure 2.5 Comparison of different kinds of residual blocks, (a) Standard residual layer consisting of 3x3 convolutions, (b) Bottleneck module consisting 1x1 convolution and 3x3 convolution, (c) Non-bottleneck module with a factorized 3x3 convolution used in ERFNet [103].

the intermediate feature maps to obtain one fixed-length representation. One example of such a network is *DeeplabV3* architecture [21] that employs dilated pyramidal pooling for different spatial resolutions [20] to incorporate larger context. The authors conducted several experiments to conclude that changing the dilation rate is effective for gaining a wider context and increasing the segmentation accuracy. The authors of DeepLab introduced *DeepLab V3+*[22] consisting of an encoder-decoder architecture along with an atrous spatial pyramidal pooling module to achieve better results.

In the next section, we are going to discuss the evaluation strategies which are used to score a deep segmentation model.

2.3 Evaluation Strategies

In order to quantify how well a trained semantic segmentation model performs, we need to employ some evaluation metrics that indicate the goodness of the performance. Some of the common metrics that are used in semantic segmentation are explained below:

2.3.1 Pixel Accuracy

Pixel Accuracy is defined as the ratio of correctly classified pixels in the image to the total number of pixels in the image. We can either define it individually for each class or as a whole across all classes. Let us consider an image of resolution H×W and the total number of pixels as p. Let us assume that p_{xy} conveys the pixels belonging to class x assigned to class y by the semantic segmentation model. Therefore, with respect to class x, pixels p_{xx} denote the true positives, pixels p_{yy} denote the true negatives, pixels p_{xy} denote the false positives and pixels p_{yx} denote the false negatives. Assuming that the total

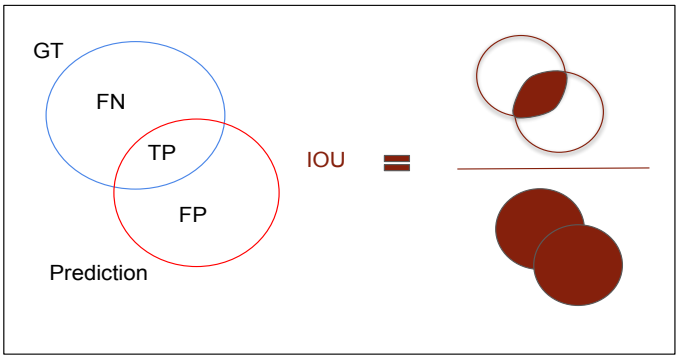


Figure 2.6 Visualization of the standard metric *Intersection-Over-Union* for evaluating the semantic segmentation models. The Venn diagram shows the pixels correctly classified as TP (True Positive) as the intersection and the union of both circles is the factor by which TP is divided to get IOU value.

number of trainable classes are $1, 2...M$, the pixel accuracy formula is defined as:

$$PA = \frac{\sum_{r=1}^{M} p_{rr}}{\sum_{r=1}^{M} \sum_{v=1}^{M} p_{rv}} \tag{2.1}$$

In a multi-class setting, the *mean pixel accuracy* (mPA) is defined as class average accuracy or the ratio of correctly labeled pixels on a per-class basis averaged across the number of classes.

$$mPA = \frac{1}{M} \sum_{r=1}^{M} \frac{p_{rr}}{\sum_{v=1}^{M} p_{rv}} \tag{2.2}$$

In scenarios with severe class imbalances, certain classes tend to dominate the image, while other classes represent only a minor fraction of the overall image. Although pixel accuracy is a straightforward metric, it can be heavily influenced by classes that occupy a substantial portion of the image.

2.3.2 Intersection-Over-Union

To overcome the limitation offered by pixel accuracy, a more sophisticated and widely accepted evaluation measure is *Intersection-Over-Union (IOU)*, also known as Jaccard Index. IoU is mathematically defined as:

$$IoU = \frac{\sum_{r=1}^{M} p_{rr}}{\sum_{r=1}^{M} \sum_{v=1}^{M} p_{rv} + \sum_{r=1}^{M} \sum_{v=1}^{M} p_{vr} - \sum_{r=1}^{M} p_{ii}} \tag{2.3}$$

It is the ratio of true positive pixels belonging to class *M* to that of ground truth pixels in class *M* as well pixels predicted to be class *M*. Figure 2.6 gives a better pictorial understanding of the metric. In the case of mean IoU, the equation changes to calculate the class-weighted average of the IoU, given by:

$$mIoU = \frac{1}{M} \sum_{r=1}^{M} \frac{p_{rr}}{\sum_{v=1}^{M} p_{rv} + \sum_{v=1}^{M} p_{vr} - p_{ii}} \tag{2.4}$$

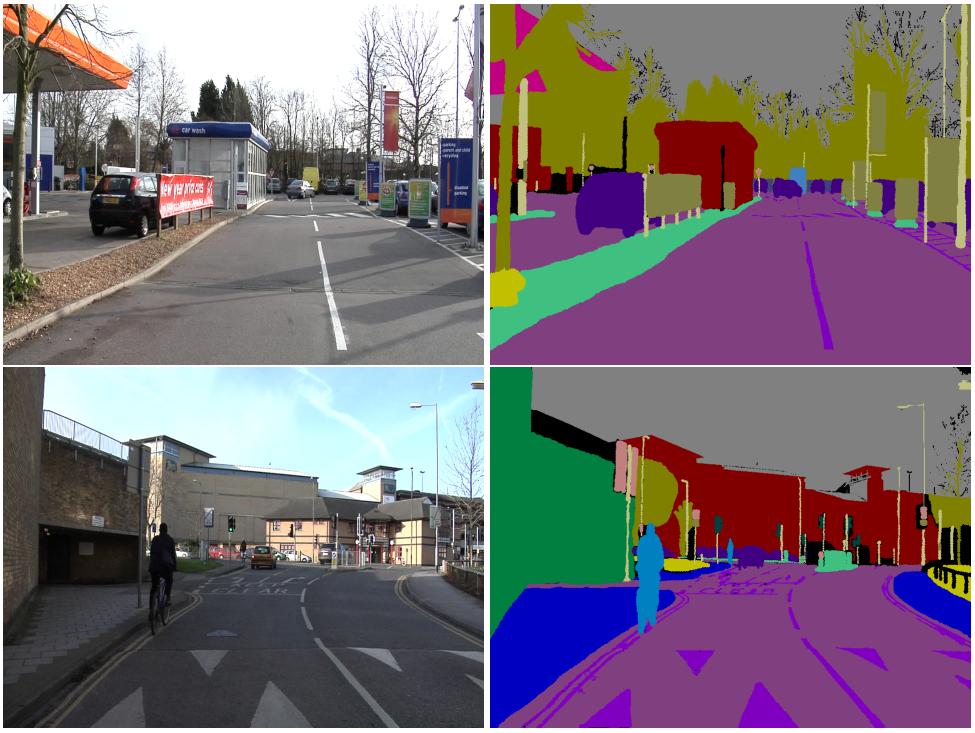


Figure 2.7 Example images from the Camvid dataset [13, 14]. The left column shows the ground truth images, and the right column indicates the color-coded per-pixel annotated images.

mIoU is the widely used metric to evaluate the performance of any semantic segmentation network on a given dataset. Table 3.3 indicates the best reported *mIoU* metric on the publicly available datasets.

2.4 Semantic Segmentation Datasets

In order to train semantic segmentation models in a supervised paradigm, we need to have a pixel-wise annotated dataset. There have been a lot of varied datasets that have been proposed belonging to different geographies, with traffic participants capturing restrained variation. This section details such proposed datasets. The mIoU metric discussed above is reported on the validation/test set of these datasets and is also mentioned in Table 2.1.

2.4.1 CamVid Dataset

CamVid [13, 14] is one the earliest proposed datasets for autonomous driving-based semantic segmentation. It consists of a total of four videos of around 22 minutes in length capturing varying scenes in and around Cambridge. The images have been annotated from a group where each pixel belongs to one of the 32 classes. The classes are broadly clubbed into four categories namely road, moving objects, fixed objects, and ceiling. The dataset contains a total of 700 images with an average resolution of 360×480.

Figure 2.8 Example images from the Cityscapes dataset [25]. The left column shows the ground truth images, right column indicates the color-coded per-pixel annotated images. Cityscapes introduced a wide variety of novel classes with diverse sets of weather, and traffic conditions.

The training set contains 367 images, while the validation and test set contains 101 and 233 test images respectively. Figure 2.7 displays sample images and the corresponding annotations from the CamVid dataset.

2.4.2 KITTI Dataset

KITTI [2, 36] is a large-scale benchmark dataset aimed at providing benchmarks to various computer vision tasks related to the autonomous driving-based scenario. The semantic segmentation benchmark of the dataset was introduced later compared to other benchmarks such as odometry, SLAM, detection, etc, and contains around 200 semantically annotated train as well as 200 test images. The data format is the same used in Cityscapes [25].

2.4.3 Cityscapes Dataset

Cityscapes [25] is a high-quality annotated dataset for training/benchmarking semantic segmentation models for street scene understanding due to varied sets of captured scenes and classes. The dataset contains dense fine annotations of urban street scenes of 25 classes out of which 19 have been chosen for evaluation, coarse annotations along with the metadata such as video sequence, GPS coordinates, etc. Each class in fine annotation belongs to one of the seven categories: vehicle, human, construction, sky, object, nature, and flat. The dataset is split into 2975 training, 500 validation, and 1274 testing images with a resolution of 2048×1024 pixels. Sample images from the dataset are shown in Figure 2.8.

2.4.4 PASCAL VOC 2012 Dataset

PASCAL VOC 2012 [30] is a popular dataset that is used for training general semantic segmentation models. It was introduced as a part of a challenge to recognize objects in realistic scenes. The challenge

Figure 2.9 Example images along with the color-coded per-pixel annotations of the PASCAL VOC dataset [30]. This dataset has images from varied scenes both indoors and outdoors focusing on segmentation in different scenarios.

also includes data for tasks such as detection, classification, action classification, etc. There are 20 object classes such as a person, dog, bus, chair, etc. belonging to 4 different categories. For the task of segmentation, the train and validation set includes images and annotations from 2007-2011 versions as well, accounting for 5717 and 5823 images. The test set contains images from 2008-2011 without any annotation. The class *person* is the most dominant category in the dataset followed by chair and car. Figure 2.9 shows some examples from the dataset.

2.4.5 Mapillary Dataset

Mapillary Vistas is a large-scale diverse dataset [92] which contains street scenes meant for training segmentation models specific to autonomous navigation-based scenarios. Training and val sets contain 18000 and 2000 images respectively while the test set has 5000 images at an average resolution of 9MB. There are a total of 100 object categories, 60 of which are specific to instance segmentation. The dataset has a much richer geographic reach covering North-South America, Europe, Africa, Asia, etc., and also differs in weather conditions, in capturing times. The images have been captured from different viewpoints as well for eg. roads, sidewalks, etc. The number of objects per image is more than that of per image object count of Cityscape [25]. Figure 2.10 depicts a few examples from the dataset.

2.4.6 ADE20K Dataset

ADE20K is a densely annotated dataset that spans diverse annotations of scenes, objects, parts of objects etc [143]. The dataset consists of 20210 finely annotated images for training, 2000 images for validation, and around 3000 testing images. There are a total of 3,169 class labels annotated, 2,693 of which are

Figure 2.10 Sample images and the corresponding color-coded per-pixel annotations depicting the urban street scenes in Mapillary dataset [92]. The first row corresponds to the dataset image captured in Asia while the second row corresponds to the dataset image captured in Oceania.

Figure 2.11 Sample images and pixel-wise color annotations depicting a few scenes of ADE20K [143]. The first row corresponds to a scene inside a caravan whereas the second row corresponds to an inside view of a house. This dataset contains a wide variety of both indoor and outdoor scene segmentation.

object and stuff classes and 476 are object part classes. For benchmarking on their test set, the researchers use only 150 categories for the task of semantic segmentation. All the images are exhaustively annotated with objects. Many objects are also annotated with their parts. Apart from the regular annotation, additional information regarding the occludes, cropping, and other attributes. The distribution of objects is non-uniform mimicking a more natural scene. An example of this dataset is shown in Figure 2.11.

2.4.7 BDD100K

BDD100K is the largest urban street driving dataset [135] for a multitude of tasks such as semantic segmentation, object detection, lane marking, multiple object tracking, etc. The dataset has a total of 100K videos possessing varied amounts of geographic conditions and weather diversity collected mostly from the populous regions of the US. Each video is of length 40 sec and has been recorded in 720p

Figure 2.12 A few example images and the corresponding pixel-wise color annotations depicting two scenes of BDD100K [135]. This is by far the largest autonomous driving dataset for a multitude of tasks such as semantic segmentation, object detection, tracking, etc.

resolution along with GPS recordings to keep track of the driving path taken. The training set contains 70K videos, whereas the validation and testing set contains 10K and 20K videos respectively. For each video, the frame available at the 10th second is chosen for annotation, while the whole video is used for tracking purposes. Figure 2.12 shows a few images and their corresponding annotation of this large-scale dataset.

Dataset	*Average Image Resolution (W×H)*	*# Train + Val. Images*	*# Test Images*	*mIoU (Best reported)*	*Labels*
CamVid [13, 14]	360×480	468	233	83.7	32
KITTI [2, 36]	375×1242	200	200	76.44	19
Cityscapes [25]	2048×1024	3475	1525	85.9	19
PASCAL VOC 2012 [30]	390×468	11540	N.A.	84.3	20
Mapillary [92]	1080×1920	20000	5000	64.9	40
ADE20K [143]	822×1038	22210	3000	63.0	150
BDD [135]	720×1280	8000	2000	58.7	19

Table 2.1 Shows the average image resolution, number of training and validation images, number of test images, the best *mean intersection-over-union* (mIoU) metric reported on the test set and the number of corresponding labels in the respective datasets. KITTI [2, 36] and BDD [135] follow the same label convention as that of Cityscapes [25]. The number of test images for PASCAL VOC 2012 [30] dataset for semantic segmentation is not publicly available.

In the chapter, we discussed the task of semantic segmentation, deep learning approaches to semantic segmentation, evaluation metrics, and some of the existing datasets to train deep segmentation models. In the next chapter, we will discuss our approach to solving the practical challenges for rapid prototyping of semantic segmentation models in resource-constrained settings by introducing two small-scale datasets that are easy to plug and play and the corresponding experiments and results to validate our hypothesis.

Chapter *3*

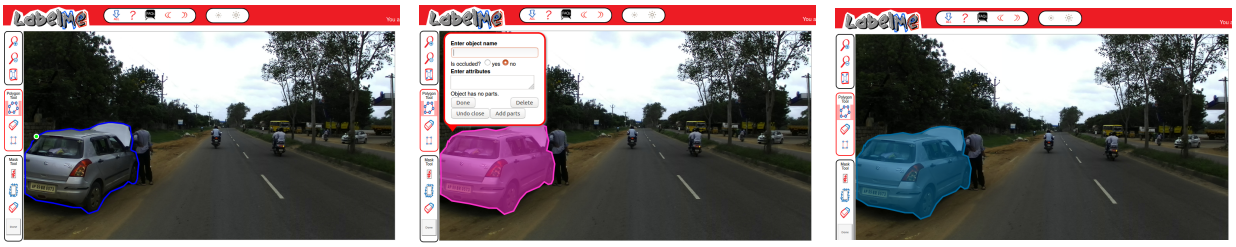
Datasets for Resource Constrained Semantic Segmentation

In this chapter, we discuss the need of having a dataset for developing solutions catering to Indian road understanding and also discuss the proposed solution to tackle the practical challenges one faces when training deep semantic segmentation models described in the previous chapter. Our contribution is summarized as follows: Several large-scale datasets, coupled with advances in deep neural network architectures have been greatly successful in pushing the boundaries of performance in semantic segmentation in recent years. However, all the existing datasets contain scenes from Western cities which are much more organized in terms of structured road conditions with well-defined traffic participants, low variation between foreground and background objects, etc. The models trained on the Western datasets might provide an initial level of results but because of the assumptions mentioned above which seldom get satisfied in the Indian context, these results cannot be further improved beyond a certain point. Therefore, a dataset for developing autonomous driving solutions catering to the Indian road scenario is of utmost need/importance. We discuss the statistics of the recently proposed dataset for Indian road conditions in this chapter. Now, though we have a dataset for Indian roads, the scale and magnitude of all the datasets prohibit ubiquitous use and widespread adoption of such models, especially in settings with serious hardware and software resource constraints. Therefore, we propose two simple variants of the IDD dataset, namely *IDD-mini* and *IDD-lite*, for scene understanding in unstructured environments. This chapter focuses on the benchmarking of these datasets using standard training regimes on GPU as well as low-constrained hardware such as RPi. Through the experiments, we show qualitatively and quantitatively that with only 1 hour of training on 4GB GPU memory, we can achieve satisfactory semantic segmentation performance on the proposed datasets. We also discuss the use of these datasets for teaching and education usage, since models trained on these datasets for a very limited duration still gives decent enough accuracy that can be part of some courses/workshop. We also provide qualitative and quantitative assessments of our dataset over state-of-the-art segmentation architectures.

3.1 Annotation Toolkit for Semantic Segmentation

This section introduces the formal methodology used to annotate semantic segmentation datasets for model development.

Pixel-wise annotation for dense labeling tasks such as semantic segmentation and instance segmentation requires a lot of time to annotate since each pixel needs to be given a class label. In order to fasten the process of the annotation, a number of methods have been introduced in the literature [17, 106, 122]. Some of the methods used superpixel-based techniques in which they divide the image into superpixels and annotate the label present in them [35, 130]. Other works use semi-supervised methods in which they annotate the images using either points [8, 129] or strokes [8, 55]. The inference in the previously explained method is done using the same procedure. The mentioned techniques help in faster annotation but quality becomes a question.



((a)) Polygon Annotation Outline ((b)) Class labelling ((c)) Final Annotation

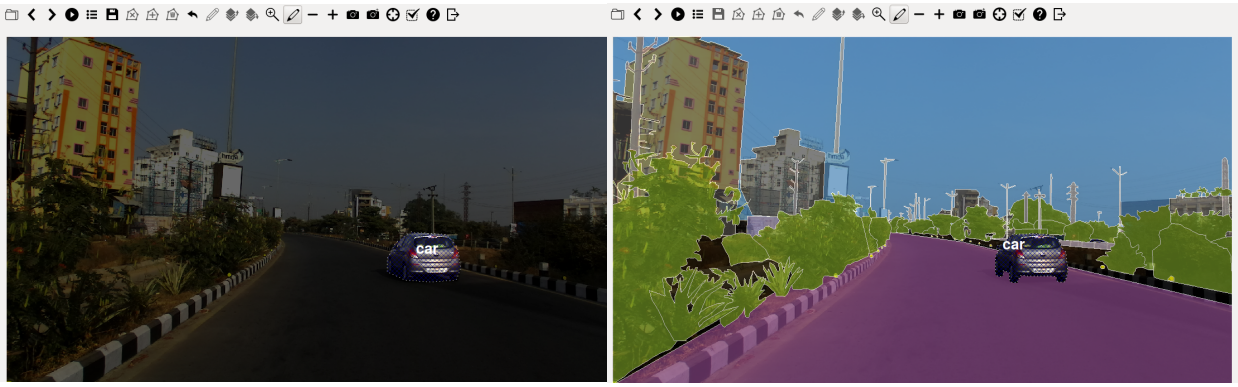
Figure 3.1 An example of annotating a car class object from the IDD [121] dataset using LabelMe [108] web based annotation tool. On the left, (a) Polygonal annotation outline depicted using blue color for the car object. The green circle is the starting point of the outline and needs to be clicked on when the beginning and end points meet. (b) The pop-up box appears once the annotation is complete. It asks for the class name of the object along with other attributes to fill. (c) The final pixel map of the car object.

Almost all publicly available datasets for semantic segmentation [25, 30, 121] nowadays use a team of annotators to define layered polygons around the object of interest. The majority of the objects present in these datasets are connected without the presence of any hole hence polygons help in annotating an object by using a small number of clicks. The majorly used annotation tools for this setup are described below:

- *LabelMe Annotation Tool:* LabelMe [108] is a web-based open-source image annotation tool for labeling objects using polygon, rectangle, circle, line or point annotation. The annotation is saved in an XML format with all the polygon corners/clicks saved as coordinates. A pipeline of the tool is shown in Figure 3.1. Some of the advantages of this tool are, (1) the annotators can annotate the scene without installing the tool locally, (2) custom functions can be implemented to improve the

segmentation or change the label set. The main disadvantage of the tool is that there is no efficient way of conducting quality check analysis.

- *Cityscape Annotation Tool:* This is an offline open-source annotation tool for labeling objects in a layer-wise polygonal format. It is quite similar to the LabelMe tool but with different GUI functions that can help the annotator fasten the labeling. An example of the tool is depicted in Figure 3.2. The output of the annotation is saved in a JSON format containing all the coordinates for all the clicks made.



