EFFICIENT DEEP NETWORKS FOR REAL-WORLD INTERACTION

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Abhishek Chaurasia

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Eugenio Culurciello, Chair

Weldon School of Biomedical Engineering

Dr. Fang Huang

Weldon School of Biomedical Engineering

Dr. Felix Xiaozhu Lin

School of Electrical and Computer Engineering

Dr. Zhongming Liu

Weldon School of Biomedical Engineering

Approved by:

Dr. Dimitrios Peroulis

Head of the School Graduate Program

To my parents and my sisters who made me the person I am today.

ACKNOWLEDGMENTS

I was once told that there are three main things which any student has to deal with during his/her Ph.D.: research interest, advisor, and financial support. Most of the time a Ph.D. student gets to choose only two out of these three things. The sad fact is, I was having a hard time finding even one out of these three during my first year until I met my advisor Dr. Eugenio Culurciello.

I would like to thank him for giving me a new family (e-Lab) far away from my home, for giving me so much freedom in my research, for helping me in my overall personality development and for advises related to my personal life. Sometimes he took those extra steps for me which most of the advisors would not even have thought about. I am grateful to Dr. Fang Huang for allowing me to work with him and for all those invaluable suggestions which he gave me during our meetings. I would also like to thank Dr. Felix Lin, and Dr. Zhongming Liu to serve on my advisory committee.

I need to mention here the never ending help and support extended to me by my e-Lab and FWDNXT members: Alfredo, Ali, Andre, Ayse, Dawood, Jonghoon, Juan, Lukasz, Michael, Thomas, and Vinayak. Thank you Kaushal, Nabheen, Rohit, Sagar and Shraddha for making these five years fun and memorable. Finally, I would like to thank my parents for supporting me in every way they possibly could and Gaurav Chatterjee and my sisters for standing next to me, believing in me and bringing me where I am today.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Abhishek Chaurasia Ph.D., Purdue University, August 2019. Efficient Deep Networks for Real-world Interaction. Major Professor: Eugenio Culurciello.

Deep neural networks are essential in applications such as image categorization, natural language processing, autonomous driving, home automation, and robotics. Most of these applications require instantaneous processing of data and decision making. In general, existing neural networks are computationally expensive, and hence they fail to perform in real-time. Models performing semantic segmentation are being extensively used in self-driving vehicles. Autonomous vehicles not only need segmented output, but also control system capable of processing segmented output and deciding actuator outputs such as speed and direction.

In this thesis we propose *efficient* neural network architectures with fewer operations and parameters as compared to current state-of-the-art algorithms. Our work mainly focuses on designing deep neural network architectures for semantic segmentation. First, we introduce few network modules and concepts which help in reducing model complexity. Later on, we show that in terms of accuracy our proposed networks perform better or at least at par with state-of-the-art neural networks. Apart from that, we also compare our networks' performance on edge devices such as Nvidia TX1. Lastly, we present a control system capable of predicting steering angle and speed of a vehicle based on the neural network output.

1. INTRODUCTION

There is no doubt that automated devices and robots can serve as an effective tool in taking mankind towards a better future. Not only they can help us in tasks which we can do, but also help us with things that might be impossible/dangerous for us, such as scouting vastness of universe or a region of natural disaster. Artificial Intelligence (AI) is the field that can be broadly related to it. AI tries to emulate human perception and decision making using sensory inputs and computer systems respectively. One section of AI: Machine Learning (ML), deals with a more focused task, where machines look into data and train themselves for specific tasks. ML algorithms assume that any phenomena occurring in this world can be represented using mathematical models and over the years researchers have come up with several such models for different applications. These models comprise one or combination of man-made filters [1, 2] or filters that are learned by the machines all by themselves during training phase [3–5]. The latter approach generally uses an objective function and tries to minimize that objective function based on the training data. One of the algorithms that fall under this type of ML is Neural network.

Neural networks (NNs) were inspired by the way animal brain processes and infers any given sensory inputs. Same as the nervous system of any species, the functional and structural unit of NN is called a neuron. Even though early artificial neurons were designed to mimic biological neurons and NN models were formulated using them in the 1940s [6, 7], there are still a lot of differences between them. They differ at the fundamental level itself: creation and destruction of connection between two neurons. Moreover, the limited knowledge of the working of a biological neuron makes it difficult to create a mathematical model for it and mimic it using an artificial neuron. Although the mathematical formulation of NN started in the 1940s; effective training of neural net became possible only after the release of backpropagation algo-

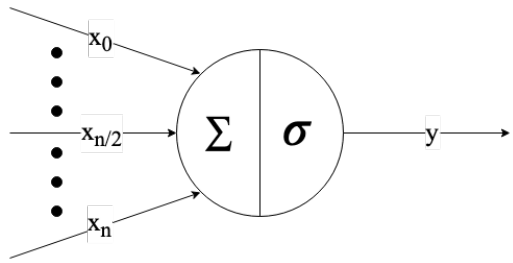

Fig. 1.1. Artificial neuron: x_i being inputs, σ can be any non-linearity and y is the output.

rithms [8, 9]. This finally led to the actual implementation of NN to perform specific tasks such as classification [5] and segmentation [10].

Generally, an artificial neuron (Figure 1.1) comprises of inputs (x_0 to x_n), non-linear activation function (σ), and an output (y). Figure 1.2 shows a simple neural network architecture formed by grouping several artificial neurons. This neural network has one input layer (L_0), one hidden layer (L_1) and one output layer (L_2). It is possible to simulate simple AND, OR gates using a neural net without any hidden layer. But the implementation of just a bit more complicated gate like XOR needs at least one hidden layer. As the tasks get more and more complicated, there is an exponential increase in the number of layers and neurons. This results in a need for massive memory and computational power which was not available a decade ago. That was one of the reasons why NN remained dormant for so many years until recently.

Advancement in machines with the ability to perform computationally intensive tasks has enabled researchers to tap deeper into neural networks. [11] successfully trained a deep neural network on multiple GPUs on ImageNet dataset [12]. NNs are data hungry and as discussed earlier, require a lot of computational power. Both, ImageNet dataset and GPUs proved to be a perfect setup to give a boost to deep neural networks (DNNs). In recent years, the availability of larger datasets and computationally-powerful machines have helped deep convolutional neural networks (CNNs) [5, 11, 13, 14] surpass the performance of many conventional computer vision

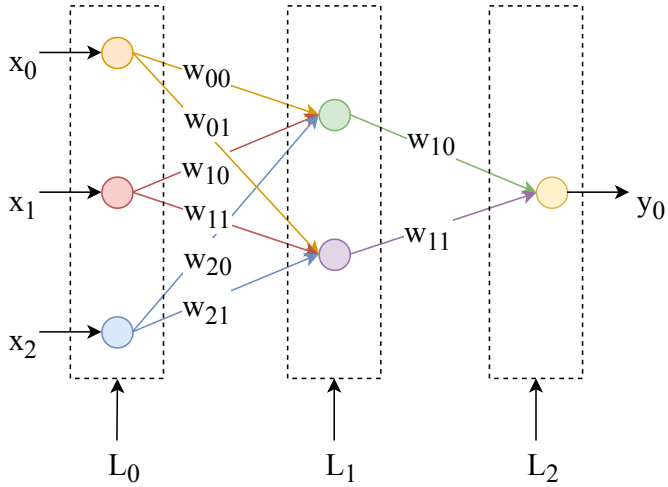

Fig. 1.2. Fully connected neural network architecture. This example has one input (L_0), one output (L_2), and one hidden L_1 layer. Each neuron of a fully connected/linear layer is linked with all the neurons of next layer.

algorithms [15–17]. In 2015, DNN even surpassed human level accuracy performance on Imagenet dataset [18]. CNNs [5,19] recent success has been demonstrated in image classification [11, 14, 20–23], localization [24, 25], scene understanding [26, 27] etc.

1.1 Neural network layers/modules

Different arrangement/organization of neurons lead to different architectures of neural network. Some of these important arrangements are discussed below. Any neural network can have a combination of any of these layers/modules.

1.1.1 Linear/Fully-connected layer

One of the very basic NN architecture is shown in Figure 1.2. The important thing to notice here is that, each neuron is connected with all the neurons of the subsequent layer, hence they are called fully-connected or linear layers. If we look at

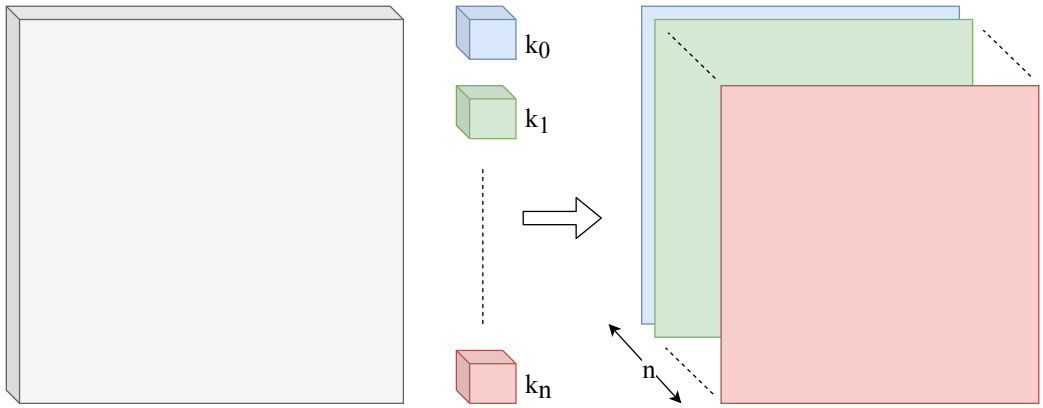

Fig. 1.3. Convolution layer with n different kernels. In a vanilla convolution layer, depth of each kernel is the same as number of input feature maps and number of output maps is equal to the number of different kernels used.

the two neurons in layer L_1 output of ith neuron can be calculated using the following formula:

$$z_{i1} = \sigma\left(\sum_{j=0}^{2} w_{ij} \times x_j\right) \tag{1.1}$$

The weight w_{ij} associated with each connection is known as parameter. A neural network with more parameters means it has more freedom to learn different filters and hence it means a stronger network. On one hand, fully-connected layers have more capacity to learn features, but on the other hand, they need a lot more data to learn those features. Apart from that, each connection leads to one extra multiplication and addition operation. So, not only this type of layer needs more data to learn, but also needs more computation power.

1.1.2 Convolution layer

Neighboring pixels of an image or successive samples of speech data are highly correlated. This correlation is exploited in the convolution layer which takes care of some of the issues which are present in a linear layer. Unlike a linear layer, only neighboring neurons of one layer are connected to neurons of the next convolution

layer. For example, in Figure 1.2, only x_0 and x_1 of L_0 will be connected to first neuron and x_1 and x_2 will be connected to second neuron of layer L_1. This results in a reduced number of connections, leading to fewer parameters and less computation. Convolution layers also use shared weights, which further reduces the number of parameters and computation. Figure 1.3 shows how convolution layer and shared weights work. In this figure, there is a 2-D input with M number of channels. Each kernel k_i moves across this input and performs convolution. As can be seen, the same k_i traces the whole input and gives one output feature map. All the output neurons of each output map were generated using the same kernel, hence sharing the same weight. Finally, n different kernels contribute to n different feature maps.

Number of parameters and operations

Two very important terms which have been talked about in the preceding sections are:

- parameters of a neural network
- number of operations required to perform for one forward pass of neural network

In Figure 1.3, lets assume input, output and kernel size is $M \times H \times W$, $N \times H \times W$ and $M \times kH \times kW$ respectively. The number of parameters is the total number of neurons present in all kernels, which is the total kernel size and is often related to model capacity.

$$\therefore \text{number of parameters} = N \times M \times kH \times kW \tag{1.2}$$

Therefore, people often think that increasing number of parameters will lead to better performance of neural network which is not the whole truth. Training a network with a lot of parameters for a simple task on a small dataset overfits the model on the training set. Also, two networks with the same number of parameters but different architecture can have different performance on the same dataset. The network architecture should

be "wisely" designed so that information learned by each parameter (information density of the network) is high.

Meanwhile, increasing the number of operations gives rise to its own set of problems. The amount of computational resources available should always be kept in mind while designing network architecture. Real-time applications which are supposed to run on the edge need a network with less number of operations as compared to the applications which can run a neural network on servers. Let us consider the same example used for calculation of the number of parameters. Each convolution operation using one kernel generates one output neuron. This roughly takes $2 \times M \times kH \times kW$ number of multiplications and additions. Each kernel has to traverse through the whole input in order to generate $N \times H \times W$ output neurons.

$$\begin{aligned} \text{Ops. to generate one output map} &= 2 \times M \times kH \times kW \times H \times W \\ \therefore \text{Number of ops. for } N \text{ output maps} &= N \times 2 \times M \times kH \times kW \times H \times W \end{aligned} \tag{1.3}$$

As we can see number of parameters is only related to kernel size, but number of operations also depends on input and output size of each layer. Therefore, increasing number of parameters do increase number of operations but it is not a linear relation.

Receptive field

The amount of input neurons one particular neuron can look at is termed as the receptive field of that neuron. In case of Figure 1.2, one neuron of L_1 is connected with three input neurons. Hence, we can say that the receptive field of each neuron of L_1 is three. On the contrary, one neuron of L_2 is connected with two neurons of L_1. That is why, it's receptive field with respect to L_1 is two, but it's receptive field with respect to L_0 is three.

Figure 1.4 demonstrates receptive field of three layered convolutional network. Similar to the previous example, the final output neuron is looking at a section of 3×3 of L_1. One neuron of L_1 is looking at 3×3 inputs of L_0. Therefore, receptive field of one neuron of L_2 with respect to the input or L_0 is 5×5.