((a)) Labelling Car Object ((b)) Full depth ordered annotation

Figure 3.2 An example of annotating a car object in the image captured from the city of Hyderabad. On the left, the car object has been annotated initially with the label shown when zoomed. On the right is the image depicting annotation using the depth ordering of the layered polygons.

In the next section, we will introduce the recently proposed IDD dataset, a dataset for understanding the unstructured environment present in the Indian context using the Cityscape annotation tool along with the annotation and quality check guideline used to annotate the images of the dataset.

3.2 Indian Driving Dataset [121]

All the datasets discussed in Chapter 2 contain scenes captured in the western setting which are more organized and constrained as the traffic participants follow the guidelines set. Compared to the datasets captured in the West, Indian Driving Dataset (abbreviated as IDD) [121] is a large-scale, recently proposed dataset for scene understanding in unconstrained and unstructured environments. By unstructured, the dataset assumes having images from scenes having not-so-delineated boundaries, not-so-strict adherence to traffic rules, and multiple traffic participants (both humans and animals). The dataset consists of 10000

Figure 3.3 Example images and the pixel-wise annotations depicting a few scenes of IDD[121]. As can be seen, this dataset introduces new classes like auto-rickshaws, billboards, etc not present in other datasets. The dataset also contains fallback labels in case the object doesn't fall into the defined set of categories.

images out of which around 7000 belong to the train set, and 1000 and 2000 belong to validation and testing sets respectively. The images have been finely annotated with 34 classes collected from 182 drive sequences of Hyderabad and Bangalore, India with an average resolution of 1678 X 968 pixels. The labels in the dataset have been divided into four level hierarchies to allow various training schemes on multiple levels and adapt to increasing complexity. Figure3.3 shows a few annotated examples of the dataset. We discuss here in the upcoming subsection, the annotation guideline and the quality check strategy adopted for annotating this dataset.

3.3 IDD: Data Annotation and Quality Check

This section explains in detail the whole process of annotation and quality check guidelines followed for the IDD dataset.

3.3.1 Annotation Guideline

The drive sequence recording received from the camera was sent to extract and prioritize frames for the process of annotation manually. After getting the frames, the fine annotations were curated using polygonal format using Cityscapes [25] offline tool as shown in Figure 3.2. The annotators were first asked to annotate images from Cityscapes [25] dataset in order to get acquainted with the finer-level details of the process. Subsequently, they were provided with additional guidelines for annotation. Some of them include images being ordered by depth ordering, occluded parts are annotated with a coarser,

Timeline	Raw Frames	Annotated Images	Man Hrs for Annotation	Man Hrs for QC
07 Aug - 13 Aug 2019	43	43	35 Hrs	210 Hrs
31 July - 06 Aug 2019	29	9	20 Hrs	273 Hrs
24 July - 30 July 2019	624	589	741 Hrs	174 Hrs
17 July - 23 July 2019	1661	1067	1568 Hrs	26 Hrs

Table 3.1 An example of the statistics from the different modules of the annotation and quality check pipeline. The man hours for annotation show the total hours needed for a team of 25 annotators combined together to annotate the data in different time shifts. As expected, the hours for quality check is less than that of annotation. The difference in the raw frames and annotated images is because those images went to the repair queue due to a large set of errors identified by the peer annotator. QC=Quality Check

Timeline	QC1 Passed	QC2 Passed	QC3 Passed	Remarks
07 Aug - 13 Aug 2019	43	1401	231	-
31 July - 06 Aug 2019	9	1972	-	-
24 July - 30 July 2019	589	2137	-	-

Table 3.2 Data corresponding to different weeks for different quality check modules of the pipeline. For the present week, the remaining set of annotations are passed for the subsequent week to be checked. If the batch as a whole is processed after QC3, then the images can be passed down for training.

conservative boundary (possibly larger than the actual object). Holes in an object through which a background region can be seen are considered to be part of the object. On average, per image took 1.5-2 hrs for annotation by a single annotator.

3.3.2 Quality Check Guideline

After the first round of annotation, the set of annotations is passed to a peer annotator selected randomly to do a thorough quality assessment and provide feedback. The peer's role is to check and verify:

- If the guidelines of annotation are followed.
- If the right labels are assigned to polygons/objects.
- Is every part of the image annotated without leaving any object unannotated?
- Are the object boundaries reasonable?

The aim of the peer review was to encourage annotators to learn from each other's mistakes. This is followed by a quality check provided by the supervisor. The supervisor's role is to check:

- All the guidelines are strictly followed.
- All the objects in the image are given appropriate labels.
- All the objects in the image are labeled without missing any part/object.

- Lot of effort is invested in identifying easily missed small-scale objects eg. traffic signs, and traffic lights.
- Are the object boundaries appropriate up to a pixel level threshold - strict check on boundary errors.

The aim of the supervisor-based QA is to lay more emphasis on ensuring pixel-level quality.

This is followed by quality assurance provided by one of the members of the research team where they sample 20% of images from the batch. If around say 98% of images in a batch pass quality assessment based on pixel level, then these batches of images are approved.

The last stage of quality assurance is done by an expert by sampling 10% of images which are thoroughly scrutinized. Following this, the batch of data is concluded as annotation and passed on for further processing. The whole process is also depicted in Figure 3.4. Table 3.1 and Table 3.2 indicate some statistics related to the whole annotation and quality check pipeline.

Although IDD was proposed to tackle challenges related to the Indian setting, it's still a very resource-intensive dataset in terms of GPU computation. It also does not take care of the resource-constrained needs for running semantic segmentation models. Moreover, there is a need to drive the research in the vision community toward achieving state-of-the-art results for various tasks using only limited labeled data. Such a research direction would have a huge impact, more so on semantic segmentation tasks that requires huge annotation of pixel-level semantic labels. Developing such standardized small-scale datasets, we wish to coalesce the efforts of the research community towards developing algorithms that need only a few labels to match state-of-the-art performance. As a result of the annotation toolkit used to annotate the IDD dataset, it lead to the motivation of curating two small datasets for semantic segmentation for resource-constrained computation namely *IDD-mini* and *IDD-lite*. In the next section, we will discuss more about the related efforts in this space, the dataset curation strategy, and its associated statistical properties.

3.4 IDD for Resource Constrained Computation

As discussed earlier, semantic segmentation is the task of assigning pixel-level semantic labels to images, with potential applications in fields such as autonomous driving [25, 121] and scene understanding. Many approaches have been proposed to tackle this task based on modern deep neural networks [20, 103, 137]. The majority of the proposed approaches use encoder-decoder networks that aggregate spatial information across various resolutions for pixel-level labeling of images.

For example, [80] proposes an end-to-end trainable network for semantic segmentation by replacing the fully connected layers of pre-trained AlexNet [61] with fully convolutional layers. Segmentation architectures based on dilated convolutions [136] for real-time performance have also been proposed in [103, 137]. However, most of these approaches come with huge overhead in training time and inference

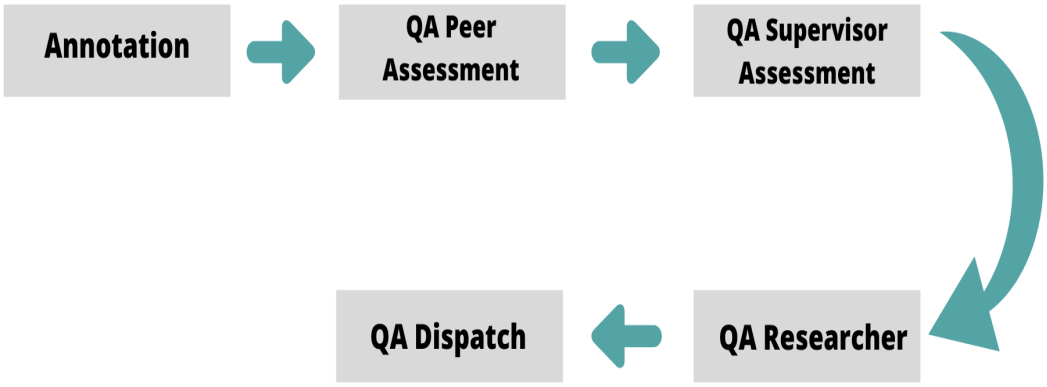


Figure 3.4 Total annotation and quality checking pipeline.

time since it requires multi-GPU training with very high GPU memory requirements. This poses multiple challenges for the widespread use of semantic segmentation datasets and architectures, resulting in huge roadblocks to research and development of such real-time systems, especially in developing regions of the world with resource constraints.

Apart from the widespread utilization of the datasets, optimizing deep learning models for resource-constrained devices is picking up pace since the real-time performance of such deep models brings the issue of scalability and transferability for the end user. For instance [11] proposed a traditional Boltzmann machine model for deep learning models deployed on FPGA boards in order to achieve better speedup. Other strategies include model compression to reduce model size for edge devices such as [42, 101], proposing efficient modules that can run on low-end hardware [51, 139]. More specifically, for semantic segmentation [83] proposed efficient ESP module which has a comparative IoU compared to heavier models such as [20], takes around 0.8 MB in size on an NVIDIA Jetson TX2 edge device and has a very less power consumption. Similarly [70] proposes a model specifically for edge devices that runs on 38 FPS with a model size of 15 MB on NVIDIA Jetson TX2. Other approaches include architecture search-based processes which take computational constraints into account during the search process [26, 126].

Keeping in mind the need for segmentation datasets for constrained settings, we propose *IDD-mini* and *IDD-lite*, as shown in Figure 5.4, which are aimed at improving the state of semantic segmentation for autonomous driving in developing regions. We believe that having these datasets would help alleviate the challenges faced in resource-constrained settings. Resource constraints can mean a lack of availability of high-end GPUs, limited time access to GPU resources, or lack of infrastructure to store large-scale datasets. The scenes and labels presented in our dataset are very different from those available in semantic segmentation datasets such as Cityscapes [25], KITTI [36] or CamVid [13, 14]. Therefore, we tend to address the following points in this chapter:

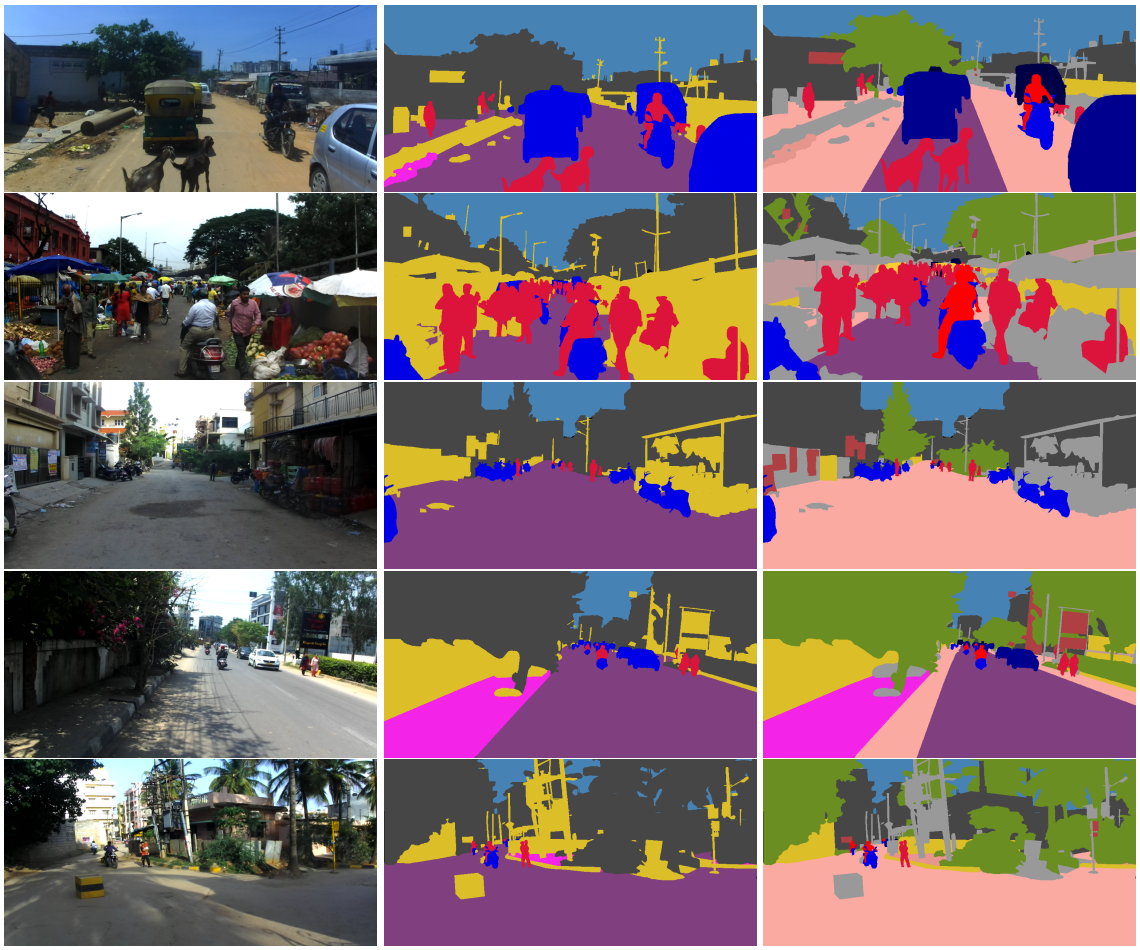


Figure 3.5 Sample images with ground truth from IDD-lite, IDD-mini with a second and third column representing 7 and 16 labels *(Best viewed when zoomed)*.

Dataset	Average Resolution	#Annotated Pixels[10^6]	#Train Images	#Val Images	Disk Space (in GB)	Label Size
IDD[121]	968×1678	11811	6993	981	18	26
Cityscapes[25]	1024×2048	9430	2975	500	12	20
IDD-mini	512×720	535	1794	253	4	16
IDD-lite	227×320	39.75	673	110	<1.5	7

Table 3.3 Comparison of state-of-the-art datasets against proposed IDD-mini and IDD-lite.

- We provide IDD-mini and IDD-lite, which are sub-sampled versions of IDD [121] with very similar label statistics and a smaller number of labels (See Section 3.4.3).
- We show that models trained only for an hour on a single 4GB GPU still achieve reasonable prediction accuracies (See Section 3.5.1).
- We deploy models trained using our datasets on Raspberry Pi and report the accuracy and runtime, giving a standardized measurement of the performance characteristics on the device (See Section 3.6.1).

3.4.1 IDD-mini

The motivation behind designing IDD-mini is to have a small-scale segmentation dataset that is useful for training image segmentation models on low-resource hardware. The full IDD dataset[1] consists of 7974 high-resolution images in the train-val set with 26 labels at the L3 label hierarchy, taken from 182 different drive sequences. To create IDD-mini, we followed a carefully evaluated sub-sampling approach to ensure the resultant dataset retains its representativeness and diversity. After conducting a comprehensive empirical analysis, we determined that sub-sampling the dataset by a factor of 4 yielded the most desirable outcomes for our specific goals. The empirical evaluation involved examining various sub-sampling factors, considering factors such as computational efficiency, label distribution preservation. This approach also ensured that the resulting dataset maintains the same proportion of labels as the original IDD dataset [121], preserving the semantic information while significantly reducing the dataset size. To create IDD-mini out of the original dataset, we resize the images such that the largest dimension is down-sampled to 720 while preserving the aspect ratio, and use the 16 labels from the L2 hierarchy of the original dataset. The train set contains 1794 training images and 253 validation images.

3.4.2 IDD-lite

The decision to utilize a sub-sampling factor of 10 for creating IDD-lite was a result of rigorous empirical investigations aimed at optimizing the dataset for very quick prototyping of semantic segmentation

[1] https://idd.insaan.iiit.ac.in/

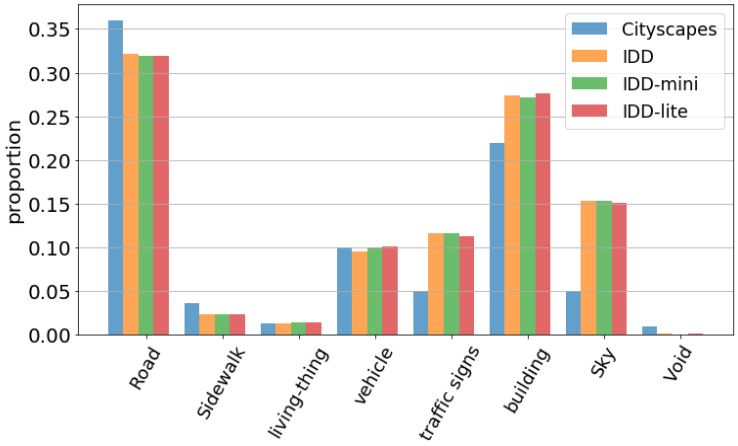


Figure 3.6 Proportion of labels in total dataset for IDD, IDD-mini and IDD-lite *(Best viewed when zoomed)*.

models, especially in resource-constrained settings. Through extensive empirical analyses, we systematically explored various sub-sampling factors and evaluated their implications on dataset characteristics, model performance, and the overall effectiveness of IDD-lite for quick prototyping and educational purposes. The evaluation encompassed considerations such as computational efficiency, label distribution preservation, inter-class variance, and the ability to maintain a diverse representation of the original dataset. The choice of a sub-sampling factor of 10 emerged as a clear and purposeful outcome from this empirical evaluation. By sub-sampling the dataset by a factor of 10, we effectively reduced the dataset size while ensuring that IDD-lite retains a meaningful representation of the original dataset's semantic information. IDD-lite's training set now comprises 673 images, and the test set includes 110 images. We re-scaled the largest dimension of the images to 320 while preserving the aspect ratio, and employed the L1 hierarchy with 7 coarse labels to maintain the necessary granularity for semantic segmentation tasks. The outcome of this empirical approach is a purposefully designed IDD-lite dataset, which strikes an optimal balance between dataset size reduction and retaining critical information for rapid model prototyping and instructional purposes. Moreover, the reduction in the required disk space from 18GB for IDD to less than $<$1.5GB for IDD-lite further enhances the dataset's accessibility and optimizes the storage footprint, making IDD-lite an ideal resource for settings with resource limitations.

3.4.3 Label Statistics

Figure 3.6 shows that the mini and lite versions of IDD follow the same distribution as the original dataset, following the technique we used to sub-sample the dataset. The proportion of pixels corresponding to categories like *Road* and *Building* occupy a large fraction of the total annotated pixels, while there is also sufficient representation for smaller classes like *vehicle* and *traffic signs*. The total absolute number of annotated pixels (in log scale) is given in Figure 3.7, to show that the number of pixels in IDD-mini, and IDD-lite are an order less than that of the original dataset.

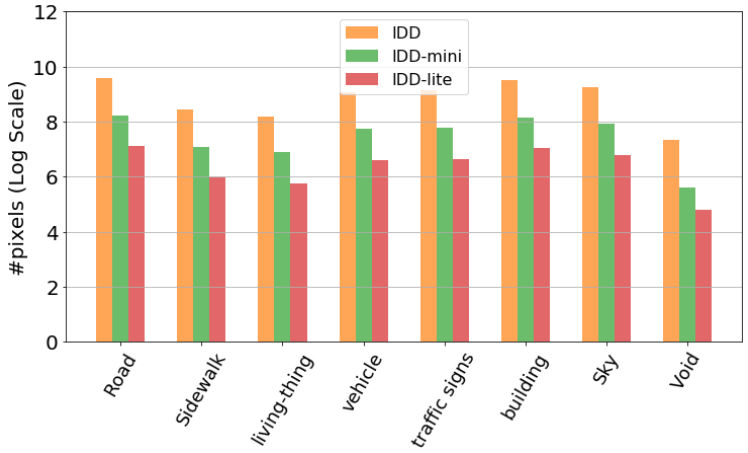


Figure 3.7 Absolute value of annotated pixels (in powers of 10) for categories in IDD, IDD-mini, and IDD-lite *(Best viewed when zoomed)*.

3.4.4 Comparison with other datasets

Comparison to another large-scale and widely used dataset, Cityscapes [25], is also presented in Figure 3.6. It consists of 2975 training images and 500 validation images at a uniform resolution of 1024×2048, with images taken from various cities and weather conditions. However, one major advantage that our datasets offer compared to cityscapes is that IDD-mini and IDD-lite contain scenes from more unstructured environments, with images captured from complex traffic and driving situations. Furthermore, the comparison from Figure 3.6 shows that in most categories, the smaller datasets match Cityscapes on the proportion of the pixels.

3.5 Experiments and Results

3.5.1 Semantic Segmentation Performance Benchmarking

In this section, we benchmark the results of the proposed datasets on two state-of-the-art architectures used for semantic segmentation, DRNet [137] and ERFNet [103]. We employ these networks since the same networks were used to benchmark against IDD [121] dataset. For DRNet, we use a ResNet-18 backbone(*drn-d-22*). We take mIoU (mean intersection over union) as the performance metric for all our experiments.