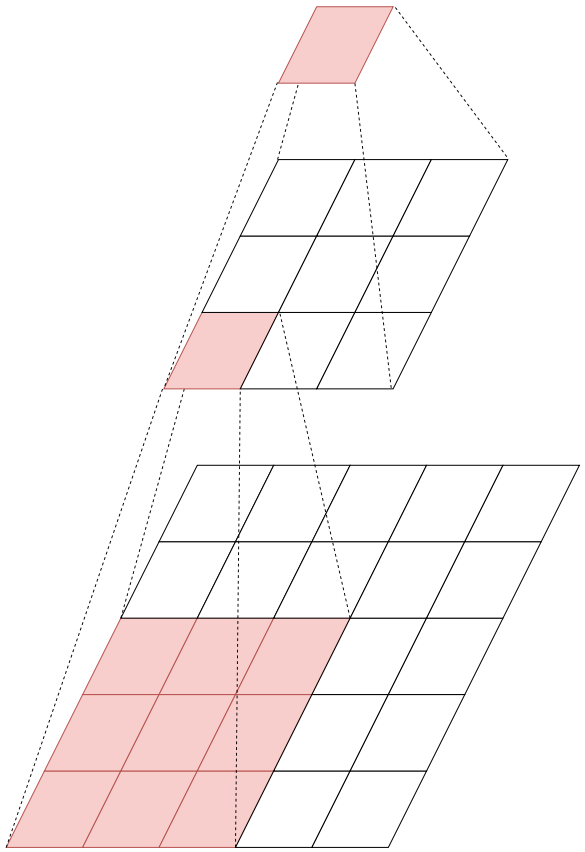

Fig. 1.4. Receptive field of a two layer convolutional neural network. Kernel size of both convolution operation is 3×3. Input size is 5×5 and output size is 1×1. Receptive field of output neuron w.r.t. input neurons is five.

1.1.3 Residual module

As the networks become deeper, it becomes more difficult to train them. One major reason behind this is the problem of vanishing gradient. During the training and backpropagation step, gradient values in general, tend to decrease as they make their way towards the first layer from the last layer. In the case of deep networks, more than often these gradients become almost zero for the layers close to input. As a result, the weights of these layers do not get updated. [22] introduced the concept of residual networks to counter this problem. Figure 1.5.a contains a neural network layer representing a non-linear function $f(x)$ alongside a bypass/residual connection.

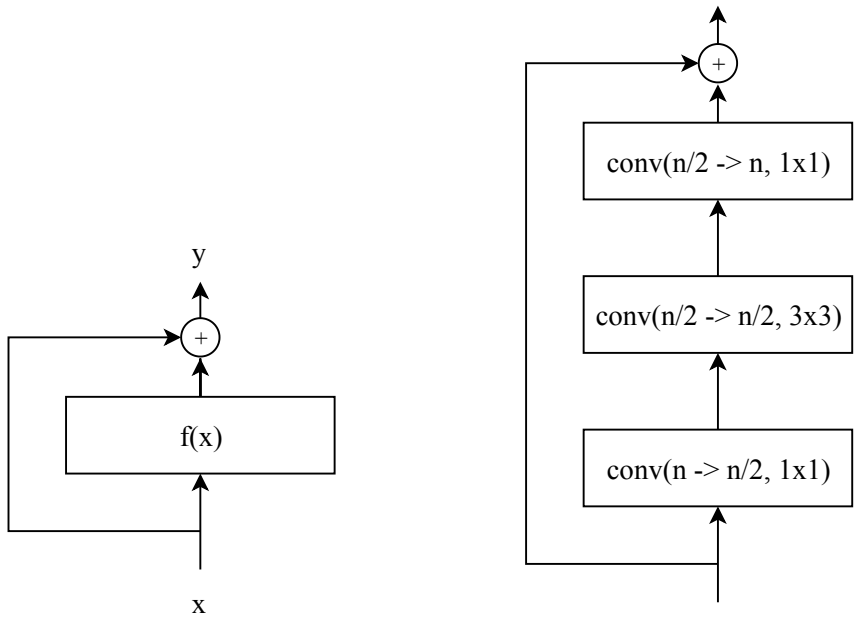

Fig. 1.5. (a) Residual module (b) Residual module with bottleneck architecture [18].

This residual connection gives gradients alternate path when required, thus enabling deeper networks to learn rich features, which otherwise would not have been possible. Moreover, it also allows NNs to mimic identity operation. In equation 1.4, if the network feels that layer with $f(x)$ is not required then it will set $f(x)$ to zero and effectively use the residual path for forward pass. It has been empirically found that it is difficult for NNs to learn identity operation using a non-linear function, which without the bypass connection would have been the case.

$$y = x + f(x) \tag{1.4}$$

$f(x)$ can be one layer or stack of layers as shown in Figure 1.5.b.

The type of arrangement shown in Figure 1.5.b is known as bottleneck architecture. The first layer in this module squeezes feature information from n to $n/2$ using 1×1 convolution and then performs 3×3 convolution on this squeezed input. The final layer of this module brings the output back to the original (as in this example to n) or higher number of feature maps. It has been shown that this squeezing and un-

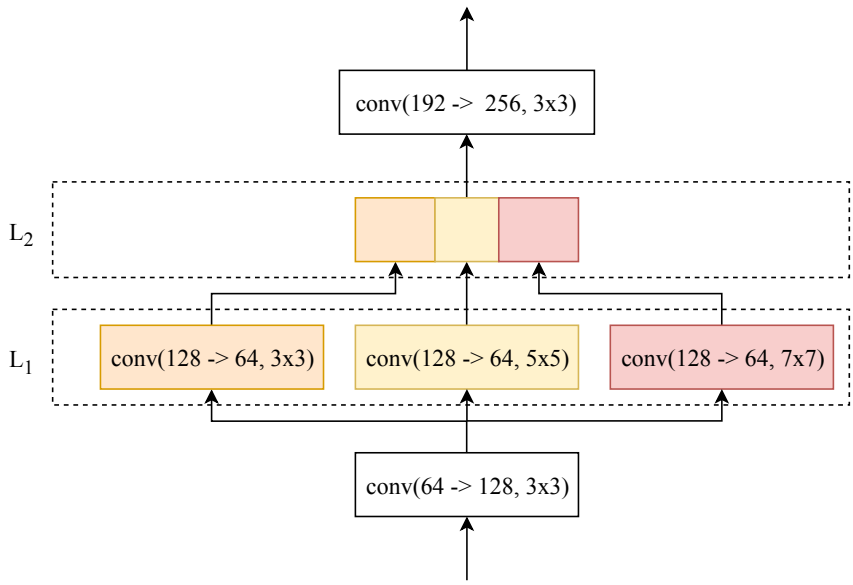

Fig. 1.6. Inception module [29]: parallel convolutional layers in L_1 allow the network to work on same input at different scale by using different kernel sizes (3×3, 5×5, and 7×7). Output of L_1 is then concatenated in L_2 along the feature dimension.