The models ERFNet and DRNet-18 were trained using the resolution depicted in Table 3.3 and validated using the resolution shown in Table 3.4. The models achieve a mIoU of 57.91% and 53.31% on IDD-mini using ERFNet and DRNet-18 respectively. Similarly, IDD-lite gives mIoU values of 66.14% on ERFNet and 55.03% on DRNet respectively.

Dataset	#L	Val. Res.	mIoU ERFNet [103]	mIoU DRN [137]
CS [25]	20	512×1024	71.50	68.00
IDD [121]	26	512×1024	55.40	52.24
IDD-mini	16	480×640	57.91	53.31
IDD-lite	7	128×256	66.14	55.03

Table 3.4 Performance (in mIoU) of the proposed datasets on semantic segmentation architectures ERFNet and DRN-d-22. Note that *val. res.* corresponds to the validation resolution for each dataset, which is obtained by cropping and resizing the original images from Table 3.3, #L is the number of trainable classes in that dataset.

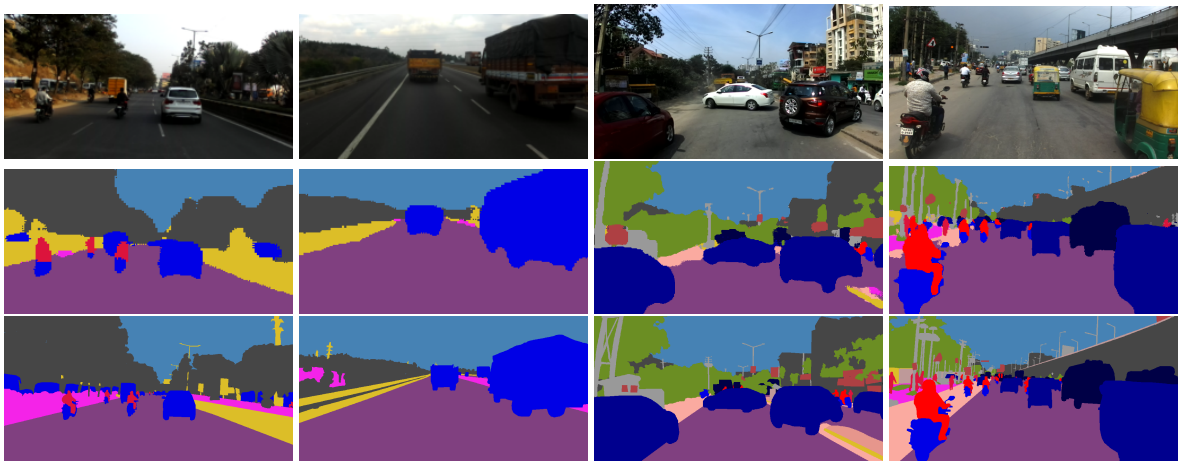


Figure 3.8 Qualitative examples of ERFNet model run on IDD-mini, IDD-lite dataset. From top to bottom - image, prediction, and ground truth. The first two columns correspond to results from IDD-mini and the last two columns are for IDD-lite.

From Figure 3.9, it can be seen that the IDD-lite dataset gives reasonably good mIoU results with just 15-20 minutes of training within 4GB GPU memory *(Best visualized when zoomed)*. We also note that while models trained on such datasets cannot directly be employed in state-of-the-art semantic segmentation applications, they will nevertheless be very useful for teaching or workshop purposes in cases with limited technical support and overall resource availability as discussed in the next chapter.

3.6 Related Effort

Deep learning has taken over the traditional methods as discussed earlier for almost all the vision tasks. However, the same performance comes at the cost of high computational and memory requirements for both the training and inference time. Training a deep learning model, for instance, a segmentation model requires multiple rounds of training, involving millions of parameters that need to be updated with each round of learning. The same happens happen at the inference time since the dimensionality of the data is huge and requires multiple operations to get the desired output. For eg, for segmentation, the input

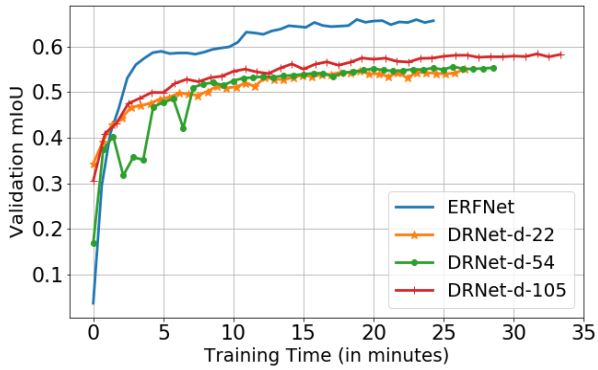


Figure 3.9 Training time (x-axis) vs. Validation mIoU (y-axis) plot for IDD-lite. Note that with only 15 minutes of training on 1 GPU using only 4GB, the model obtains >50% mIoU

undergoes a series of downsampling and upsampling operations to get the final class map. Therefore, the community is slowly shifting to designing such models that can run on low-power devices with minimal requirements. As we will see in the next chapter, neural architecture search is one of the techniques that can help in modeling such constraints into the networks to fabricate such deep learning models [126, 26]. Mentioned below are some factors which influence the use of edge devices over cloud-based services which tend to offer a similar solution.

- *Latency:* Inference time is an essential aspect of real-time performance. The response time should be as low as possible to have a near-immediate response. For e.g. in order to understand the semantics of the scene, an autonomous car should have a system that immediately understands the surroundings in order to avoid any mis-happenings. This is delayed in cloud-based service since there has to be a handshake between the devices at the two extreme ends.

- *Power consumption*: In order to deploy deep learning models on edge devices such as RPi, it is important to check whether any unnecessary processes don't run in the background which wastes the clock cycles and increases the power consumption of the device. When compared to cloud services, several processes need to run in the background in order to run the main program which ultimately relates to heavy power consumption.

- *Scalability*: While developing deep learning models for edge devices, scalability comes into play since these models have to fit a very small device and at the inference, the input has to be processed through such device. For cloud-based processing, the model sits in the cloud, and data is sent to be processed which might be an advantage but loading and unloading time then becomes a question. Hence, the effort in the case of an edge device should be that the model doesn't become the bottleneck in such a scenario.

3.6.1 Models for Raspberry Pi

- *Redundancy*: One of the factors that affect real-time processing is redundancy. Compared to cloud computing where the non-availability of a particular node can have serious implications on the complete analysis pipeline, resource-constrained devices have proved to be reliable in providing a satisfactory level of redundancy. Non-functioning of one device can be taken over by the other thus ensuring reliability in turn decreasing the off time.
- *Cost Effectiveness:* Models deployed on edge devices are more cost-effective when compared with cloud-related computing services since they are cheap and readily available for deployment.

These are the factors that help edge devices take the lead but there are some constraints for the edge devices to deploy deep learning models discussed here:

- *Efficient CNN Design*: The majority of the networks proposed are trained to keep in mind the accuracy factor rather than deployment. Hence the models become a bottleneck here. Few deep learning models proposed [51, 53, 139] have tried to address the issue by decreasing the number of parameters using different convolution/weight sharing strategies. Approaches like [26, 126] have used architectural search-based approaches keeping latency as a modeling factor and designing efficient models for deployment.
- *Pruning and Quantization*: A lot of neurons in the trained models are redundant. Removing them from the network won't affect the accuracy but instead, help in reducing the size of the model. A lot of works have been proposed recently which have a direct application for deployment on edge devices [24, 120]. More specifically, Learn2Compress [101] achieves reduction by a factor of 2 while retaining a decent accuracy for classification setting on edge devices. [99] reduced the network parameters to binary values drastically reducing the size of the model by a factor of 4.

As discussed above, in order to make deep learning models scalable for real-time application in resource-constrained environments, factors such as real-time performance, feasible cost of the hardware and low power consumption are essential. Hence, we provide the bench-marking values of runtime for segmentation models on Raspberry Pi. This device is widely available as a single-board compute platform which comes at an affordable cost, apart from being customizable and energy efficient.

More specifically, we chose Raspberry Pi 3B as the deployment hardware device for our semantic segmentation models. The device contains 1 GB RAM and has a 1.2GHz Quad-Core 4XARM Cortex-A53 CPU. We tested ERFNet and DRNet-18 networks on Raspberry Pi to calculate the inference time on the validation datasets at various resolutions. Table 3.5(b) shows the inference time of different semantic segmentation models at various resolutions on our validation datasets.

Figure 3.10 shows the run time information of each layer defined in the ERFNet architecture. Although the IDD and the IDD-lite datasets have equal trends, IDD-lite time is significantly lesser, in addition to being uniformly consistent across layers. This further reinforces our proposition that such a dataset

	IDD-lite	IDD-mini	IDD
ERFNet	1.12	5.99	10.60
DRNet-18	56.61	78.47	95.94

Table 3.5 Inference time (in sec.) of different semantic segmentation models on various versions of IDD on Raspberry Pi 3B.

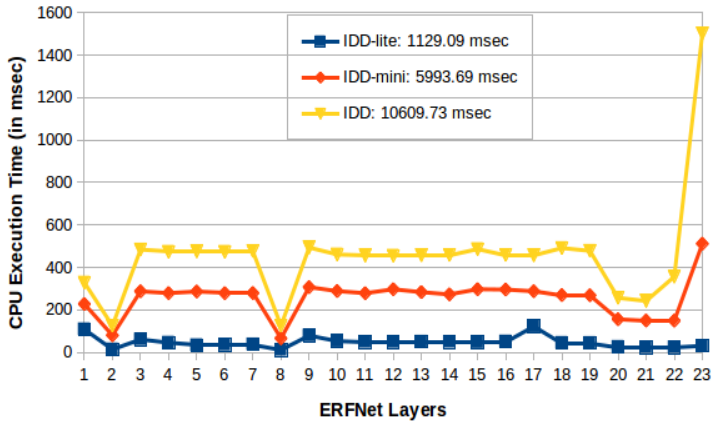


Figure 3.10 Runtime statistics per layer for ERFNet model on all three datasets (IDD, IDD-mini and IDD-lite). The total run time for each dataset is mentioned in the legend *(Best viewed when zoomed)*

can add more value to quick prototyping and help move toward real-time deployment of segmentation models.

3.7 Challenges: Semantic Segmentation in Resource-Constrained Regime

To introduce the community to the proposed datasets related to the Indian context and to motivate them to develop models that achieve better results in unstructured environments, we conducted multiple challenges at different venues

- *International Conference on Computer Vision* (ICCV) 2019: We conducted several challenges at *Workshop on Autonomous Navigation in Unconstrained Environments* (AutoNUE). These challenges were (i) Unconstrained Semantic Segmentation on IDD, (ii) Panoptic Segmentation, (iii) Segmentation on Constrained Devices, and (iv) Localization. The participants in the segmentation for constrained devices challenge were asked to run models with restricted runtime requirements and run the inference code of their models, in docker containers with restricted memory, CPUs, and runtime. The containers provided were using either Pytorch or Tensorflow in a constrained setting. They were given the instructions needed to generate the label outputs for getting the score. The output metric used was the standard *Intersection-Over-Union* (IoU) as discussed in Chapter 2 and the dataset used for the challenge was the proposed *IDD-lite*. Out of the total participants, a

team from Mapillary Research won the first prize achieving 61.84% mIoU on the test set, while the runner-up, DeepBlueAI achieved 60.72% mIoU on the test set. Table 3.6 shows the top-3 mIoU rankings of the challenge. The winners of the challenge used a Resnet50 [44] backbone modified to achieve $8\times$ downsampling along with 2048 output features. The segmentation head was the DeeplabV3 [20] model with modified dilation rates. The batch norm layers along with the ReLU activation function were replaced by Inplance-ABN [105]. The whole architecture was pre-trained on Mapillary Vistas dataset [92], discussed in Chapter 2. They also used max pooling and class uniform sampling to enforce class balance. The model was trained for 90 epochs using polynomial learning rate decay and L_2 weight decay.

- *National Conference on Computer Vision, Pattern Recognition, Image Processing, and Graphics* (NCVPRIPG) 2019: A challenge on resource-constrained semantic segmentation was also conducted at *NCVPRIPG* 2019 using the IDD-lite dataset proposed earlier. The winners of the competition used an encoder-decoder architecture [5]. The authors experimented with different versions of EfficientNet [118] and UNet [104] as decoders. EfficientB7 with UNet configuration gave the authors the best mIoU of 62.76 on the test set.

Venue	Team	mIoU (7 labels, 128p resolution)
AutoNUE, ICCV	Mapillary	61.85%
	DeepBlueAI	60.72%
	AKSM	59.11%
NCVPRIPG	USSB	62.76%
	Res-UNet	62.45%
	SSS	61.67%

Table 3.6 Shows the top-3 mIoU rankings at *ICCV 2019* and *NCVPRIPG 2019* resource constrained semantic segmentation challenge conducted on IDD-lite dataset.

3.8 IDD-mini and IDD-lite for Education

In addition to employing the IDD-lite to explore innovative architectures for the resource-constrained segmentation challenge, we have also envisioned the utilization of the proposed datasets for educational purposes. Despite significant growth in this field of vision, over the past few years, many universities and online platforms do not prioritize semantic segmentation as a fundamental concept for teaching vision-related topics, unlike the emphasis placed on classification settings. While platforms like Kaggle do host competitions that involve the application of semantic segmentation, the majority of students in developing countries, such as India, still encounter significant difficulties in grasping these concepts. In order to help the students understand the concepts, we have created some semantic segmentation notebooks which include the following contents.

- Notebook 1: Understanding Segmentation: From an older to a newer approach

In [1]:

```
#Importing the necessary modules

import cv2
import numpy as np
from matplotlib import pyplot as plt
```

In [3]:

```
# Load the input image.
img = cv2.imread('img/house.jpg')

# Convert image to grayscale
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# Display input image
plt.imshow(gray, 'gray')
plt.title('Input')
plt.show()

# Threshold using OpenCV, otsus thresholding.
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# Save the output image
cv2.imwrite("../Result/coin_thresh.jpg", thresh)

plt.imshow(thresh, 'gray')
plt.title('Output')
plt.show()
```

Figure 3.11 Sample screenshot of the first notebook to introduce the community to semantic segmentation. The notebook discusses traditional methods of performing segmentation, and datasets for deep learning-based segmentation, and concludes with a small exercise based on segmentation using thresholding. The link to the first notebook is given here [3].

- Notebook 2: Model Architectures for Semantic Segmentation
- Notebook 3: ERFNet Architecture for IDD Lite
- Notebook 4: Semi-Supervised Semantic Segmentation: A Use Case

The first notebook focuses on the basic understanding of segmentation as a problem, Gestalt principles, few traditional approaches to tackling the problem. It also gives a brief overview of datasets such as Cityscapes [25], IDD [121] and touches upon the loss functions used in deep learning mode. Finally, the notebook concludes with a thresholding-based exercise to segment out a particular object from an image. The notebook can comfortably run on a laptop CPU. Figure 3.11 provides a small screenshot of the notebook.

The second notebook explains different types of end-to-end architectures available for semantic segmentation as discussed in Chapter 2. The notebook for practical demonstration uses PSPNet [140] architecture (experimented in Pytorch) which is a non-encoder-decoder architecture to demonstrate the working of atrous spatial pyramidal pooling on a few examples. This notebook can run both on CPU and GPU but

[3]Source: `https://github.com/Ashutosh1995/Semseg-Notebooks/blob/master/SS-1.ipynb`

since the PSPNet model uses ASPP which is heavy in computation, GPU runtime would provide faster results. Figure 3.12 provides a screenshot of the notebook.

The third notebook dwells into the practical demonstration of ERFNet [103], an encoder-decoder architecture for real-time semantic segmentation trained and validated on the proposed IDD-lite dataset. The dataset and the model definition can be directly loaded onto the RAM thus allowing CPU computation to run effectively. Figure 3.13 provides the screenshot of the third notebook.

Finally, the fourth notebook goes a step beyond to discuss the idea of using semi-supervised semantic segmentation introduced in [58] to jointly train a model on datasets corresponding to different geographies. Since it uses multiple heavy high-resolution datasets such as Cityscapes [25], IDD [121] for coming up with the universal model, it does certainly require good access to GPU resources for training the models.

3.9 Summary

In this chapter, we proposed two small-scale datasets, IDD-mini and IDD-lite, to address the problem of segmentation in resource-constrained environments. We showed that these carefully designed datasets give decent qualitative and quantitative results enabling fast prototyping on low-resource hardware and hugely reducing the training and deployment costs. Our claim is that the proposed solution could be used for three main applications. This chapter discussed the use of these datasets as a part of teaching/workshop on semantic segmentation by introducing a couple of notebooks that can help the community in getting started with the same. In the next chapter, we will discuss the usefulness of the proposed solution in performing an architecture search by experimentally showing that the performance obtained using a smaller network on these datasets actually translate to a larger network with high-resolution images.

[5]Source: `https://github.com/Ashutosh1995/Semseg-Notebooks/blob/master/SS-2.ipynb`
[7]Source: `https://github.com/Ashutosh1995/Semseg-Notebooks/blob/master/SS-3.ipynb`
[9]Source: `https://github.com/Ashutosh1995/Semseg-Notebooks/blob/master/SS-4.ipynb`

In [1]:
```
#Necessary imports
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
```

In [2]:
```
#Importing the necessary functions from the files present directory

import caffe_pb2
from utils import conv2DBatchNormRelu, residualBlockPSP, pyramidPooling
from loss import multi_scale_cross_entropy2d
pspnet_specs = {"cityscapes": {"n_classes": 19, "input_size": (713, 713), "block_config": [3, 4, 23, 3]},}
```

In [3]:
```
class pspnet(nn.Module):

    def __init__(
        self, n_classes=21, block_config=[3, 4, 23, 3], input_size=(473, 473), version=None):

        super(pspnet, self).__init__()

        self.block_config = (
            pspnet_specs[version]["block_config"] if version is not None else block_config           #1
        )
        self.n_classes = pspnet_specs[version]["n_classes"] if version is not None else n_classes    #2
        self.input_size = pspnet_specs[version]["input_size"] if version is not None else input_size #3

        #The above set of lines extract the value items in the pspnet_specs mentioned above
        #BlOCK_CONFIG: represents the number of blocks in the resnet layer, please consult the residualbloc


        ############################################################################################

        #THESE SET OF LAYERS DEFINE THE RESET BLOCKS WITH DIALATION WHICH PRODUCES THE INITIAL FEATURE MAP


        # Encoder
        self.convbnrelu1_1 = conv2DBatchNormRelu(
            in_channels=3, k_size=3, n_filters=64, padding=1, stride=2, bias=False
        )
        self.convbnrelu1_2 = conv2DBatchNormRelu(
            in_channels=64, k_size=3, n_filters=64, padding=1, stride=1, bias=False
        )
        self.convbnrelu1_3 = conv2DBatchNormRelu(
            in_channels=64, k_size=3, n_filters=128, padding=1, stride=1, bias=False
        )

        # Vanilla Residual Blocks
        self.res_block2 = residualBlockPSP(self.block_config[0], 128, 64, 256, 1, 1)
        self.res_block3 = residualBlockPSP(self.block_config[1], 256, 128, 512, 2, 1)
```

Figure 3.12 Sample screenshot of the second notebook to introduce the community with semantic segmentation. The notebook discusses different types of architectures available for performing semantic segmentation. The experimentation is based on the PSPNet architecture. The link to the first notebook is given here [5].

In [1]:
```
#Importing the necessary libraries which can be used !!

import random
import time
import numpy as np
import torch
import math
```

In [2]:
```
#Importing library to do image related operations
from PIL import Image, ImageOps
```

In [3]:
```
#Importing the important functionalities of Pytorch such as the dataloader, transform's
#and optimizer related functions.

from torch.utils.data import DataLoader
from torchvision.transforms import  Resize
from torch.optim import SGD, Adam, lr_scheduler
from torchvision.transforms import ToTensor, ToPILImage
```

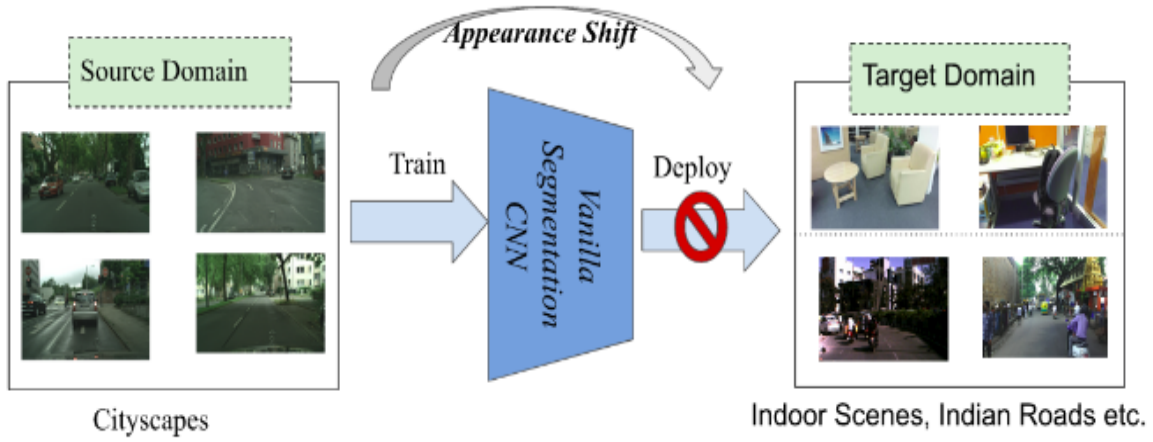
In [4]:
```
# Importing the dataset class for IDD_Lite
from dataset import idd_lite

#Importing the Relabel, ToLabel and Colorize class from transform.py file
from transform import Relabel, ToLabel, Colorize
import matplotlib
from matplotlib import pyplot as plt
%matplotlib inline
```

In [5]:
```
import importlib
from iouEval import iouEval, getColorEntry #importing iouEval class from the iouEval.py file
from shutil import copyfile
```

A few global parameters

In [6]:
```
NUM_CHANNELS = 3 #RGB Images
NUM_CLASSES = 8 #IDD Lite has 8 labels or Level1 hierarchy of labels
USE_CUDA = torch.cuda.is_available()
IMAGE_HEIGHT = 160
DATA_ROOT = '/ssd_scratch/cvit/amishra/idd1_lite'
```

Figure 3.13 Sample screenshot of the third notebook to introduce the community with semantic segmentation. The notebook discusses in detail the ERFNet network, an encoder-decoder architecture for real-time semantic segmentation using the proposed IDD-lite dataset. The notebook can be easily run from a laptop CPU as the dataset size along with the model definition is a few hundred of MB. The link to the third notebook is present here[7].



Source of the image: Slides

They introduce the concept of a single model that can be deployed in any domain for the task of semantic segmentation without affected by the **Domain shift** and **Appearance shift**. As can be seen from the figure, if we train a model on autonomous navigation based dataset like Cityscapes and deploy the trained model in a similar setting or in a total differnt domain, there is a loss of accuracy due to shift in the appearance of the objects. To address domain shift problem, they minimize supervised as well as within and cross-domain unsupervised losses, introducing a novel feature alignment objective basedon pixel-aware entropy regularization.

We are going to start the notebook and also discuss the concepts in between code snippets.

In this notebook, we are going to used two pixel level annotated datasets for autonomous navigation namely **[Cityscapes](https://www.cityscapes-dataset.com/)** and **[IDD](https://idd.insaan.iiit.ac.in/)**. They can be downloaded from these links. We will need atleast one gpu for running this experiment.

These two datasets are totally different in terms of geographical location, weather, traffic participants etc. but for them there label spaces overlap.

In [1]:
```
#First a bit of basic library imports

import os
import pdb
import time
import torch,sys
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
from torch.optim import SGD, Adam, lr_scheduler
from torch.autograd import Variable
from torch.utils.data import DataLoader, ConcatDataset
import torchvision
import torch.nn.functional as F
```

Figure 3.14 Sample screenshot of the fourth notebook to introduce the community with semantic segmentation. This notebook introduces a method of performing semantic segmentation in a semi-supervised fashion. It introduces [58] to jointly train a model based on datasets belonging to different geographies to obtain a universal model that can be deployed in any city. The link to the third notebook is present here[9].

Chapter *4*

Neural Architecture Search for Resource Constrained Semantic Segmentation

In this chapter, we will discuss our effort in developing efficient models for semantic segmentation using the proposed datasets (introduced in Chapter 3) that also scale the performance to bigger datasets such as IDD [121] or Cityscapes [25] using *Neural Architecture Search*, a design methodology that has been on the rise from the last few years in automating the architecture design of neural networks. *Neural Architecture Search*, abbreviated as NAS, is an automated method used to discover optimal neural network architectures for specific tasks. It is a process where computational algorithms explore a vast search space of potential architectures to find the best-performing ones. NAS aims to reduce the manual effort and expertise required to design neural networks and can lead to improved performance compared to human-designed architectures. To better understand NAS, let's compare it with some examples of human-designed architectures:

- *Human-Designed Convolutional Neural Networks (CNNs)*: Classic CNN architectures like LeNet [64], AlexNet [61], VGG [111], and ResNet [44] were manually designed by human experts. These architectures were created based on intuition, empirical insights, and domain knowledge. Researchers made decisions about the number of layers, kernel sizes, and overall architecture based on their understanding of the problem and previous experiences. While these human-designed networks have been highly successful, they may not be the most efficient or optimal choices for every task or dataset.
- *Handcrafted Recurrent Neural Networks (RNNs)*: Recurrent Neural Networks (RNNs) [49], LSTM (Long Short-Term Memory) [50] and GRU (Gated Recurrent Unit) [23] were manually crafted and designed to handle sequential data. These architectures were designed with specific gating mechanisms to capture long-range dependencies. The design choices were guided by the nature of sequential data and challenges in gradient propagation. However, there remains a possibility of more effective architectures for particular sequential tasks that human experts may not have considered.

- *Predefined Network Blocks*: In transfer learning, researchers often use predefined network blocks like Inception modules or ResNet blocks to build larger architectures for specific tasks. These blocks are designed based on heuristics and prior knowledge about efficient feature extraction and representation learning. While they work well for transfer learning and fine-tuning, they might not be the most optimal choice when starting from scratch or tackling novel problems.

Comparatively, Neural Architecture Search automates the process of finding architectures and goes beyond human intuition and heuristics. NAS methods employ techniques such as reinforcement learning, evolutionary algorithms, and Bayesian optimization to search through a vast space of possible architectures, considering different layer types, connectivity patterns, and hyperparameters. By automating the design process, NAS can discover novel and efficient architectures that outperform or match human-designed ones on various tasks. The advantage of NAS lies in its ability to explore a more extensive range of architectures and optimize them for specific tasks, potentially leading to models with superior performance, reduced computational cost, and improved generalization compared to architectures designed manually by human experts.

4.1 Introduction

In the field of computer vision, the optimization of deep neural networks for semantic segmentation tasks has become increasingly crucial for achieving state-of-the-art results. To accomplish this task effectively, deep neural networks with intricate architectures are often employed. However, navigating through the hyper-parameter space to come up with the most optimum configuration and architecture for semantic segmentation is a demanding task due to the huge training costs involved with deep neural networks. In the context of classification, previous works [73] perform an architectural search on CIFAR dataset [60] to show that the best performance also applies to larger scale datasets like ImageNet [61]. Several works have tried the application of reinforcement learning [15, 142], evolutionary algorithm [73] in the same context. However, there have been fewer works [19, 72] to conduct an architectural search on dense segmentation tasks due to the resource intensiveness of the task. SqueezeNAS [109] tries to search for efficient semantic segmentation architecture in a hardware-constrained budget but takes 15 GPU days to converge and obtain the desired objective. Therefore, smaller datasets that enable quick prototyping for hyperparameter search, and help in replicating the results on larger datasets are essential. This would bring down the cost of training and would aid in improving overall performance.

Hence the contribution of this chapter can be summarized as follows:

- We introduce and discuss various aspects in detail related to *Neural Architecture Search* (See Section 4.2).

Figure 4.1 The broad sub-divisions of NAS [28]. An architecture *A* is selected by a search strategy from a defined search space *SS*. To evaluate the architecture, i.e. how good/bad it performs on a particular task, the architecture then goes into the performance estimation procedure, which returns the score/accuracy.

- We dive into the applicability of NAS for computer vision tasks such as image classification, and semantic segmentation. Additionally, we also discuss the role of optimization algorithms that aid in architecture search. (See Section 4.3 and Section 4.4)
- We establish that the accuracy of various models trained on our datasets correlates well with the accuracy on large-scale datasets, especially in cross-domain settings. This allows for fast prototyping and architectural search for semantic segmentation algorithms (See Section 4.5).

4.2 Neural Architecture Search: An Overview

Deep learning has achieved success in recent years because instead of extracting traditional hand-crafted features, it provided a way in which the network itself decides the need for down-weighting or up-weighting activations with the help of weights that subsequently helps in solving the task at hand. Though from one perspective, the feature engineering in the case of CNN has been very successful since these features are generic and also help in transferring the knowledge from one task to other [134], there has been an enormous amount of increase in the area of architectural engineering, leading to more complex network design patterns. *Neural Architecture Search* or NAS (in short form), is the process of designing complex architectures automatically, in the whole process of automating machine learning paradigm. NAS can be understood as a subset of AutoML intersecting with hyperparameter optimization (which we are going to discuss in the upcoming section). This section is inspired by the survey on NAS by [28]. The working of NAS can be broadly divided into three categories: search space, search strategy, and performance estimation strategy. These subcategories of NAS are described below:

- ***Search Space***: Search space is defined as the domain of all possible architectures that can be outlined. One possible way of defining the architectures in space is to incorporate prior information about human-designed architectures. This can help us reduce the vast space of all available architectures and ease the search process. By leveraging the knowledge gained from successful architectures in the past, we can guide the search toward more promising areas of the space. However, there are some drawbacks to relying heavily on prior knowledge in the search space. One significant concern is the potential introduction of human bias into the search process. When we

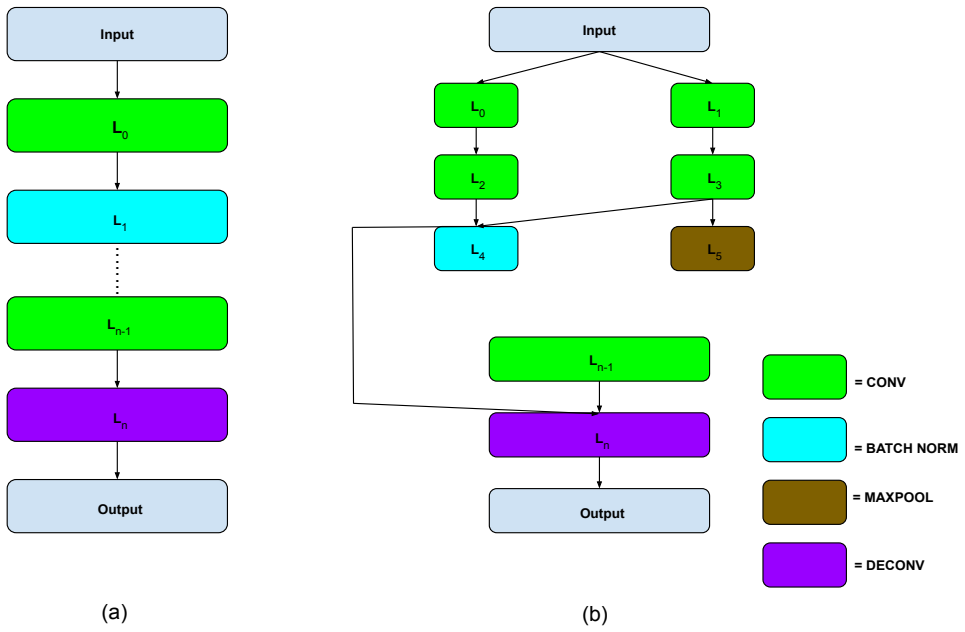


Figure 4.2 An example of few search spaces [28]. (a) Simple stacking of different layers (represented by the color) to form a network, (b) A more complex chain of network design consisting of concatenation, diverse branches, residual block, etc.

restrict the search to architectures that are similar to human-designed ones, it limits the exploration of more unconventional or novel architectures that might yield superior performance. This can lead to a repetitive style of architecture, where the algorithm may converge to a sub-optimal solution that closely resembles existing well-known architectures. Moreover, relying only on prior knowledge can hinder the discovery of unique designs that push the boundaries of what is currently known to work well. NAS is an opportunity to explore the full landscape of possible architectures, and overly constrained search spaces could prevent us from uncovering hidden gems that lie beyond the realm of conventional wisdom. To address these issues, striking a balance between prior knowledge and unconstrained exploration is crucial. Hybrid approaches that combine human-designed architectures with novel architectural components or operations can allow the algorithm to leverage the strengths of both approaches. For instance, cell-based search spaces incorporate predefined cells representing common building blocks, but they also allow the algorithm to learn how to combine and arrange these cells to create novel architectures. An example of some of the basic search spaces is depicted in Figure 4.2.

- ***Search Strategy***: Search strategy describes the method used to instantiate architectures by navigating the search space, which is often large or even unbounded. The goal of the search strategy is to efficiently explore the space to find architectures that are not underperforming while also identifying high-performing architectures in the minimum time span possible. The choice of search

strategy significantly impacts the effectiveness and computational efficiency of Neural Architecture Search (NAS). Some examples of commonly used search strategies commonly are:

1. *Bayesian Optimization-Based Approaches:* Bayesian optimization is a probabilistic model-based optimization technique that balances exploration and exploitation in the search process. It constructs a surrogate model of the objective function (e.g., validation accuracy) and employs an acquisition function to select the most promising architectures for evaluation. Bayesian optimization has been widely used in NAS due to its ability to efficiently explore the search space and converge to promising architectures. For instance, [9] defines search optimization using a generative model based on expected movement criteria by searching parameters through random search technique. [10] stated about the practicality of Bayesian optimization specifically for image classification tasks when tuning hundreds of parameters using random search technique. [114] defines a novel kernel technique with conditional parameter spaces that explicitly includes information about which parameters are relevant in a given model structure using Bayesian optimization. They show this novel kernel, combined with the power of Bayesian optimization obtains better results over several simpler baseline kernels.

2. *Reinforcement Learning-Based Approaches:* Reinforcement learning (RL) is a popular approach in NAS, where the search process is modeled as a sequential decision-making problem. The NAS algorithm acts as an agent that generates and evaluates architectures, and it receives rewards based on the performance of the resulting networks. By optimizing its policy through trial and error, the agent learns to generate architectures that lead to higher rewards. RL-based NAS methods have shown promising results in discovering high-performing architectures. [6] defines a modeling algorithm named MetaQNN that generates high-performing CNN architectures for specific learning tasks using Q-learning, ϵ-greedy exploration strategy and experience relay. In [144], the authors use a recurrent neural network in order to obtain a set of unique architecture definitions and train the RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. Similarly, in [145], the authors learn novel architectures on smaller datasets using reinforcement learning for image classification tasks and then show the transfer-ability onto bigger datasets.

3. *Neuro-Evolutionary-Based Approaches:* Neuro-evolutionary algorithms, inspired by the principles of natural selection, evolve populations of architectures over multiple generations. The search process involves genetic operators like mutation and crossover, which drive the evolution of architectures. Neuro-evolutionary-based NAS methods can handle both discrete and continuous search spaces and have the advantage of being computationally efficient. Work like [3] introduces GNARL, an evolutionary program that acquires both the structure and weights for recurrent networks simultaneously. The algorithm's empirical

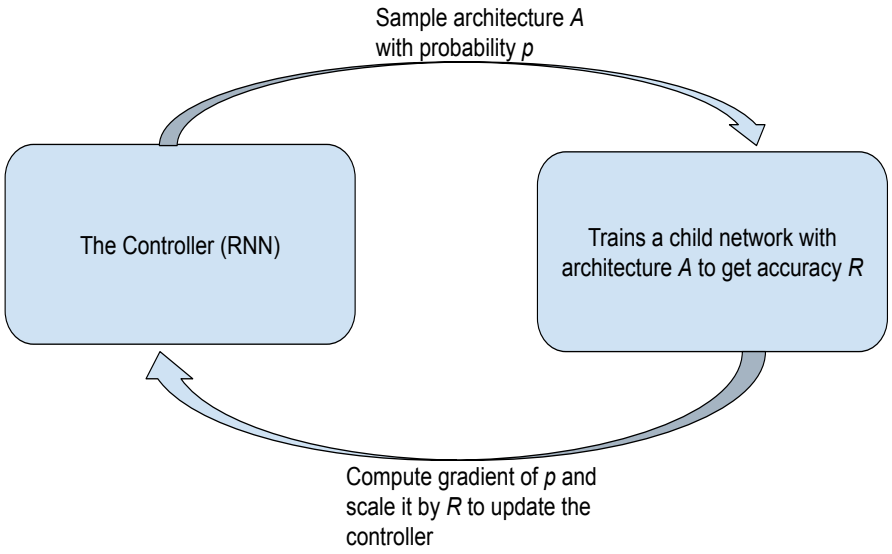


Figure 4.3 Depiction of the NAS procedure presented in [144] in which the controller emits an architecture definition which is trained and validated to return the accuracy. The parameters of the controller are then updated on the basis of the accuracy obtained.

acquisition approach enables the emergence of complex behaviors and topologies, which may not be attainable using conventional network induction methods due to artificial architectural constraints. [32] gives an overview of the most prominent methods for evolving artificial neural networks with a special focus on the advances in the synthesis of learning architectures.

4. *Hybrid Approaches:* Some NAS methods combine multiple search strategies to leverage their individual strengths. For example, some approaches use gradient-based methods in conjunction with evolutionary techniques to efficiently explore continuous search spaces while maintaining strong performance. Examples of hybrid approaches include *Efficient Neural Architecture Search* (ENAS) [29] and *Genetic CNN* [127].

Choosing the appropriate search strategy depends on the characteristics of the search space and the computational resources available. An effective search strategy allows NAS algorithms to discover high-performing architectures efficiently, accelerating the advancement of deep learning models across various tasks and domains.

- ***Performance Estimation***: The search mechanism in NAS can return multiple architectures that fulfill the criteria of generating novel networks from the search space. However, evaluating these

architectures is a critical step in the NAS process. Performance estimation involves assessing the capabilities of the candidate architectures using various methods. One straightforward approach is to split the data into training, validation, and test sets and evaluate the performance of each architecture on the validation set. While this method provides reliable estimates, it becomes computationally heavy when the number of architectures to be explored becomes substantial. To address the computational burden of performance estimation, recent research has focused on developing techniques that minimize run-time requirements. Some estimation strategies include curve extrapolation of various hyper-parameters or loss curves. These methods aim to predict the performance of architectures based on a subset of evaluations [7, 27]. By intelligently extrapolating performance trends, they reduce the number of costly evaluations required for the full search space. Another approach to efficient performance estimation involves weight transfer methods. In these techniques, knowledge gained from training similar models is transferred to the network in consideration. Instead of training each architecture from scratch, the weights of previously trained models with similar characteristics are used as a starting point for the new architecture. This approach significantly reduces the computational cost of evaluating different architectures [29, 98].

4.3 Neural Architecture Search for Vision tasks

There has been a lot of work in the classification setting for finding novel architectures through the use of NAS algorithms. Inspired by the same, there have been a few works in the domain of semantic segmentation as well. Here we are going to discuss a few methods that have gained recognition in the community.