squeezing of feature maps helps neural networks in learning better features [18,22,28]. Since 3×3 kernel is used to perform convolution on smaller input, this inherently helps in reducing the number of operations and parameters of a neural network. Using equations 1.2 and 1.3 we can make a numerical comparison between Figure 1.5.a, where $f(x)$ is just one convolution layer with 3×3 kernel size and n input and output channels (without bottleneck) and Figure 1.5.b (with bottleneck).

$$
\begin{aligned}
\text{number of parameters with bottleneck} &= 9N \\
\text{number of parameters without bottleneck} &= 2 \times N + 9 \times \frac{N}{2} \\
&= \frac{13}{2}N \\
\text{number of operations with bottleneck} &= 18HWN^2 \\
\text{number of operations without bottleneck} &= \frac{18}{4}HWN^2 + \frac{4}{2}HWN^2 \\
&= \frac{13}{2}HWN^2
\end{aligned}
$$

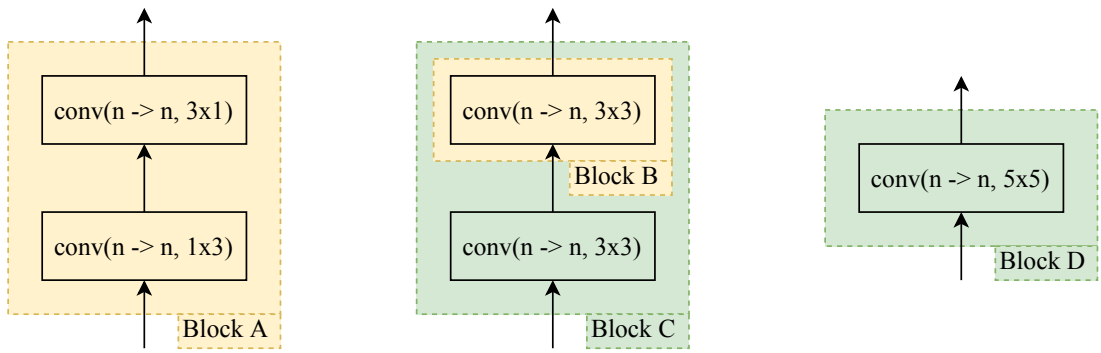

Fig. 1.7. Comparison of receptive field of convolution layer with different kernel size. *Block A* has the same receptive field as *Block B*, and *Block C* has the same receptive field as *Block D*.

1.1.4 Inception module

Neural networks are not inherently scale invariant but carefully designing the architecture can make them scale invariant. One of the ways to make the network scale invariant is using inception module [23, 29]. Figure 1.6 shows a basic form of inception module with three parallel convolutional layers in L_1 with three different kernel sizes. Since the kernel sizes are different; each parallel layer has a different receptive field. This implies that each parallel layer is capable of dealing with different scales. Inception modules can also have normal bypass connections with no layers (same as residual blocks) or with a kernel size of 1×1.

One more thing to note here is that L_2 is concatenation whereas in the residual module it is an addition. To keep the number of operations in check, number of output feature maps of L_1 is generally kept low so that when they get concatenated in L_2, this does not result in a very wide network. Moreover, it is possible to have the same receptive field as 5×5 using two layers of convolution with a kernel size of 3×3 each. This can be seen in Figure 1.7 where receptive field of *Block D* is the same as *Block C*, and receptive field of *Block B* is the same as *Block A*. Using this concept, we can have a lot of variants of the inception module. *Block D* present in Figure 1.6 can be replaced with *Block C* or two *Block A*s. Similarly, 7×7 convolution can be

replaced by three layers of convolution with 3×3 kernel. Doing so we not only keep the receptive field the same, but we also increase non-linearity in our network. Apart from that, the number of operations and parameters also drop by a lot.

$$\text{number of parameters in Block A} = 6n$$

$$\text{number of parameters in Block B} = 9n$$

$$\text{number of operations in Block A} = 12HWn^2$$

$$\text{number of operations in Block B} = 18HWn^2$$

$$\text{number of parameters in Block C} = 18n$$

$$\text{number of parameters in Block D} = 25n$$

$$\text{number of operations in Block C} = 36HWn^2$$

$$\text{number of operations in Block D} = 50HWn^2$$

1.1.5 Separable convolution

Figure 1.8 shows an example of spatially separable convolution [23]. It is similar to the operation in *Block A* of Figure 1.7. Spatial separable convolution uses asymmetric kernels and tries to reduce the number of operations while keeping the receptive field the same. In classical computer vision terms, a kernel is called to be separable if it can be split into two separate kernels and still give the same result. One such example

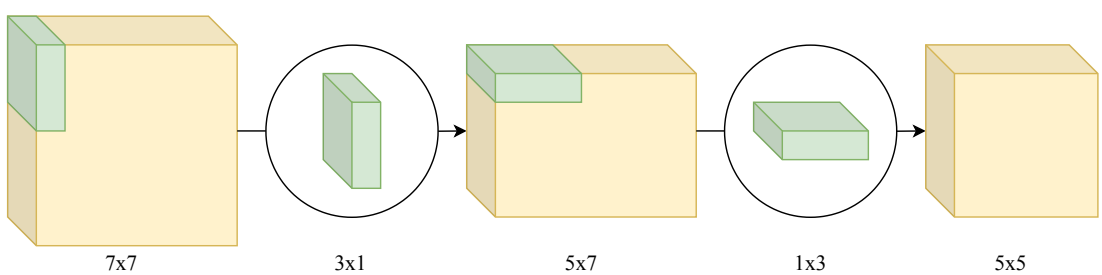

Fig. 1.8. Spatially separable convolution [23]: each kernel looks into the input information along only one spatial dimension.

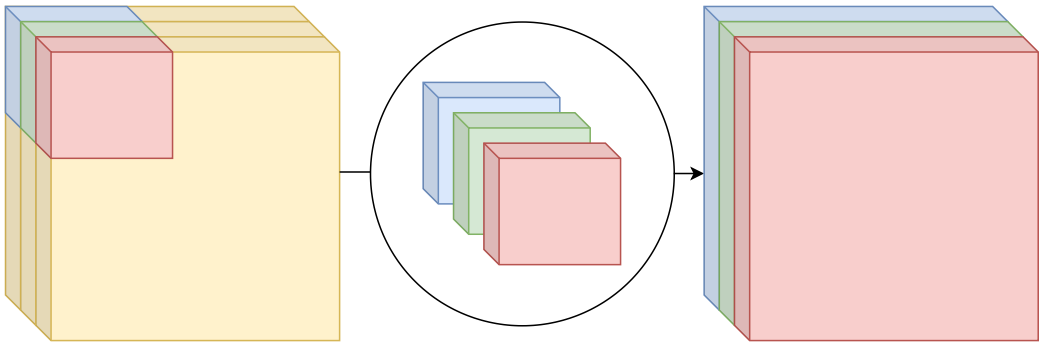

Fig. 1.9. Depth-wise separable convolution [32]: unlike vanilla convolution, depth of each kernel does not need to be of the same size as the number of input channels. Each kernel looks at only one or a group of input feature maps.

is a Sobel kernel [30] which acts as an edge detector. But in case of deep learning, the term "separable convolution" is loosely used, and splitting a convolution layer of 3×3 into two convolution layers of 1×3 and 3×1 is called as spatially separable convolution even though they might not give same results. In this type of convolution, kernel size is modified only in the spatial dimension, meaning a 3-D volume kernel is split into two 2-D volume kernels (Figure 1.8). It has been shown that further reduction in dimension is possible using *flattened convolution* where a 3-D kernel can be broken into 1-D kernels [31].