- *Image Classification:* The earliest works in Neural Architecture Search (NAS) for image classification, [144] employ a recurrent neural network (RNN) as a controller to generate a description of a neural network architecture in the form of a string. This description includes details such as the number of input/output channels, kernel sizes, strides, and other architectural parameters. The controller emits the description, also known as the child network, which is then trained and evaluated on the validation set. Reinforcement learning is used to update the parameters of the controller based on the performance of the child model. Although this approach leads to an increase in classification accuracy, it requires exploring a large space of possible architectures, resulting in a computationally intensive process. The approach generates around 12,800 architectures and requires approximately 22,000 GPU days for training. The whole procedure trains on the CIFAR10 dataset [60]. Figure 4.3 illustrates the framework used in this approach. To address the resource-intensive nature of NAS, subsequent works propose searching for repetitive blocks or cells instead of exploring entire architectures, as suggested by [145]. These repetitive blocks, such as normal cells (that maintain the dimensionality) and reduction cells (that reduce spatial dimensions), are stacked together to form the complete architecture. By searching for these blocks, the search space significantly reduces, and the best-performing blocks could easily be transferred to other

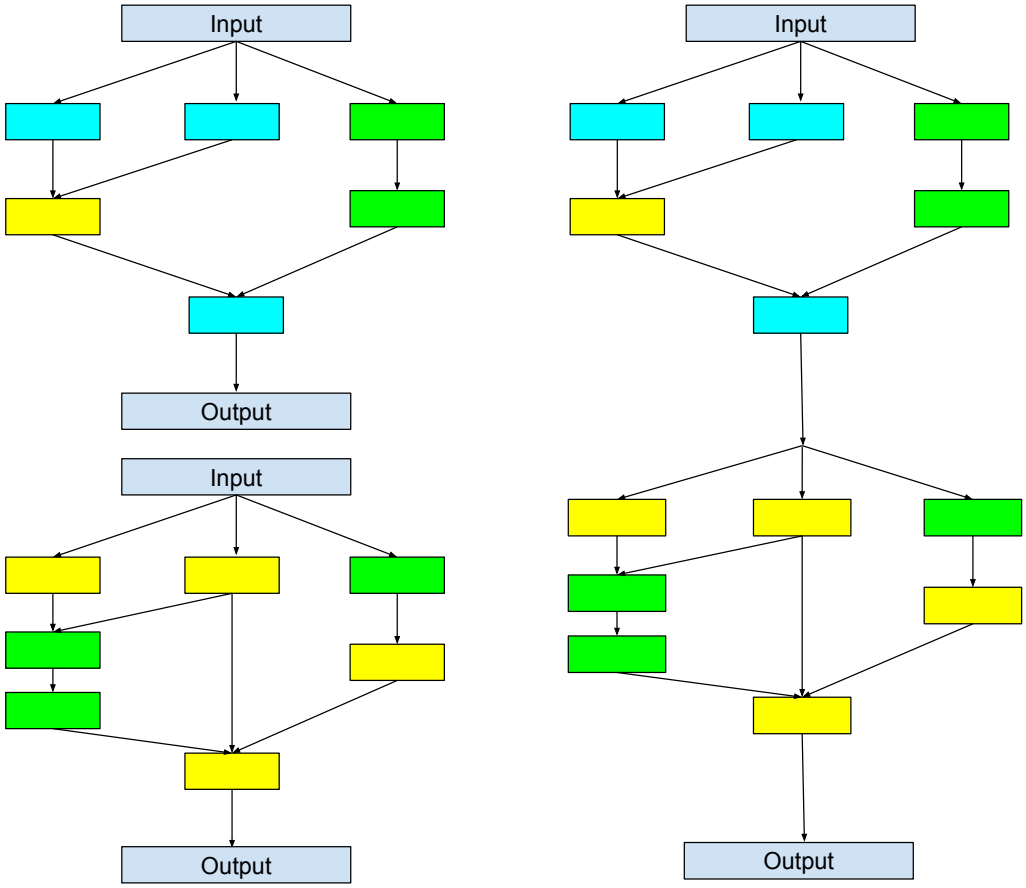


Figure 4.4 A normal and a reduction cell search space [28] used in [145]. On the left top is the normal cell, and on the bottom is the reduction cell. On the right is the architecture formed by stacking these two types of cells. Different colors correspond to different types of layers involved in the overall processing of inputs.

datasets by adapting the number of cells used within a model. Figure 4.4 illustrates the normal and reduction cells and their incorporation into the network architecture. In a similar approach, [73] defines repetitive cells used within a predefined evolutionary search algorithm that underwent mutations based on their fitness scores. To expedite the search process, other sets of works focus on speeding up the search by defining the model as a sub-graph of a single directed acyclic graph. Before the search process begins, all possible sets of networks are initialized. The algorithm then decides whether to choose a single path with the help of reinforcement learning [98] or use a weighted relaxation of the edges [74]. While these approaches accelerate the search process, they limit the exploration of architectures since the paths are pre-defined at the beginning.

- *Semantic Segmentation:* Proposed NAS approaches for semantic segmentation have been influenced by the methods discussed earlier. For instance, [19] proposes a method to search for a single cell on top of a fully convolutional network. By focusing on a cell-based search space, the authors

aim to discover efficient and effective architectures for semantic segmentation tasks. Similarly, [90] introduces a NAS approach to search for an efficient decoder within the encoder-decoder framework. This technique attempts to optimize the decoder part of the network to enhance segmentation performance. However, both of these methods heavily rely on existing image classifiers, which constrain the description of the search space. The efficiency of the decoder is thus determined by the classifier network used, limiting the exploration of more diverse and innovative architectures. To address these limitations, [91] propose a remedy to their previous work [90]. They allow for the discovery of tiny networks by removing expensive techniques such as knowledge distillation. This enhancement enabled them to explore more lightweight and efficient architectures for semantic segmentation. Moreover, their method does not require a full encoder definition, and instead, they discover the model definition using only 60K pre-trained parameters. This reduction in complexity makes the NAS process more efficient and accessible for discovering high-performing architectures for semantic segmentation. Another notable approach by [72] adapts the relaxation of path edges for semantic segmentation. This method demonstrates good qualitative and quantitative results in segmentation tasks. However, similar to other approaches mentioned earlier, it also has a limited set of architectural definitions to explore. Despite achieving promising results, the constrained search space might hinder the exploration of more diverse and innovative architectures that could potentially outperform the existing ones.

4.4 Hyper-parameter Optimization Algorithms for Architecture Search

Neural Architecture Search (NAS) can be considered a subset of hyper-parameter tuning, and both processes aim to optimize the performance of deep neural networks. NAS focuses on discovering efficient cells or blocks that can be stacked together to create novel architectures, while hyper-parameter tuning involves finding the optimal set of hyper-parameters that yield better scores or accuracy on the target task. In NAS, the search space includes architectural choices such as the number of layers, types of layers, and connectivity patterns. The goal is to explore this space to find architectures that are not only different from human-designed networks but also perform well on the given task. NAS usually involves a more exhaustive and computationally intensive search process due to the vast number of possible architectures. On the other hand, hyper-parameter tuning takes place within the context of a fixed network architecture. The search space in hyper-parameter tuning includes hyper-parameters like learning rates, batch sizes, weight decay, and other training-related settings. Unlike NAS, hyper-parameter tuning generally occurs before the training procedure starts, and it does not involve changing the network's structural components. The primary objective of hyper-parameter tuning is to find the hyper-parameter configuration that maximizes the model's performance on the validation set or minimizes the training loss. Furthermore, the idea of optimizing the optimization algorithm itself as a hyper-parameter can be seen as a meta-optimization approach. In this context, the optimization algorithm's hyperparameters (e.g., learning rate schedules, momentum, etc.) are treated as parameters that can be tuned during the training

process. Meta-optimization aims to find the best combination of optimization hyper-parameters that lead to faster convergence or better generalization on the target task. This approach is still an active area of research and holds promise for further improving the efficiency and effectiveness of training deep neural networks. In other words, hyperparameter optimization can be described by the following equation.

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, f(\theta) \tag{4.1}$$

In this equation, $f(\theta)$ is the function of parameters that we wish to either minimize or maximize. θ^* are the set of hyper- parameters which yields best results on the val set. We are now going to discuss different hyperparameter tuning algorithms.

4.4.1 Grid Search

The approach described here is a simple and straightforward method for hyper-parameter optimization, often referred to as *grid search*. In grid search, each hyper-parameter is discretized, i.e. it is divided into a set of predefined values or intervals. The search space is then defined as the cartesian product of all possible combinations of these discretized values, resulting in a grid-like structure. For example, if there are two hyper-parameters, each with three possible values, the grid search would explore nine different combinations 3×3 of the hyper-parameters. Once the search space is defined, the network is trained for each combination of hyper-parameters in the grid. The network's performance is evaluated using a validation set or through cross-validation, and the set of hyper-parameters that yields the best performance metric (e.g., highest accuracy or lowest loss) on the validation data is selected as the final configuration. Grid search can be parallelized in case of access to multiple computing resources or a distributed computing environment. This means that instead of training each combination sequentially, multiple combinations can be trained simultaneously, significantly reducing the overall search time. Parallelizing grid search is advantageous as it can speed up the optimization process, especially when exploring large search spaces with numerous hyper-parameters. However, one significant drawback of grid search is that it can be computationally expensive and time-consuming, particularly when dealing with a model that has a large number of hyper-parameters or when the range of values for each hyper-parameter is extensive. The search space grows exponentially with the number of hyper-parameters, making it impractical for models with hundreds or thousands of hyper-parameters. As a result, grid search may not be feasible for complex models or tasks with a high-dimensional hyperparameter space.

4.4.2 Random Search

Random search is an alternative hyper-parameter optimization algorithm that differs from grid search in its sampling approach. Instead of systematically exploring all combinations of hyper-parameters, random search randomly samples values from the defined search space for each hyper-parameter. The number of trials, or random samples, is predefined as a parameter of the algorithm. Unlike grid search,

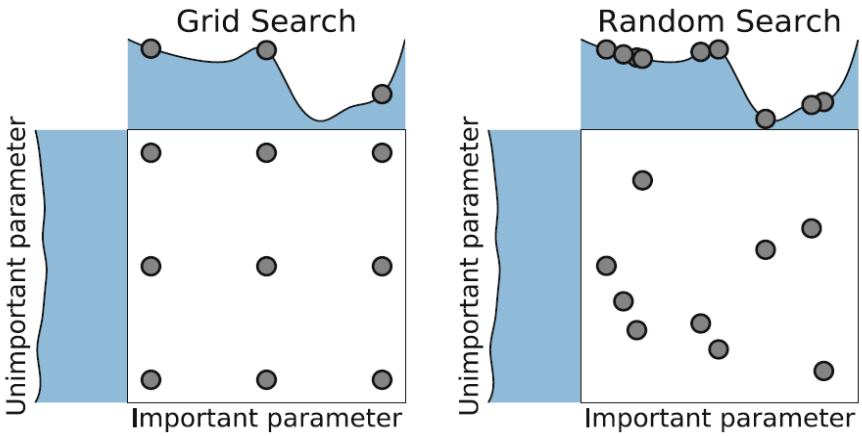


Figure 4.5 *Grid Search* versus *Random Search* [10, 95]. *Grid search* can explore few distinct values compared to *Random Search*.

random search does not impose a specific structure on the search space, which allows it to explore a more diverse set of hyperparameter configurations. One of the primary advantages of random search is that it is not constrained by time limits. In grid search, the total number of combinations to be explored can be substantial, leading to long optimization times. In contrast, random search allows to define the number of trials based on their available computational resources and time constraints. This flexibility makes the random search a practical choice for optimization tasks with limited computational budgets or when there is no clear indication of how long the optimization process should take. Another advantage of random search is its ability to handle correlated hyper-parameters. In some cases, certain hyper-parameters may interact with each other or have dependencies, and their values may affect the model's performance together. In grid search, these dependencies may not be adequately explored if they do not align with the predefined combinations. However, random search, by virtue of its randomness, can discover such correlations more effectively. When random samples are drawn, they might inadvertently cover regions of the search space where correlated hyper-parameters have an impact on the model's performance. As a result, random search can help find the most optimum values for both correlated hyper-parameters. However, one drawback of random search is that it may require more trials (random samples) compared to grid search to find the best hyper-parameter configuration, especially if the search space is large or has complex interactions between hyper-parameters. The random search may also lead to sub-optimal or redundant samples, which can be inefficient in terms of resource utilization. Random Search is also fast to run and parallelize but has a similar disadvantage to grid search. If the model is heavy to run, then it might take a longer time to converge to the best set of hyper-parameters.

4.4.3 Bayesian Optimization

Bayesian optimization methodology basically keeps a record of the past hyper-parameters that are evaluated on the validation set. It uses a probabilistic approach to map the hyper-parameters to a

probability value of the score of the objective function. The equation of Bayesian optimization can be described as follows:

$$P(score|hyper - parameters) \tag{4.2}$$

The algorithm works according to the following steps:

- Define a set of hyper-parameters to define the search space.
- Formulate an objective function that takes a set of hyper-parameters as input and returns a performance score (e.g., accuracy or loss) based on the model's performance with those hyper-parameters. The optimization can aim to minimize or maximize this objective function, depending on the specific task.
- Construct a probabilistic model (often a Gaussian process), also known as a *surrogate* of the objective function. This surrogate model approximates the behavior of the objective function across the hyper-parameter space, allowing for efficient exploration and uncertainty modeling.
- Employ an *acquisition function* to select the next set of hyper-parameters for evaluation. The acquisition function quantifies the potential improvement that can be achieved by evaluating specific hyper-parameters based on the surrogate model's uncertainty. It balances exploration (sampling regions of uncertainty) and exploitation (focusing on promising regions).
- Maintain a mapping of past evaluations of hyper-parameters and their corresponding performance scores. This mapping is crucial for updating the surrogate model and incorporating new information during the optimization process.
- Iteratively improve the surrogate model as new hyper-parameter configurations are evaluated. The optimization continues until a stopping criterion is met, such as reaching a specified number of iterations or utilizing the available computation budget effectively.

4.4.3.1 Acquisition Function

The *acquisition function* plays a crucial role in guiding the Bayesian optimization process by selecting the next set of hyper-parameters to evaluate. One of the most commonly used acquisition functions is the *Expected Improvement* (EI). The EI measures the potential improvement in the objective function's value compared to the current best-known value (either maximum or minimum, depending on the optimization goal).

When trying to find the minimum of the objective function, the EI can be computed using the following equation:

$$f_{min}(x) = \max(0, y_{min} - y_{\text{lowest expected}}) \tag{4.3}$$

Here, y_{min} represents the minimum value observed so far in the optimization process, and $y_{\text{lowest expected}}$ is the lowest possible value that could be obtained from the confidence range of each possible x value (i.e., the uncertainty associated with the surrogate model's predictions).

On the other hand, if the optimization goal is to find the maximum value, the equation can be modified accordingly to compute the *Expected Improvement* for maximization:

$$f_{max}(x) = \max(0, y_{\text{highest expected}} - y_{max}) \tag{4.4}$$

In this case, $y_{\text{highest expected}}$ represents the highest possible value within the confidence range of each x value, and y_{max} is the maximum value observed so far. By evaluating the acquisition function for different hyper-parameter configurations, Bayesian optimization can prioritize the most promising configurations to explore in the next iteration. The EI guides the optimization toward regions where improvements are likely to be significant, balancing the exploration of uncertain regions and exploitation of potentially high-performing configurations. The use of the EI acquisition function contributes to the effectiveness and efficiency of Bayesian optimization in finding optimal hyper-parameter configurations in a sample-efficient manner.

4.4.4 Tree Parzen Estimator

Tree-structured Parzen Estimators (TPE) is a popular algorithm used for handling categorical hyper-parameters in a tree fashion. It differs from the Bayesian approach in the way it models the probabilistic function or the surrogate. While Bayesian optimization models the posterior probability $P(Y|x)$, where *Y* is the score on the surrogate and *x* is the hyper-parameters, TPE takes a different approach. It applies Bayes' rule to model both $P(x|Y)$ and $P(Y)$. This allows TPE to explore the hyper-parameter space differently and can be advantageous for handling categorical or discrete hyper-parameters. The TPE algorithm requires collecting some data in the initial iterations before it can be effectively applied. A common approach is to use a random search for these initial iterations, which helps in gathering diverse observations from the search space. Once the data is collected, it is divided into two sets of groups. The initial group contains observations that yielded the best scores after evaluation (i.e., $y < y$), while the second group contains all the remaining observations (i.e., $y >= y$). The value of y^* represents a threshold, and it is defined as a parameter for the TPE algorithm. The main objective of TPE is to find hyper-parameters that are more likely to exist in the first group than in the second. To achieve this, TPE computes the likelihood probability for both groups. Using the likelihood probabilities, a bunch of candidate values are sampled from the first group. Amongst these sampled candidates, TPE aims to find the single candidate that is more probable to belong to the first group and less likely in the second group. The Expected Improvement (EI) formula is utilized to make this determination, as described in the equation:

$$\text{Expected Improvement}(x) = \frac{l(x)}{g(x)} \tag{4.5}$$

Here, *l(x)* is the likelihood probability of the first group (observations with $y < y$), and *g(x)* is the likelihood probability of the second group (observations with $y >= y$). The next best hyperparameter point for the algorithm is the one that maximizes $l(x)/g(x)$.

We have discussed neural architecture search and the various components associated in detail, including search spaces, search strategies, and performance estimation methods. Neural architecture search (NAS) as discussed before, has emerged as a powerful technique for automating the design of deep learning models, alleviating the burden of manual architecture engineering. In the next section, we will explore the application of neural architecture search in a resource-constrained environment, where computational resources are limited. We will examine how NAS can be adapted to efficiently explore the search space and discover optimal architectures under resource constraints. Furthermore, we will demonstrate the effectiveness of NAS on proposed datasets explained in Chapter 3, highlighting its potential in optimizing model performance and how the performance obtained correlates to bigger datasets.

4.5 Experiments and Results

In this section, we demonstrate how IDD-lite and IDD-mini datasets can be useful for architecture searches with limited resources. Neural Architecture Search [98] (NAS) on tasks like dense semantic segmentation need thousands of iterations GPU-days for convergence. Architecture search can be computationally very intensive as each evaluation typically requires training a neural network. Therefore, it is common to restrict the search space to reduce complexity and increase the efficiency of architecture search. More recent papers on architecture search have shifted to searching the repeatable cell structure while keeping the outer network level structure fixed by hand. This strategy needs fewer GPU hours and can conduct large experiments in a constrained time. This would be a major advantage in resource-constrained environments. Currently, this strategy is limited to tasks like image classification, where small-scale datasets are available, but not for semantic segmentation due to the lack of such small-scale datasets. We propose that IDD-mini and IDD-lite can be used to do architecture search in a resource-efficient way for semantic segmentation task.

We consider a scenario where we need to come up with the best architecture with a limited resource budget. We even explicitly consider a scenario where we can only afford a fixed set of parameters for the architecture. Now, we need to find the best-performing architecture among them. We conduct two experiments and show that architecture search results conducted on IDD-lite with a custom ERFNet model actually translate to a different domain like Cityscapes [25]. Thus, we are also exploring the generalizability of architectural search through such experiments.

4.5.1 Identifying Optimal Cell Structure

Our aim is to find the best architecture in a custom-designed architectural space using IDD-lite. Also, we show that the results correlate to Cityscapes dataset [25]. We use ERFNet [103] outer network level structure as the basis for our model. Within this structure, we replace the non-bottleneck layer proposed in the original ERFNet with a custom structure with an architecture skeleton(Conv-module) as proposed in [72] shown in Figure 4.6. Each of the blocks in this Conv-module(skeleton) is filled from a set of 1 atrous 3×3 layer, 3 separable 5×5 layers, 2 atrous 5×5 layers, 4 separable 3×3 layers, to ensure that we have same number of parameters overall. This forms the search space for architecture search. We use architectures given in Table 4.1 to conduct experiments on the Cityscapes dataset [25] and IDD-lite, respectively.

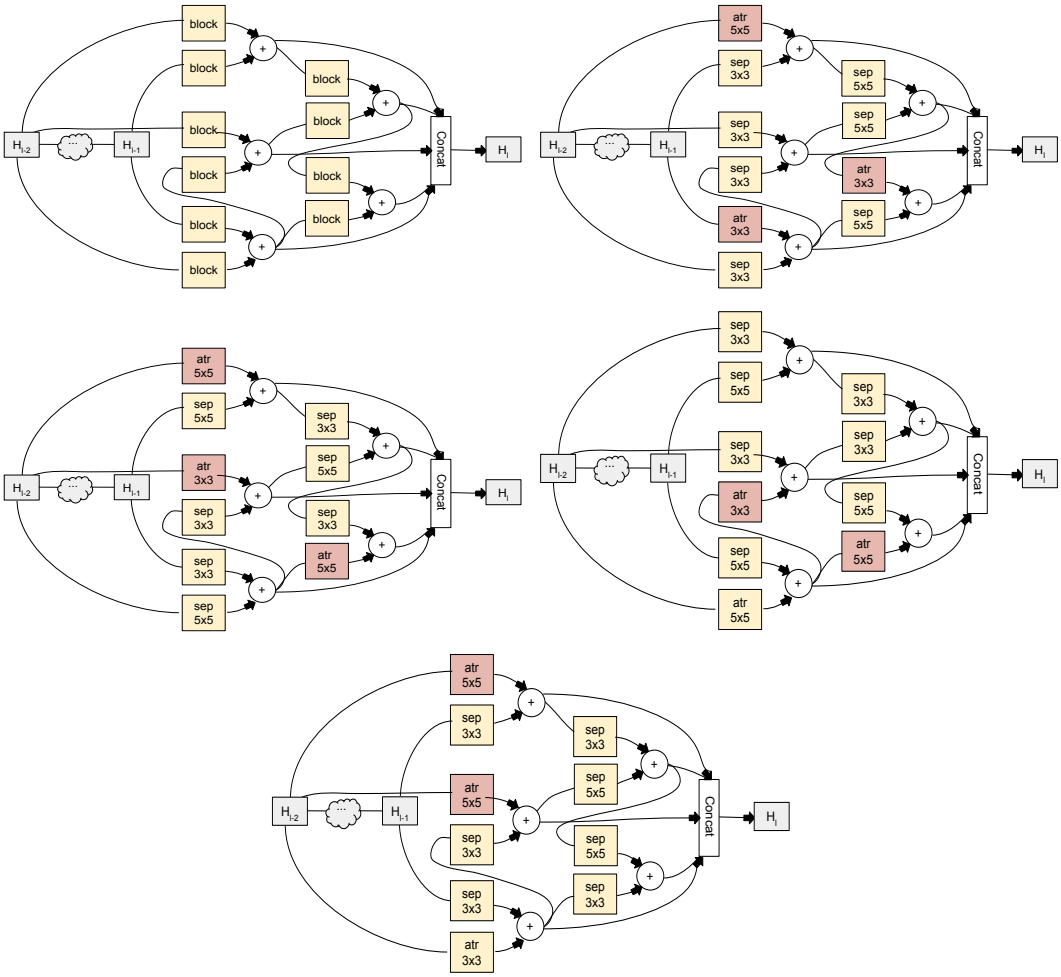


Figure 4.6 To the left in 1st row is Conv-module skeleton taken from [72]. From the 1st row right to the bottom in the scanning order are the ERFNet modified models with Conv-module Cell structures named as A*, B*, C*, D*.

Layer	Type	Layer	Type
1	**Downsampler block**	1	**Downsampler block**
2	**Downsampler block**	2	**Downsampler block**
3-5	**3 x Conv-module**	3	**1 x Conv-module**
5-7	**2 x Conv-module**	4	**1 x Conv-module**
8	**Downsampler block**	5	**Downsampler block**
9-16	**8 x Conv-module**(dilated)	6	**Conv-module**(dilated 2)
17	**Deconvolution**(upsampling)	7	**Deconvolution**(upsampling)
18-19	**2 x Conv-module**	8	**1 x Conv-module**
20	**Deconvolution**(upsamling)	9	**Deconvolution**(upsamling)
21-22	**2 x Conv-module**	10	**1 x Conv-module**
23	**Deconvolution**(upsampling)	11	**Deconvolution**(upsampling)

Table 4.1 Modified version(left) and Compressed version (right) of ERFNet architecture that is used to run experiments on Cityscapes dataset and IDD-lite, respectively.

4.5.2 Implementation details

The best cell structure identified using architecture search gave 64.54% IoU (Model A) on Cityscapes dataset [25]. We also take 3 models (B, C, D) from the pool of possible Conv modules and verify if the performance on the smaller dataset match with that on Cityscapes. The quantitative and the qualitative results are presented in Table 4.3 and Figure 4.7. It is to be noted that though the architecture search was conducted on IDD-lite, we were able to get the best-performing architecture for a different domain like Cityscapes.

4.5.3 Correlation to Efficient Segmentation Models

There have been lots of interest in efficient CNN module designs that have lower compute needs, while still achieving good prediction accuracies [51, 139]. [119] reports results with architecture variations of ERFNet named as D*, D2*, D4*, D8* on Cityscapes [25]. These correspond to the usage of depthwise separable convolutions instead of the bottleneck modules of ERFNet along with grouping parameters on the 1x1 convolution. We conduct the same experiments on the IDD-lite dataset with compressed ERFNet architecture (see Table 4.1 for quantitative and Figure 4.8 for qualitative results) and show that our results correlate with [119].

4.6 Summary

In this chapter, we introduced the idea of *Neural Architecture Search* and its varied components that can provide impetus in finding novel architectures which are different from human-designed models. We also discussed some of the widely used hyper-parameter tuning algorithms which can be considered as a parent

Models	IoU (CS)	Params	IoU (IDD-lite)
ERFNet	70.45	2038448	53.975
D*	68.55	547120	52.01
DG2*	65.35	395568	50.71
DG4*	61.42	319792	48.88
DG8*	59.15	281904	46.40

Table 4.2 Depthwise Separable Convolution, Groups on ERFNet Architecture tested over IDD-lite dataset using Compressed ERFNet from Table 4.1.

Models	IoU(CS)	IoU(IDD-lite)
A*	64.54	58.15
B*	59.21	56.93
C*	55.96	55.46
D*	52.35	53.64

Table 4.3 Custom Cell architecture on compressed ERFNet tested over IDD-lite dataset correlate with the same tests on Cityscapes Dataset with modified ERFNet from Table 4.1.

task of NAS. We also saw that using these techniques mentioned, we were able to get good performing models on IDD-lite, the results of which correlate with bigger datasets such as Cityscapes [25]. In the next chapter, we will see how we can employ automatic pruning algorithms in the context of semantic segmentation with the help of novel schema of *HyperNetworks* in a resource-constrained setting.

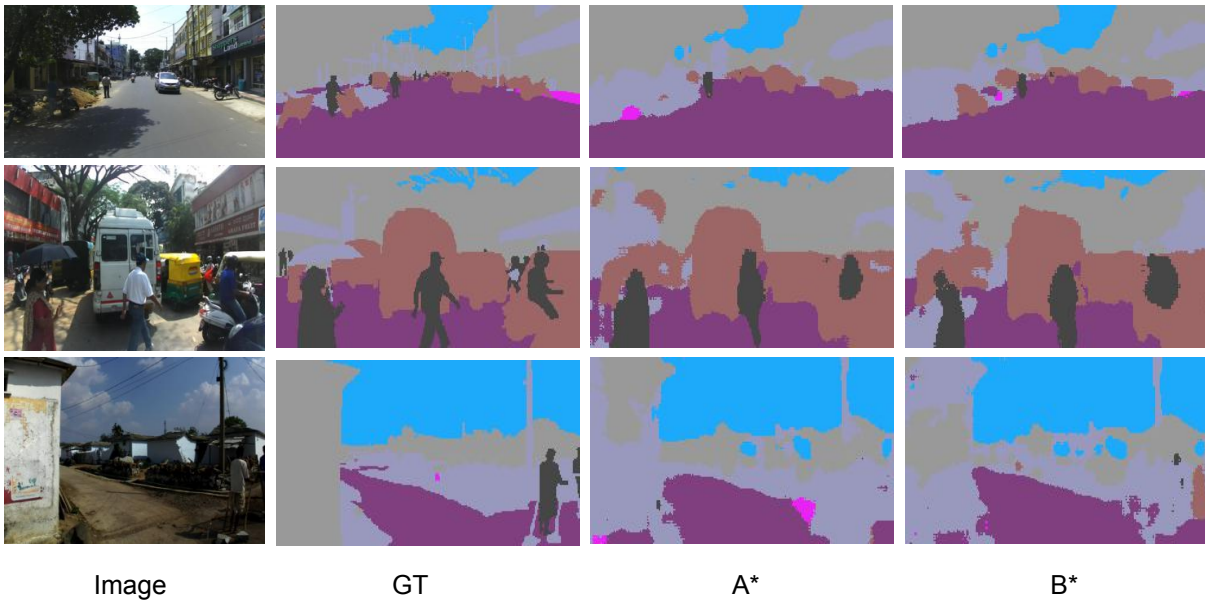


Figure 4.7 Qualitative examples of predictions on validation set from custom cell configuration obtained from the NAS operation as indicated in Table 4.3.

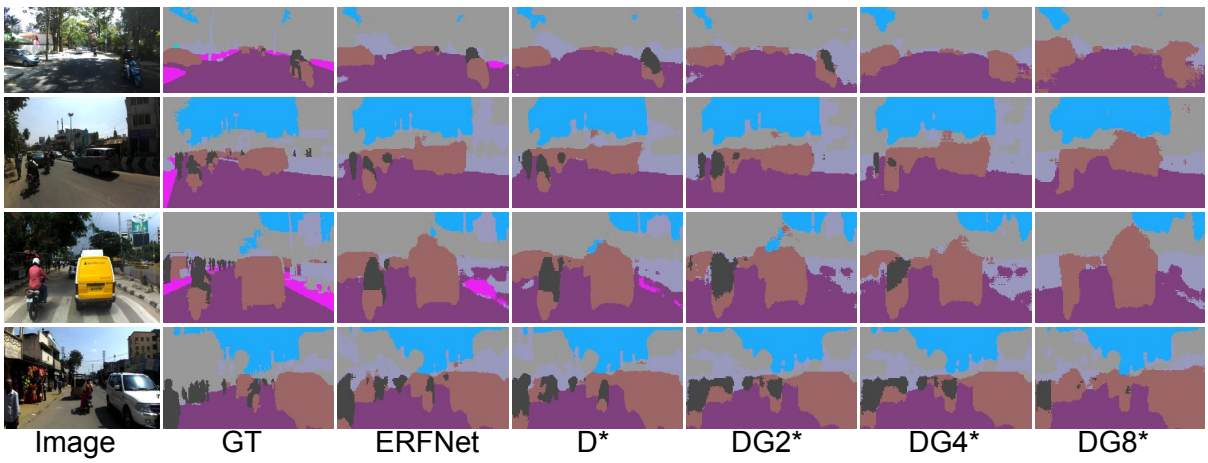


Figure 4.8 Qualitative examples of predictions on validation set generated by models with depth separable convolutions and different grouping parameter trained on samples from the IDD-lite dataset.

Chapter *5*

Automatic Pruning for Resource Constrained Semantic Segmentation

This chapter delves into the paradigm of automatic pruning in the context of semantic segmentation, specifically in a resource-constrained environment, using the proposed datasets introduced in Chapter 3. As discussed in Chapter 2, semantic segmentation plays a crucial role in providing pixel-level understanding of images for various scene-understanding vision tasks. However, semantic segmentation models often demand substantial computational resources during both training and inference, posing challenges in resource-constrained scenarios. To address this issue, we present a compression algorithm based on differentiable meta-pruning through hypernetworks, known as MPHyp. Our proposed method MPHyp leverages hyper networks that take latent vectors as input and output weight matrices for the segmentation model. Utilizing L_1 sparsification through the proximal gradient optimizer, MPHyp updates the latent vectors, introducing sparsity, and enabling automatic model pruning. This approach offers the advantage of achieving controllable compression during training and significantly reducing the training time. We conduct extensive experiments, comparing our methodology with a popular pruning approach, and demonstrate its efficacy by effectively reducing the number of parameters and floating-point operations while maintaining a high mean *Intersection over Union* (mIoU) metric. We perform experiments on two widely accepted semantic segmentation architectures: UNet [104] and ERFNet [103]. Furthermore, our ablation study highlights the effectiveness of our proposed methodology in achieving efficient and reasonable segmentation results, making it a promising solution for resource-constrained environments in semantic segmentation tasks.

5.1 Introduction

Semantic segmentation [20, 21, 80, 137] is a dense prediction task of assigning class labels to each pixel in an image. It has been widely used in autonomous driving, medical imaging, and satellite imaging applications. However, semantic segmentation models such as DeepLabV1,V2 [20, 21], DRN [137] generally have a large number of parameters. So deploying such models in environments with limited

resources, such as mobile phones or embedded systems, can be challenging due to their large memory requirements during inference.

One potential approach to overcome the computational limitations is to adopt methods that reduce the model's complexity, like model compression, for instance, model pruning [33, 48, 81], or neural architecture search [39, 109, 117, 144]. However, these methods above have their inherent problems. For instance, the method [109] is based on a neural architecture search to find the best architecture within a predefined search space. The search space includes various architectural components, such as the number of layers, layer types (convolutional, recurrent, etc.), skip connections, and other hyper-parameters. But, it takes 8 Nvidia Turing GPUs with 24GB of VRAM to find such architectures, which is extremely expensive in terms of training complexity. On the other hand, pruning methods such as the lottery ticket hypothesis [33], a compression algorithm based on iterative pruning have shown a significant segmentation accuracy drop.

In order to efficiently prune semantic segmentation architectures, we propose a compression method based on differentiable meta-pruning using hyper networks MPHyp, that substantially minimizes the training resources and simultaneously retains satisfactory segmentation performance. Our proposed technique is inspired from [67]. We call it meta-pruning since the parameters of the hypernetwork are responsible for generating weights of the segmentation model falling under the umbrella of meta-learning. The key concept behind the proposed approach is the hyper network that associates latent vectors for each layer in the segmentation network. This latent vector control the output channel of the layer in consideration. Given the association of network layers, the latent vector serves as a controlling factor for the subsequent layer's input channel as well. During training, the hyper network receives the latent vectors from the current and preceding layers, which dictate the output and input channels of the current layer, respectively. The hyper network generates a weight matrix for the specific layer of the segmentation model. To achieve automatic pruning, we use the l_1 regularizer that helps in the sparsification of the latent vectors. Subsequently, we employ a proximal gradient to update and obtain the sparsified latent vectors. Together, this strategy leads to differentiability in the pruning mechanism. Once the compression ratio reaches the pre-determined level, the compression method stops. The sparsification of the latent vectors results in compressed outputs from the hyper networks since the latent vectors and layers of the segmentation model are correlated. The proposed method offers the advantage of streamlining network pruning by focusing solely on the latent vectors, eliminating the need for additional complexities or human assistance. We find that our proposed method outperforms the baseline pruning method by a significant margin. We refrain from comparing with neural architecture search methods as they require huge training resources. To showcase the efficacy of our method, we performed extensive experimentation on IDD-lite 3: a semantic segmentation dataset targeted for resource-constraint scenarios. In this context, resource constraint means a lack of availability of better computing power. The images provided in this dataset are sampled and scaled from IDD [121], which are very different compared to other sophisticated semantic segmentation datasets like Cityscapes [25], Mapillary [92]. We chose

UNet [104] and ERFNet [103] as our semantic segmentation models due to the wide acceptability and application of these networks in various domains [76, 82].

In essence, the contribution of this chapter can be summarized as follows:

- We discuss in detail different types of pruning algorithms commonly employed, including various manual and automatic pruning methods (See Section 5.2).
- In addition to that, we engage in comprehensive discussions regarding the various compression techniques employed within the framework of semantic segmentation (See Section 5.3).
- We present the innovative notion of *HyperNetworks*, which constitutes a family of neural networks specifically designed to generate weights for larger networks. Moreover, we delve into an extensive exploration of their versatile applications in diverse sub-fields of vision (See Section 5.4 for more details).
- Additionally, we conduct a thorough investigation into the pivotal role played by *HyperNetworks* in the process of automated pruning of semantic segmentation models (refer to Section 5.5 for comprehensive insights).
- To substantiate our claims, we conduct a series of experiments and present both qualitative and quantitative results, effectively demonstrating the significant reduction in model size attained while adhering to a resource-constrained budget (refer to Section 5.6 for detailed experimental findings).

Finally, we conclude the chapter with a succinct summary of our findings and the potential implications of utilizing *HyperNetworks* for efficient model pruning and resource optimization.

5.2 Comparison of Pruning Algorithms: Manual vs. Automatic

Network pruning is a widely used technique to reduce redundancy in deep neural networks. By selectively removing certain parts of the network, we can optimize its performance and efficiency. Different components of the network, such as convolutional layers and fully connected layers, have varying impacts on factors like computational complexity and model size. Thus, we can achieve specific optimization goals by targeting different parts for pruning.

The pruning process typically involves three main stages:

Training: During training, the model's parameters are adjusted to learn the specific task at hand from a large dataset. Once training is completed, pruning is applied to the trained model.

Pruning: In this stage, we focus on identifying and removing redundant parts of the network structure. These could include filters, blocks, channels, and other components that might not significantly contribute to the model's performance. To achieve efficient pruning results, the algorithms used for identifying unnecessary structures play a critical role in various pruning methods.

Fine-tuning: Fine-tuning is a crucial step that follows the pruning operation and aims to recover and enhance the performance of the pruned model. After pruning, the model may experience a decline in performance due to the removal of certain components. Fine-tuning addresses this issue by retraining the pruned model on the original task-specific dataset. During fine-tuning, only the remaining parameters in the pruned model are updated while keeping the previously pruned components fixed. This process helps the model adapt and regain its predictive capabilities on the task at hand. The fine-tuning process typically involves using a lower learning rate than the initial training, as the model is already partially trained and should only undergo modest updates to prevent over-fitting.

Re-pruning: Re-pruning is an iterative process that aims to further optimize the pruned model's performance and efficiency. Once the model undergoes fine-tuning, it becomes a sub-model containing both pruned and retained components. This sub-model is then subjected to another round of evaluation by the pruning algorithm. The pruning algorithm reevaluates the importance of the remaining components based on the updated weights and activations obtained from the fine-tuning process. During re-pruning, the algorithm identifies additional components that can be safely pruned without significantly affecting the model's performance. The re-pruning process iteratively repeats, gradually refining the model's architecture to meet specific pruning objectives, such as reducing computational complexity or model size. The number of iterations for re-pruning can be pre-defined, or it can be determined based on specific convergence criteria or validation performance. By going through multiple iterations of fine-tuning and re-pruning, the model's performance can be further improved while maintaining its efficiency. It is essential to strike a balance during the re-pruning process, as aggressive pruning may lead to the loss of critical information, and too conservative pruning might not yield substantial efficiency gains. Therefore, careful experimentation and analysis are necessary to determine the optimal pruning rate for achieving the desired trade-off between model size and performance.

The majority of deep neural network pruning techniques are *manual pruning* algorithms, as illustrated in Figure 5.1(a), can be categorized into ***unstructured pruning*** [42, 43, 133] in which arbitrary parameters are removed, and ***structured pruning*** [77, 123] where sets of weights are eliminated. Structured pruning further divides into *filter pruning*, and *channel pruning* [125]. In filter pruning, entire filters are removed based on specific metrics, preserving the original convolutional structure and avoiding the need for dedicated hardware [66]. Some studies use the l_1*-norm* to judge filter importance [66], while others use feature map rank [68] or geometric median [47] to prune less important filters. Other works consider the contribution of filters based on output channel importance [81] or employ Taylor expansion as a pruning criterion [88]. Channel pruning treats the same output channel of different filters as a group. Some studies propose LASSO regression-based channel selection and least squares reconstruction for channel pruning [48]. In contrast, another approach leverages the scaling factor in the Batch Normalization (BN) layer to measure channel importance [77]. Mitsuno et al. [85] conducted an analysis of channel effects on model performance, providing valuable insights into channel pruning.

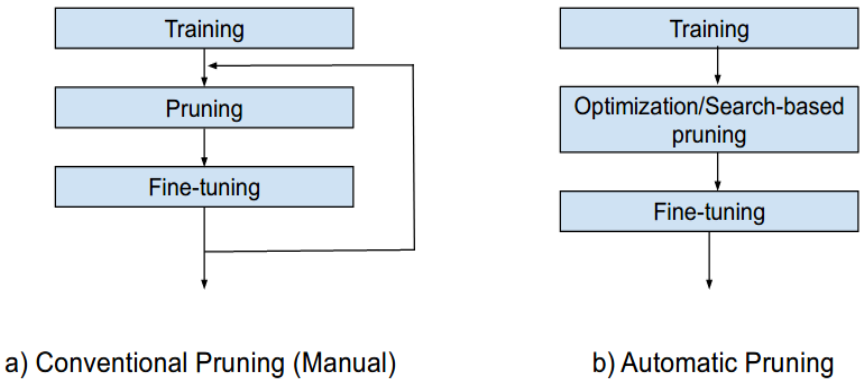


Figure 5.1 Pruning techniques: *Manual* vs *Automatic*. a) After the fine-tuned model is obtained, the model undergoes further pruning operations in order to remove any redundant connections. b) In automatic pruning, the algorithm, through the use of advanced search/optimization-based methods, automatically prunes the unproductive parameters without any human intervention and finally fine-tunes the model to recover the lost performance.

Conventional manual pruning methods, as discussed above, heavily depend on the expertise of human practitioners to strike a balance between computational complexity, parameter size, and model accuracy. However, these manual approaches often struggle to identify the optimal pruning rate strategy, leading to sub-optimal results. To address this constraint, *automatic network pruning* algorithms, as depicted in Figure 5.1(b), are designed to automatically eliminate redundant connections within the network, while adhering to specific constraints, such as a predetermined number of parameters, FLOPs (floating-point operations), or hardware platform limitations. For instance, Liu et al. [79] proposed leveraging automated machine learning (AutoML) techniques to determine pruning hyper-parameters through learning, automating the pruning process for improved efficiency and effectiveness. Determining the adaptive compression rate for each layer in a deep neural network has also been attempted using reinforcement learning [46]. Moreover, a platform-aware neural network adaptation technique called *NetAdapt* [132] was introduced, aiming to sequentially determine suitable pruning rates through progressive compression and short-term fine-tuning. Similarly, [138] leverages a layer-by-layer greedy search approach to automate the compression process for models. Some AutoML-based pruning methods also pay attention to the compression rate of pruning. For instance, [68] introduces predefined hierarchical pruning rates, which unfortunately restrict its ability to search for more precise and optimal pruning rates. On the other hand, [65] proposes a method to find the relative optimal pruning rate but is limited to selecting from a few pre-defined candidate models. In contrast, [69] supports automatic search of pruned structures but lacks hierarchical pruning capabilities, resulting in sub-optimal compression rates. One of the major challenges in previous AutoML-based methods is the absence of sensitivity analysis and subsequent optimization of pruning schemes, making it difficult to determine the optimal number of filters for each

convolutional layer accurately. Recent research indicates that the key aspect of pruning is determining the number of filters, rather than focusing solely on selecting important filters or channels. Addressing this concern, Cai et al. [16] propose a method that measures the sensitivity of each quantization scheme to obtain a scheme with reduced loss. Inspired by this work [16, 72], the APRS method [113] considers the total sensitivity and approaches the problem of automatic pruning as a search for an appropriate number of filters. This approach leverages Pareto optimization in the model formulation, allowing for precise reduction of parameter size or computational cost while preserving performance.

In conclusion, automatic pruning techniques have emerged as a critical solution for reducing the computational burden and memory footprint of deep neural networks at the least expense of compromising their performance. These methods leverage various strategies, such as sensitivity analysis and optimization, to efficiently identify and remove non-essential parameters from the network architecture.