Another form of separable convolution is depth-wise separable convolution and it was introduced as part of *Xception* network architecture [32]. Instead of 3-D volume kernels, in depth-wise separable convolution, only 2-D kernels are used in Figure 1.9. When input feature maps of a layer are independent of each other; each map can be processed by a separate kernel. One assumption here is that the input feature maps are orthogonal to each other. To make sure that this assumption is true, most of the time depth-wise separable convolution is preceded with a 1×1 point-wise convolution layer. One benefit of using depth-wise convolution is the reduction in the number of parameters and operations. Consider an example where we want to use

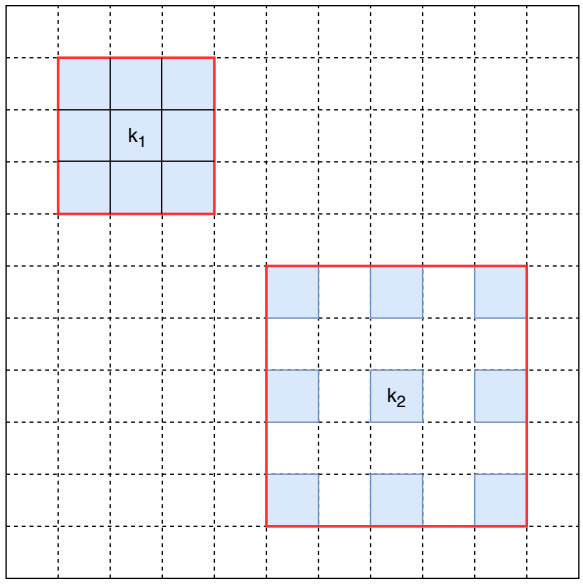

Fig. 1.10. Dilated convolution [33]: 3×3 convolution without dilation (k_1), and with dilation of 1 (k_2).

normal convolution with 3×3 kernel going from N to N feature maps with a spatial size of input and output being $H \times W$. From equation 1.2 and 1.3, we can say that:

$$\text{number of parameters} = 9N^2$$

$$\text{number of operations} = 18HWN^2$$

Now if we replace this layer with 1×1 point-wise convolution and 3×3 depth-wise separable convolution,

$$\text{number of parameters} = N^2 + 9N$$

$$\text{number of operations} = 2HWN^2 + 18HWN$$

When N is very large, we can say that the number of parameters and operations were reduced by a factor of $kH \times kW$, here 9.

1.1.6 Dilated convolution

One way to have a bigger receptive field is to have a bigger kernel; which means a lot of computation. Apart from the architectural choices discussed before, another

approach is to use dilated convolution [33]. Neighboring pixels of an image, in general, are highly correlated. Therefore, instead of using 3×3 kernel with an arrangement as k_1 of Figure 1.10, we can skip one pixel and have an arrangement as k_2. The latter has a receptive field of 5×5 kernel but complexity of 3×3 kernel. In this example, k_1 is a normal convolutional kernel or a kernel with zero dilation and k_2 has a dilation of 1. Another obvious alternative which we can think of is getting rid of the redundancy/correlation in the pixels by downsampling them. Downsampling comes with a cost of information loss, whereas in case of dilated convolution, the kernel can go through the immediate next pixel which was skipped in the previous stride. Therefore, there is no actual loss of information when using dilated convolution.

1.2 Motivation

Even though convolutional neural networks (CNNs) are being extensively used for categorization, natural language processing, style-transfer, etc.; scene understanding is one important area that has gained a lot of attention. This shift in focus has been mostly because of the advancement in home-automation, wearable technology (headsets for virtual/augmented reality), and self-driving vehicles. One important aspect of scene understanding is pixel-level classification/semantic segmentation [34, 35]. Similar to a classification network where our goal is to identify what is there in one image; in semantic segmentation, the goal is to classify each and every pixel. As a result, the input and output resolution of segmentation networks need to be similar. Since segmentation is about extracting precise and detailed information from an input, most of the time the provided input image is of high resolution. On one hand, categorization networks are trained on images with resolution generally raging between 200×200 to 300×300, on the other hand, the input of segmentation networks generally range between 640×360 (nHD) to 7680×4320 (8K UHD). From equation 1.3, we know that the number of operations is proportional to input and

output size. Therefore, segmentation networks are one of the most computationally expensive networks.

Even though semantic segmentation targets applications that require real-time operation, ironically most of the current deep networks require excessively large processing time. Networks such as YOLO [36], Fast RCNN [37], SSD [38] focus on real-time object detection but there is very little to no work done in this direction in case of semantic segmentation [39]. Inspired by auto-encoders [20, 40], most of the existing techniques for semantic segmentation use an encoder-decoder pair as the core of their network architecture. Here the encoder encodes information into feature space (discriminator), and the decoder maps this information into spatial categorization (generator) to perform segmentation. State-of-the-art segmentation networks, use categorization models which are winners of ImageNet Large Scale Visual Recognition Challenge (ILSCRC) as their discriminator. The generator either uses the stored pooling indices from discriminator or learns the parameters using convolution to perform upsampling [41, 42]. Parameterized upsampling is done by either using simple interpolation followed by convolution or by using transpose convolution [43,44]. Moreover, encoder and decoder can be either symmetric (same number of layers in encoder and decoder with the same number of pooling and unpooling layers), or they can be asymmetric.

In [41] a pre-trained VGG16 [21] was used as the discriminator. Pooling indices after every max-pooling step was saved and then later used for upsampling in the decoder. Later on, researchers came up with the idea of deep deconvolution network [42, 43], a fully convolutional network (FCN) combined with skip architecture [44], which eliminated the need of saving pooling indices. Networks designed for classification and categorization mostly use a fully-connected layer as their classifier; in FCN they were replaced with convolutional layers. Standard pre-trained encoders such as: AlexNet [11], VGG [21], and GoogLeNet [14] have been used for segmentation. To get precise segmentation boundaries, researchers have also tried to cascade their deep convolutional neural network (DCNN) with post-processing steps, like the

use of Conditional Random Field (CRF) [26,45]. Instead of using networks that were designed to perform image classification, [33] proposed to use networks specifically designed for dense predictions. Pooling or downsampling using strided convolution increases the receptive field but lowers/loses spatial information. To overcome this, dilated/atrous convolution was used in [33,45]. They used fewer downsampling layers but still had similar receptive field and complexity because of dilated convolution. Apart from this, recently recurrent neural networks (RNNs) were used to get contextual information [46] and to optimize CRF [47]; but the use of RNN in itself makes it computationally expensive.