In the forthcoming section, we will delve into the compression techniques that have been employed in the context of semantic segmentation. Subsequently, we will introduce the concept of hypernetworks and how it can be employed as a potent tool that synergizes with automatic pruning, offering a novel and effective approach to network compression in the domain of semantic segmentation.

5.3 Semantic Segmentation Compression

In the context of compression for semantic segmentation, neural architecture search (NAS) based methods have been adopted to reduce computation overload and find lightweight models. By exploring a predefined search space, NAS-based techniques aim to identify superior model architectures. For example, in semantic segmentation, an approach proposed by Liu et al. [72] focuses on discovering a repeatable cell structure and network architecture to enhance performance. However, achieving the target architecture in NAS often comes at the cost of significant memory overhead. In the realm of compression methods tailored for semantic segmentation, various techniques have been proposed, specifically targeting pruning. Architecture pruning, in general, seeks to eliminate redundancies in over-parameterized models to enable faster inference while preserving essential model parameters. Some approaches, like those presented in [43, 87], aim to compress models by leveraging specific hardware architecture or library support, referred to as non-structured pruning. Conversely, other methods perform pruning across the entire network architecture, as seen in [48, 81]. However, the performance of these pruning methods for complex tasks like semantic segmentation remains an open question and requires further investigation.

5.4 HyperNetworks

HyperNetworks were first proposed in [41] that used a smaller network to generate weights for a bigger network. HyperNetworks have been widely used for neural architecture search [12], temporal forecasting [96], and also model compression [78]. The key idea behind hypernetwork is based on

evolutionary methods that evolve smaller networks to generate weights of a larger network. A more efficient method to generate during the weight generation process, the search space is constrained within a smaller weight space [112]. Alternatively, the structure of the network can be kept fixed while weights are evolved through discrete cosine transform. This technique is referred to as a compressed weight search [59]. Similar to the aforementioned approach is Differentiable Pattern Producing Networks (DPPNs) that evolve the structure of the network while the weight parameters are learned [31]. Apart from evolutionary-based algorithms, CNN-based approaches such as [54] generate weights to apply a transformation to the images. [56] uses the concept of dynamic filters that are generated based on input for the task of spatial transformation and deblurring. As discussed in Chapter 2, segmentation models are generally large, and compression techniques such as neural architecture search require extensive computational resources to extract efficient semantic segmentation models. In the following section, we will demonstrate how hyper networks can serve as a solution to the aforementioned problems within an automatic pruning framework, ensuring efficient model compression with minimal training resources.

5.5 Automatic Pruning via HyperNetworks for Semantic Segmentation

In this section, we discuss the process of compression of semantic segmentation models using hypernetwork. Figure 5.2 illustrates the overall design of the proposed method, which is discussed in subsequent sections in more detail.

5.5.1 Architectural Design

We now describe the process of adapting a hypernetwork into a semantic segmentation network. The proposed hypernetwork architecture constructed for a single-layer neural network has the following layers:

- ***Latent Layer***: that takes latent vectors as input and generates latent matrix.
- ***Embedding Layer***: is responsible for projecting latent matrix to embedding matrix.
- ***Output Layer***: transforms the embedding matrix to a final weight matrix of the corresponding layer.

As described above, we know the process of adapting a single-layer neural network to a hypernetwork, so we extend the same process for a semantic segmentation model. Suppose, we have a semantic segmentation model of K-a layered network that requires to be pruned. Initially, we introduce a latent vector to all the layers that need to be pruned, as latent vectors are responsible for generating pruned weights. We keep the shape of the latent vector equal to the output channels for all the pruned layers to keep dimension consistency.

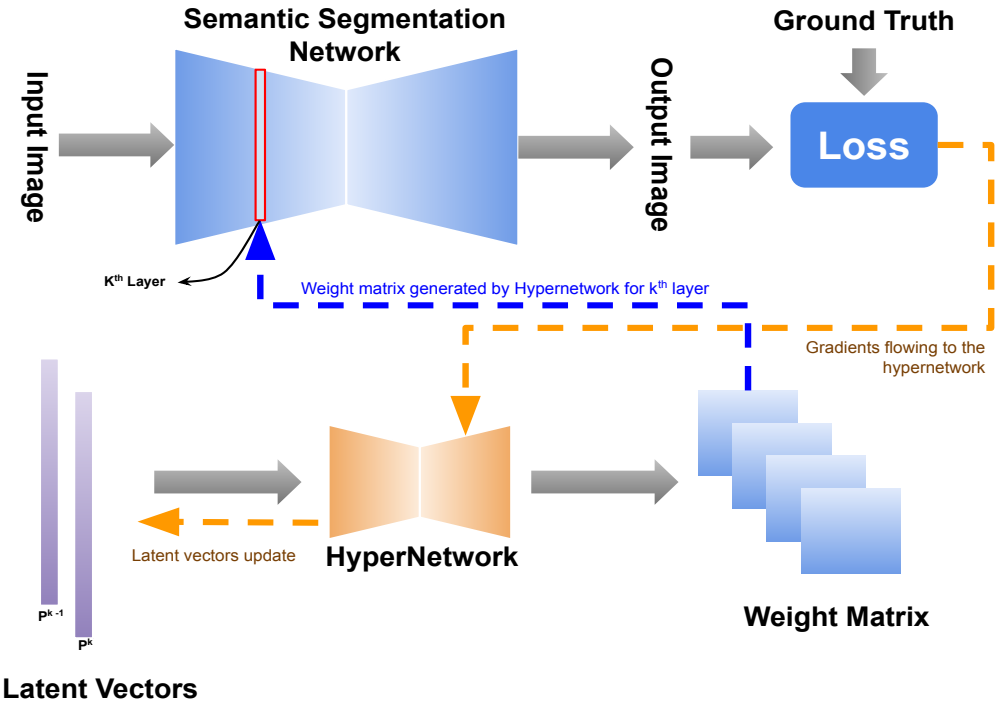


Figure 5.2 Shows a pictorial representation of the proposed algorithm designed for compressing semantic segmentation networks using automatic differentiable pruning. For a layer of the segmentation network is associated with a latent vector. This latent vector is responsible for generating weights of the specific layer. While training, latent vectors get sparsified leading to the pruning of the weights of the specific layer. The construction of the algorithm is such that the latent vector is covariant with its corresponding weight matrix. Weights produced by the hyper network generate the final output through the segmentation model.

Assume for a given semantic segmentation layer k, the dimensionality of the weight matrix generated by k^{th} layer is $l \times m \times w \times h$, where l and m denote the output and input channels of k^{th} semantic segmentation layer, and $w \times h$ denotes its corresponding kernel width and height. Also, consider the latent vector for the k^{th} layer to be p^k. Since, the size of the latent vector is equal to the number of output channels, the latent vector for k^{th} layer is $p^k \in R^l$. Thus, the dimensionality of the latent vector of the $k\text{-}1^{th}$ layer is $p^{k-1} \in R^m$.

Now, we pass latent vectors p^k and p^{k-1} as input to the hypernetwork corresponding to the k^{th} layer to generate the latent matrix, given by,

$$\mathbf{P}^k = \mathbf{p}^k . \mathbf{p}^{k-1^T} + \mathbf{V}_0^k, \tag{5.1}$$

where,

$$\mathbf{P}^k, \mathbf{V}_0^k \in R^{n \times c}$$

$[T]$ represents the transpose of the matrix and [.] denotes matrix multiplication. $\mathbf{V}_0$ is the bias matrix. Consequently, the latent matrix is further projected to an embedding dimension with the help of the

embedding layer that is given by,

$$\mathbf{C}_{ij}^{k} = \mathbf{P}_{ij}^{k}.\mathbf{w}_{1}^{k} + \mathbf{v}_{1}^{k} \quad i = 1..l, j = 1...m, \tag{5.2}$$

$\mathbf{C}_{ij}^{k}, \mathbf{w}_{1}^{k}, \mathbf{v}_{1}^{k} \in R^{e}$, where ***e*** represents the embedding dimension. The elements of $\mathbf{w}_{1}^{k}$ and $\mathbf{v}_{1}^{k}$ are considered to be unique. We exclude the subscript *(i,j)* for ease of understanding. Now, we multiply $\mathbf{w}_{1}^{k}$, $\mathbf{v}_{1}^{k}$ and $\mathbf{P}_{ij}^{k}$ together to form a 3D matrix, given as $\mathbf{W}_{1}^{k}$, $\mathbf{V}_{1}^{k}$ and $\mathbf{C}_{1}^{k} \in R^{l \times m \times e}$ as formulated in Equation 5.2. We generate the final output $\mathbf{G}_{ij}^{k}$ by passing the embedding matrix, obtained in Equation 5.2 through the output layer, represented in Equation 5.3.

$$\mathbf{G}_{ij}^{k} = \mathbf{C}_{ij}^{k}.\mathbf{w}_{2}^{k} + \mathbf{v}_{2}^{k} \quad i = 1..l, j = 1...m, \tag{5.3}$$

where,

$$\mathbf{G}_{ij}^{k}, \mathbf{v}_{2}^{k} \in R^{wh}$$

and,

$$\mathbf{w}_{2}^{k} \in R^{wh \times e}$$

We can observe the vectors $\mathbf{w}_{2}^{k}$, $\mathbf{v}_{2}^{k}$ and $\mathbf{G}_{ij}^{k}$ generate the final weight of the segmentation layer *k*. The final dimensionality of variables involved in Equation 5.3 are $\mathbf{W}_{2}^{k} \in R^{l \times m \times wh \times e}$ and $\mathbf{V}_{2}^{k}$ and $\mathbf{G}^{k} \in R^{l \times m \times wh}$.

In functional form, we can represent the output $\mathbf{G}^{k}$ as:

$$\mathbf{G}^{k} = h(\mathbf{p}^{k}, \mathbf{p}^{k-1}; \mathbf{W}^{k}, \mathbf{V}^{k}), \tag{5.4}$$

Here, h(.) indicates the overall transformation applied on the input latent vectors, parameterized by latent vectors and per layer weight tensors. In case of skip or residual connections present in the segmentation network, we handle it by concatenating the latent vector of the current layer and the associated skip connection layer to create the latent matrix using Equations 5.1 and 5.3.

5.5.2 Sparsification of Latent Vectors

The next step of our proposed method is to introduce sparsification in latent vectors that are associated with layers of the semantic segmentation model. We introduce sparsity in latent vectors because all the latent vectors are connected to each other, and each latent vector is covariant with the output channels of the specific layer of the segmentation network. Sparsification of latent vectors leads to a reduction in the count of output channels leading to compression of the network. We achieve this by constraining latent vectors under L_1-norm formulated as:

$$R(p) = \sum_{k=1}^{K} ||p^{k}||_{1} \tag{5.5}$$

where *R* is the regularization term. Now, we introduce differentiability with the help of the proximal gradient algorithm. In summary, the task of the proximal gradient algorithm is to sparsify and update the latent vectors. The proximal update equation is given by,

$$\mathbf{p}[k+1] = \mathbf{prox}_{\lambda\mu R}(p[k] - \lambda\mu\nabla L(p[k])) \tag{5.6}$$

5.5.3 Semantic Segmentation Network Pruning

After the sparsification of latent vectors by employing L_1-norm followed by proximal operation, the latent vectors are forced to become sparse without any human assistance leading to automatic pruning. The latent vector sparsification leads to the generation of pruned weights for the corresponding layer of the segmentation network. After the compression stage, we obtain sparse latent vectors p^k and p^{k-1} for k^{th} semantic segmentation layer. After this, we apply masks t^k, t^{k-1} on the pruned vectors that are near zero with a predefined threshold τ. If the value is greater than the threshold, the returned value is one else zero. The sparsified latent vector, $\bar{p}^k$ is pruned using mask (t^k). After the desired compression ratio is met, we then fine-tune the pruned semantic segmentation network to improve the accuracy. We refer to fine-tuning as a converging stage. The hypernetwork defined earlier gets scrapped, followed by the onset of the normal training regime. It is important to note that if the compression ratio increases, then the number of compression epochs also increases.

5.6 Experiments

5.6.1 Models and Dataset

We employ UNet [104] and ERFNet [103] as the base semantic segmentation architectures on which all the experiments are performed. We use IDD-lite, introduced in Chapter 3 for our experiments. IDD-lite is a custom dataset specifically designed for resource-constrained scenarios. This dataset contains 1400 training images and 400 validation images sampled and scaled from the updated IDD dataset [121], keeping the largest dimension to 320 while preserving the image's aspect ratio with the hierarchy of 7 coarse labels.

5.6.2 Training Detials

We use a single NVIDIA 1080Ti GPU for simultaneous training and compression through our proposed method using Pytorch deep learning library. The initial learning rate is set to 0.1, and total epochs are set to 250. We employ a reduced learning rate on the plateau scheduler. The training batch size is kept at 32, and the validation batch size is 4. The embedding dimension is fixed to 8 in all the experiments. The sparsity regularization factor is fixed to be 0.5. Consequently, once the difference between the target compression factor and the actual compression factor is within a range of 2% of the target compression factor, the model is fine-tuned, and the compression stage stops.

5.6.3 Performance Metric

We evaluate the performance of pruned semantic segmentation model through mean *Intersection over Union* metric, given as:

$$IOU = \frac{TP}{TP + FP + FN} \tag{5.7}$$

Here, TP, FP, and FN are the number of true positives, false positives, and false negatives at the pixel level. For assessing the efficiency of pruning methods, we use the number of (a) floating point operations (GFlops), (b) parameters, and (c) flop ratio indicating the ratio of remaining flops and original flops.

UNet					
Compression Ratio	Method	mIoU	GFlops	Params(M)	Flop Ratio
0.00	Un-Pruned	65.71	20.84	7.786	-
0.50	L_1-Pruning [131]	43.18	10.42	3.897	0.5
	MPHyp	55.03	10.73	3.504	0.514
0.75	L_1-Pruning [131]	26.71	5.212	1.951	0.25
	MPHyp	57.01	5.369	0.702	0.257
0.90	L_1-Pruning [131]	19.09	2.098	0.778	0.10
	MPHyp	54.61	2.298	1.015	0.1102
0.95	L_1-Pruning [131]	18.93	1.042	0.406	0.05
	MPHyp	52.9	1.361	0.76	0.065
ERFNet					
Compression Ratio	Method	mIoU	GFlops	Params(M)	Flop Ratio
0.00	Un-Pruned	58.06	3.72	2.063	-
0.50	L_1-Pruning [131]	56.52	1.867	0.993	0.5
	MPHyp	55.49	1.794	0.959	0.482
0.75	L_1-Pruning [131]	53.70	0.931	0.499	0.250
	MPHyp	55.14	0.937	0.408	0.251
0.90	L_1-Pruning [131]	51.64	0.368	0.21	0.098
	MPHyp	52.24	0.403	0.128	0.118
0.95	L_1-Pruning [131]	41.42	0.183	0.117	0.049
	MPHyp	51.63	0.256	0.109	0.068

Table 5.1 Shows the performance of our proposed method MPHyp on UNet and ERFNet. It can be observed that our method shows better performance than L_1-pruning, with better accuracy and having a smaller network.

5.7 Results

5.7.1 Quantitative Results

In this section, we discuss the results obtained from our proposed method: Meta-pruning based Hyper-Network (MPHyp) on UNet and ERFNet.

Methods	Drivable	Non-drivable	Living Thing	Vehicles	Roadside-Objects	Far-objects	Sky	mIoU
Compression Ratio:50 %								
Un-Pruned	92.51	36.89	49.97	69.79	43.19	72.96	94.67	65.71
L_1-Pruning [131]	91.1	4.07	47.03	65.87	0.0	0.0	94.15	43.18
MPHyp	91.2	31.42	43.39	67.46	41.29	71.78	93.69	55.03
Compression Ratio:75 %								
Un-Pruned	92.51	36.89	49.97	69.79	43.19	72.96	94.67	65.71
L_1-Pruning [131]	90.16	3.34	0.0	0.0	0.0	0.0	93.5	26.71
MPHyp	92.61	33.26	47.06	**72.28**	**44.27**	72.72	93.92	57.01
Compression Ratio:90 %								
Un-Pruned	92.51	36.89	49.97	69.79	43.19	72.96	94.67	65.71
L_1-Pruning [131]	40.12	0.00	0.00	0.00	0.00	0.00	93.51	19.09
MPHyp	91.61	28.87	43.48	67.07	41.93	70.38	93.51	54.61
Compression Ratio:95 %								
Un-Pruned	92.51	36.89	49.97	69.79	43.19	72.96	94.67	65.71
L_1-Pruning [131]	40.15	0.00	0.00	0.00	0.00	0.00	92.36	18.93
MPHyp	91.06	22.34	41.64	66.00	40.34	68.86	92.97	52.90

Table 5.2 Shows the mIoU performance of MPHyp at various compression ratios for UNet architecture. We can observe that MPHyp significantly outperforms L_1-Pruning with higher mIoU at a higher compression ratio and shows comparable performance with the Un-Pruned network. It is also important to note that MPHyp shows better performance (in Bold) than the un-pruned method on the Vehicles and Roadside-Objects class, which could be advantageous in autonomous applications.

Method	Embedding Dimension	mIoU	Flop Ratio
MPHyp	8	55.03	0.5114
MPHyp	16	54.22	0.5196
MPHyp	32	53.82	0.5042

Table 5.3 Shows the MPHyp performance at various embedding dimensions. We can observe as the embedding dimension increases, the semantic segmentation decreases. However, on the other hand, we have improved the flop ratio with increasing embedding dimension. We keep the pruning ratio fixed at 0.5.

Table 5.1 shows the performance of MPHyp with L_1-based channel pruning. We can observe that MPHyp preserves the segmentation accuracy at different compression ratios, whereas L_1-pruning shows a significant drop in mIoU. Also, it is evident that for higher compression ratios, our proposed approach outperforms the L_1-pruning technique, further decreasing the flops and parameter count. Simultaneously, MPHyp achieves a lower flop ratio and GFlops, resulting in faster network inference. We also show class-wise mIoU results for different pruning methods in Table 5.2 at different compression ratios. It can be observed MPHyp significantly outperforms L_1-Pruning with higher mIoU at a higher compression ratio and shows comparable performance with an Un-Pruned network. Interestingly, MPHyp shows better performance than the un-pruned method on the Vehicles and Roadside-Objects class, which are important classes in autonomous applications. On average, the training time of MPHyp at different compression ratios was around 1Hr on 1080ti, which is significantly lower than existing neural architecture search methods and L_1-pruning which has an average training time of 3Hrs.

Method	Proximal Regularization	mIoU	Flop Ratio
MPHyp	L_1	55.03	0.5114
MPHyp	L_2	53.81	0.5293

Table 5.4 Shows the MPHyp performance for different sparsity regularization formulations at a pruning ratio of 0.5.

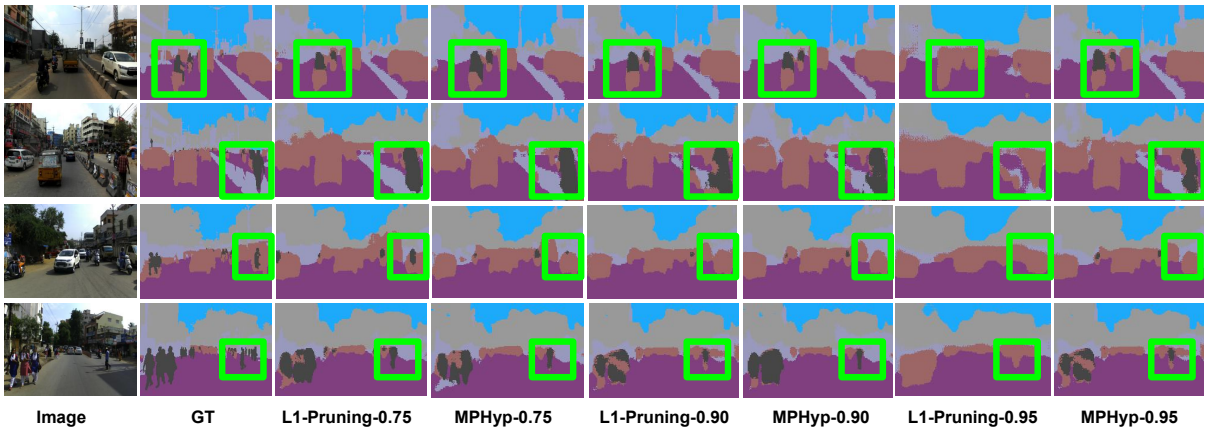


Figure 5.3 Qualitative results at different compression ratios comparing L_1-pruning [131] and our proposed method is based on ERFNet architecture. From the results, it is evident that our proposed methodology performs better than L_1-pruning [131] in identifying small and living objects class. Segmentation results enclosed in green boxes show that MPHyp is able to preserve segmentation results at various pruning rates for different classes.

5.7.2 Qualitative Results

Figure 5.3 and Figure 5.4 displays the qualitative results obtained from our proposed approach and the baseline pruning algorithm for ERFNet and UNet architectures respectively. It is evident from Figure 5.3 that our proposed approach that MYHyp predictions are better for both the *rider* and *pedestrian* classes compared to L_1-pruning. In Figure 5.4, the first row demonstrates the prediction of classes *rider* and *bike*, and the second row focuses on the model's prediction for *car* class. The third and fourth rows illustrate classes *sidewalk* and *sky* respectively. As it can be observed, MYHyp predictions have better fine details compared to L_1-pruning.

5.8 Ablations

We perform two ablations to see the performance of our proposed approach.

Embedding dimension: We increase the embedding dimension from the default value of 8 to 16 and 32 in Table 5.3. We see that there is a trade-off between the mIoU value and the flop ratio. Though the

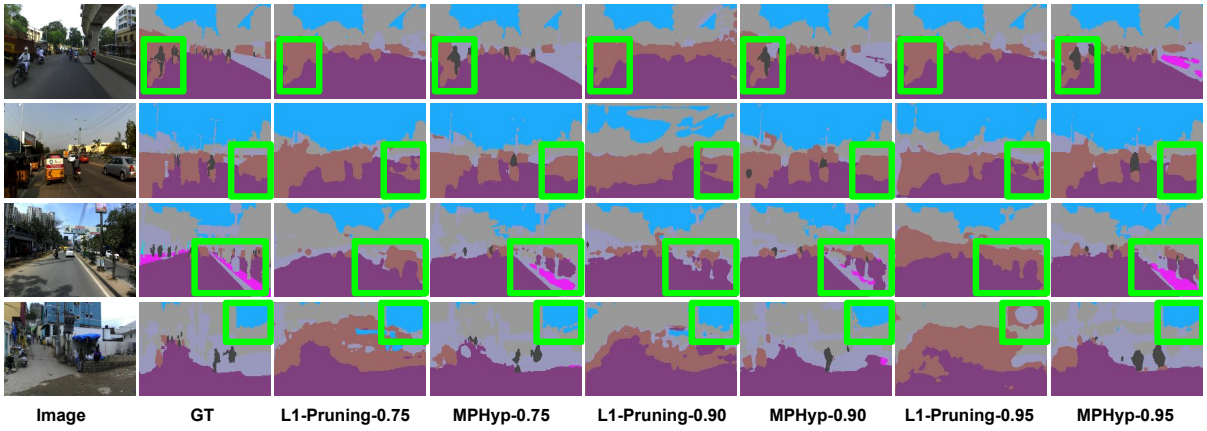


Figure 5.4 Qualitative results at different compression ratios comparing L_1-pruning [131] and our proposed method is based on UNet architecture. From the results, it is evident that our proposed methodology performs better than L_1-pruning [131] in identifying small and living objects class. Segmentation results enclosed in green boxes show that MPHyp is able to preserve segmentation results at various pruning rates for different classes.

flop ratio is marginally better for the higher embedding dimension, the mIoU metric follows a downward trend.

Proximal Gradient Regularizer: We also compare the effect of L_2 regularizer and L_1 regularizer to sparsify the latent vector, keeping the same pruning strategy in Table 5.4. We can observe that L_1 regularizer shows better mIoU and flop ratio compared to L_2 regularizer. It is also important to note that to have a reasonable convergence speed λ used for L_2 regularizer must be significantly larger than that L_1 regularizer.

5.9 Summary

Within this chapter, we have extensively explored various pruning algorithms readily available in the field, as well as compression algorithms commonly employed for the task of semantic segmentation. Furthermore, we also looked into the *HyperNetworks* and their potential application in automatic pruning. The proposed solution offers the unique advantage of differentiability due to the integration of hypernetworks with semantic segmentation architectures. The use of the proximal gradient algorithm along with L_1 sparsification, complemented by the proposed hypernetwork design, facilitates the efficient discovery of a compact representation for the given architecture. To validate the efficacy of our approach, we conducted comprehensive experiments utilizing two widely accepted semantic segmentation architectures, thus providing substantial evidence of its effectiveness. In essence, our method offers a solution and opens research avenues for efficient pruning techniques for semantic segmentation using *HyperNetworks* in a resource-constrained environment.

Chapter *6*

Conclusion

This chapter summarizes the definitive results of the research done throughout this thesis. The chapter also includes a concise yet insightful discussion of the potential future directions that can be explored based on the findings and results achieved in this thesis.

6.1 Summary

The thesis began with Chapter 1, where we focused on the practical challenges encountered during the widespread adoption of large-scale pixel-perfect annotated semantic segmentation datasets like Cityscapes [25] and IDD [121]. We addressed the complexities associated with training semantic segmentation models on such massive datasets. To cater to Indian scenarios, we presented the motivation behind proposing resource-constrained datasets, exploring their applicability in (i) *teaching*, (ii) *architecture search*, and (iii) *automatic pruning*.

In Chapter 2, we defined and discussed the problem of semantic segmentation, while also exploring various deep-learning approaches for supervised semantic segmentation. Additionally, we delved into different evaluation metrics used for segmentation assessment and explored a range of datasets available for training segmentation models from diverse geographical contexts.

In Chapter 3, we emphasized the necessity of having a dataset specifically tailored to the Indian road scenario for the development of an autonomous navigation-based system, modified to handle the complexities of the unstructured environment present in India. We provided an in-depth overview of the proposed IDD dataset [121], along with the toolkit utilized for annotating such large-scale datasets. Furthermore, we presented the annotation and quality-checking pipeline especially designed for IDD [121]. As we recognized the lack of publicly available datasets suited for resource-constrained settings, we proposed two datasets for semantic segmentation within such contexts. The experiments and results confirmed the effectiveness of our proposed solution. We highlight how these datasets were leveraged in the ICCV and NCVPRIPG 2019 challenges to discover more suitable models for inferring semantic segmentation

within a resource-constrained regime. We also explored the applications of these datasets, particularly in the development of a set of notebooks to discuss semantic segmentation. We introduced four notebooks that utilize both traditional and deep learning-based methods for semantic segmentation, providing accompanying screenshots and open-source links.

Proceeding to Chapter 4, we dived into the concept of *Neural Architecture Search* (NAS) and its various components. We demonstrated how NAS can help in identifying cells or blocks that can be stacked together to form novel architectures, with the potential to transfer performance gains to larger datasets. The chapter focused on the application of NAS strategies in diverse vision tasks, including image classification and semantic segmentation. Through experimentation, we validated that novel cell operations obtained using a modified ERFNet [103] architecture on the proposed IDD-lite dataset can be effectively transferred to larger datasets like Cityscapes [25], yielding correlated results. We also presented the results achieved using efficient CNN module designs as proposed in [119].

Lastly, Chapter 5 detailed pruning, particularly focusing on automatic pruning with the additional advantage of *HyperNetworks*. We illustrated how leveraging the proposed IDD-lite dataset can facilitate the discovery of efficient semantic segmentation architectures within resource-limited budgets. This attempt opens up new avenues for exploring practical solutions to address resource constraints effectively.

6.2 Future Directions

One of the possible future directions is to explore other strategies to efficiently perform subset selection in larger datasets. This aims to match the statistics of curated datasets with bigger existing ones, leading to improved validation scores using fewer samples, as done in this set of works. For instance, inter-class variance or label distribution per image could be relevant factors for sub-sampling.

Considering the importance of sample-efficient learning in real-world scenarios, exploring techniques for few-shot or one-shot learning within resource-constrained settings could be another promising research direction. These approaches could allow models to generalize well and adapt quickly to new and unseen data, thus reducing the demand for extensive data collection and annotation.

Finally, considering the broader context of responsible AI, conducting fairness and bias assessments on resource-constrained models is of utmost importance. Research focusing on developing methods to ensure fairness, equity, and transparency in model predictions, particularly within constrained settings, can lead to more ethical and inclusive AI solutions.

Outcomes

1. Conference Paper:

- **Semantic Segmentation Datasets for Resource Constrained Training**
  $Ashutosh\ Mishra^*$, $Sudhir\ Kumar^*$, $Tarun\ Kalluri^*$, $Girish\ Varma$, $Anbumani\ Subramanian$, $Manmohan\ Chandraker$, $CV\ Jawahar$
  Accpeted at National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG) 2019
  (* = equal contribution) ***Citations:*** 10

- **Towards Efficient Semantic Segmentation Compression via Meta Pruning**
  $Ashutosh\ Mishra^*$, $Shyam\ Nandan\ Rai^*$, $Girish\ Varma$, $CV\ Jawahar$
  Accepted at 8^{th} International Conference on Computer Vision and Image Processing (CVIP) 2023
  (* = equal contribution)

2. Related Publications

- IDD: A Dataset for Exploring Problems of Autonomous Navigation in Unconstrained Environments
  $Girish\ Varma$, $Anbumani\ Subramanian$, $Anoop\ Namboodiri$, $Manmohan\ Chandraker$, $CV\ Jawahar$
  Accepted at IEEE Winter Applications on Computer Vision (WACV) 2019

3. Semantic Segmentation Notebooks for Education

- https://github.com/Ashutosh1995/Semseg-Notebooks

4. ICCV Challenge on Semantic Segmentation in Resource Constrained Setting

- http://idd.insaan.iiit.ac.in/evaluation/an19-leader-board/#an2_constrained

5. NCVPRIPG Challenge on Semantic Segmentation in Resource Constrained Setting

- https://cvit.iiit.ac.in/ncvpripg19/idd-challenge/

Bibliography

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels. Technical report, 2010.

[2] H. Alhaija, S. Mustikovela, L. Mescheder, A. Geiger, and C. Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision (IJCV)*, 2018.

[3] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

[4] 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.

[5] B. Baheti, S. Innani, S. Gajre, and S. Talbar. Eff-unet: A novel architecture for semantic segmentation in unstructured environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 358–359, 2020.

[6] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[7] B. Baker, O. Gupta, R. Raskar, and N. Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[8] A. Bearman, O. Russakovsky, V. Ferrari, and L. Fei-Fei. What's the point: Semantic segmentation with point supervision. In *European conference on computer vision*, pages 549–565. Springer, 2016.

[9] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[10] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[11] M. N. Bojnordi and E. Ipek. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–13. IEEE, 2016.

[12] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. SMASH: one-shot model architecture search through hypernetworks. In *ICLR 2018*.

[13] G. J. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

[14] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV (1)*, pages 44–57, 2008.

[15] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[16] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

[17] L. Castrejon, K. Kundu, R. Urtasun, and S. Fidler. Annotating object instances with a polygon-rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5230–5238, 2017.

[18] C. Chen, F. Tung, N. Vedula, and G. Mori. Constraint-aware deep neural network compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 400–415, 2018.

[19] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in neural information processing systems*, pages 8699–8710, 2018.

[20] 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, 2017.

[21] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[22] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[23] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[24] M. D. Collins and P. Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[25] 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 *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[26] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 11398–11407, 2019.

[27] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[28] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[29] T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[31] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot, and D. Wierstra. Convolution by evolution: Differentiable pattern producing networks. In *Proceedings of the GECC 2016*, pages 109–116, 2016.

[32] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary intelligence*, 1(1):47–62, 2008.

[33] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *In International Conference on Learning Representations (ICLR) 2019*.

[34] J. Fritsch, T. Kühnl, and A. Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1693–1700, 2013.

[35] F. Galasso, R. Cipolla, and B. Schiele. Video segmentation with superpixels. In *Asian conference on computer vision*, pages 760–774. Springer, 2012.

[36] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[37] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[38] 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.

[39] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan. Mixed precision neural architecture search for energy efficient deep learning. In *IEEE/ACM ICCAD*, 2019.

[40] B. Gunsel, A. M. Ferman, and A. M. Tekalp. Temporal video segmentation using unsupervised clustering and semantic object tracking. *Journal of Electronic Imaging*, 7(3):592–605, 1998.

[41] D. Ha, A. M. Dai, and Q. V. Le. Hypernetworks. In *In Proceedings of ICLR 2017*.

[42] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *In ICLR*, 2016.

[43] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[44] 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.

[45] X. He, R. S. Zemel, and M. Á. Carreira-Perpiñán. Multiscale conditional random fields for image labeling. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II. IEEE, 2004.

[46] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*, 2018.

[47] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4340–4349, 2019.

[48] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[49] S. Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. *Advances in neural information processing systems*, 8, 1995.

[50] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[51] 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.

[52] C. Huang, L. Davis, and J. Townshend. An assessment of support vector machines for land cover classification. *International Journal of remote sensing*, 23(4):725–749, 2002.

[53] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[54] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. *NeuRIPS*, 28, 2015.

[55] S. D. Jain and K. Grauman. Click carving: Segmenting objects in video with point clicks. In *Fourth AAAI Conference on Human Computation and Crowdsourcing*, 2016.

[56] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. *NeuRIPS*, 29, 2016.

[57] J. Jordan. Skip connections in FCN. `https://www.jeremyjordan.me/semantic-segmentation/`, 2018. [Online; accessed 05-Aug-2023].

[58] T. Kalluri, G. Varma, M. Chandraker, and C. Jawahar. Universal semi-supervised semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5259–5270, 2019.

[59] J. Koutnik, F. Gomez, and J. Schmidhuber. Evolving neural networks in compressed weight space. In *Genetic and evolutionary computation*, pages 619–626, 2010.

[60] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[62] L. Ladickỳ, C. Russell, P. Kohli, and P. H. Torr. Associative hierarchical crfs for object class image segmentation. In *2009 IEEE 12th International Conference on Computer Vision*, pages 739–746. IEEE.

[63] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

[64] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.

[65] B. Li, B. Wu, J. Su, and G. Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 639–654. Springer, 2020.

[66] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *In ICLR*, 2016.

[67] Y. Li, S. Gu, K. Zhang, L. Van Gool, and R. Timofte. Dhp: Differentiable meta pruning via hypernetworks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*, pages 608–624. Springer, 2020.

[68] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020.

[69] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian. Channel pruning via automatic structure search. *In: IJCAI*, 2020.

[70] Z. Q. Lin, B. Chwyl, and A. Wong. Edgesegnet: A compact network for semantic segmentation. *arXiv preprint arXiv:1905.04222*, 2019.

[71] T. Lindeberg and M.-X. Li. Segmentation and classification of edges using minimum description length approximation and complementary junction cues. *Computer Vision and Image Understanding*, 67(1):88–98, 1997.

[72] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019.

[73] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

[74] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[75] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[76] X. Liu, J. Qi, W. Zhang, Z. Bao, K. Wang, and N. Li. Recognition method of maize crop rows at the seedling stage based on ms-erfnet model. *Computers and Electronics in Agriculture*, 2023.

[77] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[78] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *IEEE ICCV*, 2019.

[79] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *In: ICLR*, 2019.

[80] 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.

[81] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[82] J. McGlinchy, B. Johnson, B. Muller, M. Joseph, and J. Diaz. Application of unet fully convolutional neural network to impervious surface segmentation in urban environment from high resolution satellite imagery. In *IGARSS 2019*.

[83] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *Proceedings of the european conference on computer vision (ECCV)*, pages 552–568, 2018.

[84] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3061–3070, 2015.

[85] K. Mitsuno and T. Kurita. Filter pruning using hierarchical group sparse regularization for deep convolutional neural networks. In *2020 25th international conference on pattern recognition (ICPR)*, pages 1089–1095. IEEE, 2021.

[86] H. Mobahi, S. R. Rao, A. Y. Yang, S. S. Sastry, and Y. Ma. Segmentation of natural images by texture and boundary compression. *International journal of computer vision*, 95(1):86–98, 2011.

[87] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In *ICML*, pages 2498–2507. PMLR, 2017.

[88] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *In ICLR*, 2016.

[89] N. Moon, E. Bullitt, K. Van Leemput, and G. Gerig. Automatic brain and tumor segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 372–379. Springer, 2002.

[90] V. Nekrasov, H. Chen, C. Shen, and I. Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9126–9135, 2019.

[91] V. Nekrasov, C. Shen, and I. Reid. Template-based automatic search of compact semantic segmentation architectures. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1980–1989, 2020.

[92] G. Neuhold, T. Ollmann, S. Rota Bulo, and P. Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4990–4999, 2017.

[93] R. Nock and F. Nielsen. Statistical region merging. *IEEE Transactions on pattern analysis and machine intelligence*, 26(11):1452–1458, 2004.

[94] S. Osher and N. Paragios. *Geometric level set methods in imaging, vision, and graphics*. Springer Science & Business Media, 2003.

[95] Pablo. Grid-Search vs Random Search. `https://shorturl.at/fnJQY`, 2020. [Online; accessed 05-Aug-2023].

[96] Z. Pan, Y. Liang, J. Zhang, X. Yi, Y. Yu, and Y. Zheng. Hyperst-net: Hypernetworks for spatio-temporal forecasting. *arXiv preprint arXiv:1809.10889*, 2018.

[97] W. Pedrycz, A. Skowron, and V. Kreinovich. *Handbook of granular computing*. John Wiley & Sons, 2008.

[98] 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.

[99] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *ArXiv*, abs/1603.05279, 2016.

[100] D. Ravì, M. Bober, G. M. Farinella, M. Guarnera, and S. Battiato. Semantic segmentation of images exploiting dct based features and random forest. *Pattern Recognition*, 52:260–273, 2016.

[101] S. Ravi. Projectionnet: Learning efficient on-device deep networks using neural projections. *arXiv preprint arXiv:1708.00630*, 2017.

[102] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[103] 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, 2017.

[104] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[105] S. Rota Bulò, L. Porzi, and P. Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5639–5647, 2018.

[106] C. Rother, V. Kolmogorov, and A. Blake. " grabcut" interactive foreground extraction using iterated graph cuts. *ACM transactions on graphics (TOG)*, 23(3):309–314, 2004.

[107] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[108] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008.

[109] A. Shaw, D. Hunter, F. Iandola, and S. Sidhu. SqueezeNAS: Fast neural architecture search for faster semantic segmentation. In *ICCV Neural Architecture Search Workshop*, 2019.

[110] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European conference on computer vision*, pages 1–15. Springer, 2006.

[111] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[112] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Journal of Artificial Life*.

[113] Q. Sun, S. Cao, and Z. Chen. Filter pruning via automatic pruning rate search. In *Proceedings of the Asian Conference on Computer Vision*, pages 4293–4309, 2022.

[114] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv preprint arXiv:1409.4011*, 2014.

[115] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[116] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[117] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *IEEE Computer Vision and Pattern Recognition Conference*, 2019.

[118] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[119] N. Vallurupalli, S. Annamaneni, G. Varma, C. Jawahar, M. Mathew, and S. Nagori. Efficient semantic segmentation using gradual grouping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 598–606, 2018.

[120] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. 2011.

[121] G. Varma, A. Subramanian, A. Namboodiri, M. Chandraker, and C. Jawahar. Idd: A dataset for exploring problems of autonomous navigation in unconstrained environments. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1743–1751. IEEE, 2019.

[122] T. Wang, B. Han, and J. Collomosse. Touchcut: Fast image and video segmentation using single-touch interaction. *Computer Vision and Image Understanding*, 120:14–30, 2014.

[123] Z. Wang, C. Li, and X. Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14913–14922, 2021.

[124] G.-Q. Wei, K. Arbter, and G. Hirzinger. Automatic tracking of laparoscopic instruments by color coding. In *CVRMed-MRCAS'97*, pages 357–366. Springer, 1997.

[125] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[126] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[127] L. Xie and A. Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017.

[128] A. Xu, L. Wang, S. Feng, and Y. Qu. Threshold-based level set method of image segmentation. In *2010 Third International Conference on Intelligent Networks and Intelligent Systems*, pages 703–706. IEEE, 2010.

[129] D. Xu, E. Ricci, Y. Yan, J. Song, and N. Sebe. Learning deep representations of appearance and motion for anomalous event detection. *arXiv preprint arXiv:1510.01553*, 2015.

[130] K. Yamaguchi, M. H. Kiapour, L. E. Ortiz, and T. L. Berg. Parsing clothing in fashion photographs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3570–3577. IEEE, 2012.

[131] C. Yang, Z. Yang, A. M. Khattak, L. Yang, W. Zhang, W. Gao, and M. Wang. Structured pruning of convolutional neural networks via l1 regularization. *IEEE Access*, 7:106385–106394, 2019.

[132] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[133] S. Ye, T. Zhang, K. Zhang, J. Li, K. Xu, Y. Yang, F. Yu, J. Tang, M. Fardad, S. Liu, et al. Progressive weight pruning of deep neural networks using admm. *In CVPR*, 2018.

[134] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[135] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[136] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[137] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 472–480, 2017.

[138] J. Yu and T. Huang. Autoslim: Towards one-shot architecture search for channel numbers. *In. ICLR*, 2019.

[139] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[140] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *CVPR*, 2017.

[141] L. Zheng, G. Li, and Y. Bao. Improvement of grayscale image 2d maximum entropy threshold segmentation method. In *2010 International Conference on Logistics Systems and Intelligent Management (ICLSIM)*, volume 1, pages 324–328. IEEE, 2010.

[142] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018.

[143] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[144] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *In: Proc. ICLR*, 2017.

[145] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.