However, these networks are slow during inference due to their large architectures and numerous parameters. Table 1.1 lists out the computational complexity of some of the existing networks. Here downsampling factor 1 indicates default image resolution was used, whereas factor of 4 means the input image size used for training was 512×256. It is obvious that most of the networks will not even fit in one Nvidia graphics processor unit (GPU) with 12 GB RAM and they will not run on edge devices which have even fewer compute power and memory. There are several researchers, working on designing neural network hardware accelerators [**cite**]. They are mostly focused on inference on the edge. We compare our designed networks' performance on Nvidia TX1 which is being widely used in self-driving vehicles and we also present results achieved on Inference Engine [48,49] which is an field programmable gate array (FPGA) based neural network hardware accelerator.

In our work, we have attempted to get accurate instance level prediction without compromising the processing time of the network. Generally, spatial information is lost in the encoder due to pooling or strided convolution is recovered by using the pooling indices or by full convolution. We hypothesize and later prove that instead of the above techniques; bypassing spatial information, directly from the encoder to the corresponding decoder improves accuracy along with a significant decrease in processing time. We also show a few other modules which help in reducing model complexity while showing minimal to no effect on network accuracy.

Table 1.1.
State-of-the-art deep neural networks trained on cityscapes dataset [50] with default input image resolution of 2048×1024.

Model	Downsample factor	Parameters (million)	Operations (GOps)
SegNet [42]	4	39.8	315.1
FCN 8s [33]	1	134.5	2685.1
PSPNet [51]	1	68.1	4344.7
DeepLab v3+ [52]	1	59.3	1419.5
FRRN B [53]	2	23.9	915.5

As discussed earlier, the proposed networks can be used in self-driving vehicles. The segmented outputs of these networks are not sufficient in itself. Another vital component of autonomous vehicles is the driving control system. This control system processes this segmented output and predicts the desirable actuator output. In this thesis, we also design a control system capable of deciding the steering angle and speed of the vehicle based on the lane marking segmentation mask generated by our segmentation network.

Major contributions of this thesis are:

- designing an efficient neural network architecture using existing modules,
- using novel approach to share encoder information with decoder,
- finding lane markings using proposed segmentation network, and
- development of lane following assistant.

2. EFFICIENT DEEP NEURAL NETWORK ARCHITECTURE FOR REAL-TIME SEMANTIC SEGMENTATION

The ability to perform pixel-wise semantic segmentation in real-time is of paramount importance in practical mobile applications. Recent deep neural networks aimed at this task have the disadvantage of requiring a large number of floating point operations and have long run-times that hinder their usability. In this chapter, we propose a novel deep neural network architecture, created specifically for tasks requiring low latency operation. Our model is up to 18× faster, requires 75× fewer FLOPs, has 79× fewer parameters, and provides similar or better accuracy to existing models. We have tested it on CamVid, Cityscapes and SUN datasets and report on comparisons with existing state-of-the-art methods, and the trade-offs between accuracy and processing time of a network. We also present performance measurements of the proposed architecture on embedded systems.

2.1 Network architecture

The architecture of our network is presented in Table 2.1. It is divided into several stages, as highlighted by horizontal lines in the table and the first digit after each block name. Output sizes are reported for an example input image resolution of 512×512. We adopt a view of ResNets [22] that describes them as having a single main branch and extensions with convolutional filters that separate from it, and then merge back with an element-wise addition, as shown in Figure 2.1(b). Each block consists of three convolutional layers: a 1×1 projection that reduces the dimensionality, a main convolutional layer (`conv` in Figure 2.1(b)), and a 1×1 expansion. We place Batch Normalization [54] and PReLU [18] between all convolutions. Just as in the original

paper, we refer to these as bottleneck modules. If the bottleneck is downsampling, a max pooling layer is added to the main branch.

Table 2.1.
Proposed architecture. Output sizes are given for an example input of 512×512.

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
4× bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

Also, the first 1×1 projection is replaced with a 2×2 convolution with stride 2 in both dimensions. We zero pad the activations, to match the number of feature maps. `conv` is either a regular, dilated or full convolution (also known as deconvolution or fractionally strided convolution) with 3×3 filters. Sometimes we replace it with asymmetric convolution i.e. a sequence of 5×1 and 1×5 convolutions. For the regularizer, we use Spatial Dropout [25], with $p = 0.01$ before bottleneck2.0, and $p = 0.1$ afterwards.

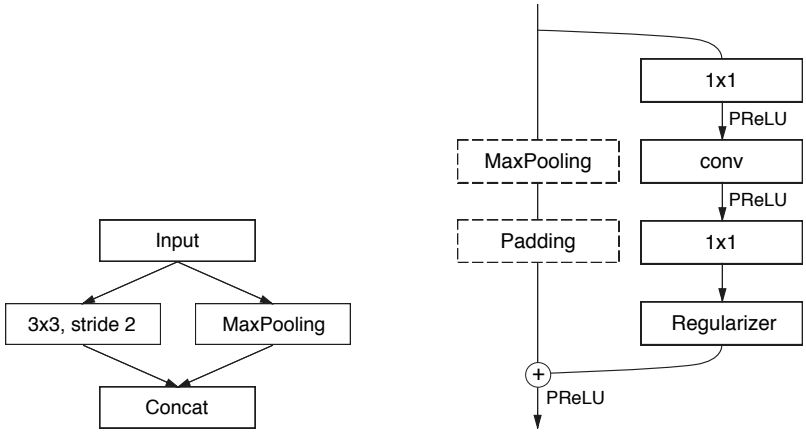

Fig. 2.1. (a) Initial block. MaxPooling is performed with non-overlapping 2×2 windows, and the convolution has 13 filters, which sums up to 16 feature maps after concatenation. This is heavily inspired by [29]. (b) Bottleneck module. `conv` is either a regular, dilated, or full convolution (also known as deconvolution) with 3×3 filters, or a 5×5 convolution decomposed into two asymmetric ones.

The initial stage contains a single block, that is presented in Figure 2.1(a). Stage 1 consists of 5 bottleneck blocks, while stage 2 and 3 have the same structure, with the exception that stage 3 does not downsample the input at the beginning (we omit the 0th bottleneck). These three first stages are the encoder. Stage 4 and 5 belong to the decoder.

We did not use bias terms in any of the projections, in order to reduce the number of kernel calls and overall memory operations, as cuDNN [55] uses separate kernels for convolution and bias addition. This choice didn't have any impact on the accu-

racy. Between each convolutional layer and the following non-linearity we use Batch Normalization [54]. In the decoder, max pooling is replaced with max unpooling, and padding is replaced with spatial convolution without bias. We did not use pooling indices in the last upsampling module, because the initial block operated on the 3 channels of the input frame, while the final output has C feature maps (the number of object classes). Also, for performance reasons, we decided to place only a bare full convolution as the last module of the network, which alone takes up a sizeable portion of the decoder processing time.

2.2 Design choices

In this section we will discuss our most important experimental results and intuitions, that have shaped the final architecture of our model.

Feature map resolution

Downsampling images during semantic segmentation has two main drawbacks. Firstly, reducing feature map resolution implies loss of spatial information like exact edge shape. Secondly, full pixel segmentation requires that the output has the same resolution as the input. This implies that strong downsampling will require equally strong upsampling, which increases model size and computational cost. The first issue has been addressed in FCN [44] by adding the feature maps produced by encoder, and in SegNet [41] by saving indices of elements chosen in max pooling layers, and using them to produce sparse upsampled maps in the decoder. We followed the SegNet approach, because it allows to reduce memory requirements. Still, we have found that strong downsampling hurts the accuracy, and tried to limit it as much as possible.

However, downsampling has one big advantage. Filters operating on downsampled images have a bigger receptive field, that allows them to gather more context. This is especially important when trying to differentiate between classes like, for example,

rider and pedestrian in a road scene. It is not enough that the network learns how people look, the context in which they appear is equally important. In the end, we have found that it is better to use dilated convolutions for this purpose [33].

Early downsampling

One crucial intuition to achieving good performance and real-time operation is realizing that processing large input frames is very expensive. This might sound very obvious, however many popular architectures do not to pay much attention to optimization of early stages of the network, which are often the most expensive by far.

The first two blocks our our network heavily reduce the input size, and use only a small set of feature maps. The idea behind it, is that visual information is highly spatially redundant, and thus can be compressed into a more efficient representation. Also, our intuition is that the initial network layers should not directly contribute to classification. Instead, they should rather act as good feature extractors and only preprocess the input for later portions of the network. This insight worked well in our experiments; increasing the number of feature maps from 16 to 32 did not improve accuracy on Cityscapes [50] dataset.

Decoder size

In this work we would like to provide a different view on encoder-decoder architectures than the one presented in [42]. SegNet is a very symmetric architecture, as the encoder is an exact mirror of the encoder. Instead, our architecture consists of a large encoder, and a small decoder. This is motivated by the idea that the encoder should be able to work in a similar fashion to original classification architectures, i.e. to operate on smaller resolution data and provide for information processing and filtering. Instead, the role of the the decoder, is to upsample the output of the encoder, only fine-tuning the details.

Nonlinear operations

A recent paper [28] reports that it is beneficial to use ReLU and Batch Normalization layers before convolutions. We tried incorporating these ideas, but they had a detrimental effect on accuracy. Instead, we have found that removing most ReLUs in the initial layers of the network improved the results. It was quite a surprising finding so we decided to investigate its cause.

We replaced all ReLUs in the network with PReLUs [18], which use an additional parameter per feature map, with the goal of learning the negative slope of nonlinearities. We expected that in layers where identity is a preferable transfer function, PReLU weights will have values close to 1, and conversely, values around 0 if ReLU is preferable. Results of this experiment can be seen in Figure 2.2.

Initial layers weights exhibit a large variance and are slightly biased towards positive values, while in the later portions of the encoder they settle to a recurring pattern. All layers in the main branch behave nearly exactly like regular ReLUs, while the weights inside bottleneck modules are negative i.e. the function inverts and scales down negative values. We hypothesize that identity did not work well in our architecture because of its limited depth. The reason why such lossy functions are learned might be that that the original ResNets [28] are networks that can be hundreds of layers deep, while our network uses only a couple of layers, and it needs to quickly filter out information. It is notable that the decoder weights become much more positive and learn functions closer to identity. This confirms our intuitions that the decoder is used only to fine-tune the upsampled output.

Information-preserving dimensionality changes

As stated earlier, it is necessary to downsample the input early, but aggressive dimensionality reduction can also hinder the information flow. A very good approach to this problem has been presented in [29]. It has been argued that a method used by the VGG architectures, i.e. as performing a pooling followed by a convolution

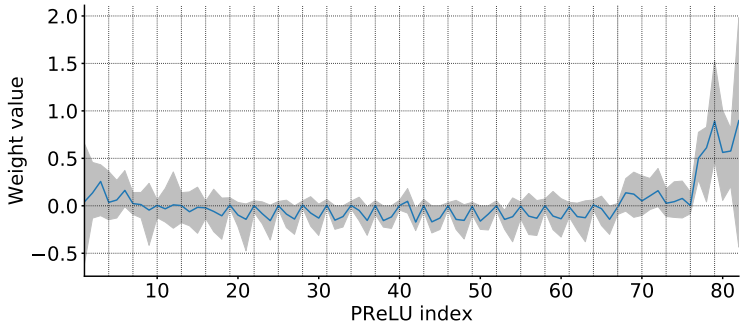

Fig. 2.2. PReLU weight distribution vs network depth. Blue line is the weights mean, while an area between maximum and minimum weight is grayed out. Each vertical dotted line corresponds to a PReLU in the main branch and marks the boundary between each of bottleneck blocks. The gray vertical line at 67th module is placed at encoder-decoder border.

expanding the dimensionality, however relatively cheap, introduces a representational bottleneck (or forces one to use a greater number of filters, which lowers computational efficiency). On the other hand, pooling after a convolution, that increases feature map depth, is computationally expensive. Therefore, as proposed in [29], we chose to perform pooling operation in parallel with a convolution of stride 2, and concatenate resulting feature maps. This technique allowed us to speed up inference time of the initial block 10 times.

Additionally, we have found one problem in the original ResNet architecture. When downsampling, the first 1×1 projection of the convolutional branch is performed with a stride of 2 in both dimensions, which effectively discards 75% of the input. Increasing the filter size to 2×2 allows to take the full input into consideration, and thus improves the information flow and accuracy. Of course, it makes these layers $4\times$ more computationally expensive, however there are so few of these in our architecture, that the overhead is unnoticeable.

Factorizing filters

It has been shown that convolutional weights have a fair amount of redundancy, and each $n \times n$ convolution can be decomposed into two smaller ones following each other: one with a $n \times 1$ filter and the other with a $1 \times n$ filter [31]. This idea has been also presented in [29], and from now on we adopt their naming convention and will refer to these as asymmetric convolutions. We have used asymmetric convolutions with $n = 5$ in our network, so cost of these two operations is similar to a single 3×3 convolution. This allowed to increase the variety of functions learned by blocks and increase the receptive field.

What's more, a sequence of operations used in the bottleneck module (projection, convolution, projection) can be seen as decomposing one large convolutional layer into a series of smaller and simpler operations, that are its low-rank approximation. Such factorization allows for large speedups, and greatly reduces the number of parameters, making them less redundant [31]. Additionally, it allows to make the functions they compute richer, thanks to the non-linear operations that are inserted between layers.

Dilated convolutions

As argued above, it is very important for the network to have a wide receptive field, so it can perform classification by taking a wider context into account. We wanted to avoid overly downsampling the feature maps, and decided to use dilated convolutions [33] to improve our model. They replaced the main convolutional layers inside several bottleneck modules in the stages that operate on the smallest resolutions. These gave a significant accuracy boost, by raising IoU on Cityscapes by around 4 percentage points, with no additional cost. We obtained the best accuracy when we interleaved them with other bottleneck modules (both regular and asymmetric), instead of arranging them in sequence, as has been done in [33].

Regularization

Most pixel-wise segmentation datasets are relatively small (on order of 10^3 images), so such expressive models as neural networks quickly begin to overfit them. In initial experiments, we used L2 weight decay with little success. Then, inspired by [56], we have tried stochastic depth, which increased accuracy. However it became apparent that dropping whole branches (i.e. setting their output to 0) is in fact a special case of applying Spatial Dropout [25], where either all of the channels, or none of them are ignored, instead of selecting a random subset. We placed Spatial Dropout at the end of convolutional branches, right before the addition, and it turned out to work much better than stochastic depth.

2.3 Results

We benchmarked the performance of our work on three different datasets to demonstrate real-time and accurate for practical applications. We tested on CamVid and Cityscapes datasets of road scenes, and SUN RGB-D dataset of indoor scenes. We set SegNet [41] as a baseline since it is one of the fastest segmentation models, that also has way fewer parameters and requires less memory to operate than FCN. All our models, training, testing and performance evaluation scripts were using the Torch7 machine-learning library, with cuDNN backend. To compare results, we use class average accuracy and intersection-over-union (IoU) metrics.

2.3.1 Performance Analysis

We report results on inference speed on widely used NVIDIA Titan X GPU as well as on NVIDIA TX1 embedded system module. This network was designed to achieve more than 10 fps on the NVIDIA TX1 board with an input image size 640×360, which is adequate for practical road scene parsing applications. For inference we

merge batch normalization and dropout layers into the convolutional filters, to speed up all networks.

Table 2.2.
Performance comparison.

Model	NVIDIA TX1						NVIDIA Titan X					
	480×320		640×360		1280×720		640×360		1280×720		1920×1080	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
SegNet	757	1.3	1251	0.8	-	-	69	14.6	289	3.5	637	1.6
Our network	47	21.1	69	14.6	262	3.8	7	135.4	21	46.8	46	21.6

Inference time

Table 2.2 compares inference time for a single input frame of varying resolution. We also report the number of frames per second that can be processed. Dashes indicate that we could not obtain a measurement, due to lack of memory. Proposed architecture is significantly faster than SegNet, providing high frame rates for real-time applications and allowing for practical use of very deep neural network models with encoder-decoder architecture.

Table 2.3.
Hardware requirements. FLOPs are estimated for an input of $3 \times 640 \times 360$.

	GFLOPs	Parameters	Model size (fp16)
SegNet	286.03	29.46M	56.2 MB
Our network	3.83	0.37M	0.7 MB

Hardware requirements

Table 2.3 reports a comparison of number of floating point operations and parameters used by different models. Our model efficiency is evident, as its requirements are on two orders of magnitude smaller. Please note that we report storage required to save model parameters in half precision floating point format. This network has so few parameters, that the required space is only 0.7MB, which makes it possible to fit the whole network in an extremely fast on-chip memory in embedded processors. Also, this alleviates the need for model compression [57], making it possible to use general purpose neural network libraries. However, if one needs to operate under incredibly strict memory constraints, these techniques can still be applied to proposed neural network as well.

Software limitations

One of the most important techniques that has allowed us to reach these levels of performance is convolutional layer factorization. However, we have found one surprising drawback. Although applying this method allowed us to greatly reduce the number of floating point operations and parameters, it also increased the number of individual kernels calls, making each of them smaller.

We have found that some of these operations can become so cheap, that the cost of GPU kernel launch starts to outweigh the cost of the actual computation. Also, because kernels do not have access to values that have been kept in registers by previous ones, they have to load all data from global memory at launch, and save it when their work is finished. This means that using a higher number of kernels, increases the number of memory transactions, because feature maps have to be constantly saved and reloaded. This becomes especially apparent in case of non-linear operations. In the proposed model, PReLUs consume more than a quarter of inference time. Since they are only simple point-wise operations and are very easy to parallelize, we hypothesize it is caused by the aforementioned data movement.

These are serious limitations, however they could be resolved by performing kernel fusion in existing software i.e. create kernels that apply non-linearities to results of convolutions directly, or perform a number of smaller convolutions in one call. This improvement in GPU libraries, such as cuDNN, could increase the speed and efficiency of our network even further.

2.3.2 Benchmarks

We have used the Adam optimization algorithm [58] to train the network. It allowed the model to converge very quickly and on every dataset we have used training took only 3-6 hours, using four Titan X GPUs. It was performed in two stages: first we trained only the encoder to categorize downsampled regions of the input image, then we appended the decoder and trained the network to perform upsampling and pixel-wise classification. Learning rate of 5e−4 and L2 weight decay of 2e−4, along with batch size of 10 consistently provided the best results. We have used a custom class weighing scheme defined as $w_{\text{class}} = \frac{1}{\ln(c+p_{\text{class}})}$. In contrast to the inverse class probability weighing, the weights are bounded as the probability approaches 0. c is an additional hyper-parameter, which we set to 1.02 (i.e. we restrict the class weights to be in the interval of $[1, 50]$).

Table 2.4.
Cityscapes test set results

Model	Class IoU	Class iIoU	Category IoU	Category iIoU
SegNet	56.1	34.2	79.8	**66.4**
Proposed model	**58.3**	**34.4**	**80.4**	64.0

Cityscapes

This dataset consists of 5000 fine-annotated images, out of which 2975 are available for training, 500 for validation, and the remaining 1525 have been selected as test set [50]. Cityscapes was the most important benchmark for us, because of its outstanding quality and highly varying road scenarios, often featuring many pedestrians and cyclists. We trained on 19 classes that have been selected in the official evaluation scripts [50]. It makes use of an additional metric called instance-level intersection over union metric (iIoU), which is IoU weighed by the average object size. As reported in Table 2.4, our network outperforms SegNet in class IoU and iIoU, as well as in category IoU. It is currently the fastest model in the Cityscapes benchmark. Example predictions for images from validation set are presented in Figure 2.3.

Table 2.5.
Results on CamVid test set of (1) SegNet-Basic, (2) SegNet, and (3) Our work

Model	Building	Tree	Sky	Car	Sign	Road	Pedestrian	Fence	Pole	Sidewalk	Bicyclist	Class avg.	Class IoU
1	75.0	84.6	91.2	**82.7**	36.9	93.3	55.0	47.5	**44.8**	74.1	16.0	62.9	n/a
2	**88.8**	**87.3**	92.4	82.1	20.5	**97.2**	57.1	49.3	27.5	84.4	30.7	65.2	**55.6**
3	74.7	77.8	**95.1**	82.4	**51.0**	95.1	**67.2**	**51.7**	35.4	**86.7**	**34.1**	**68.3**	51.3

CamVid

Another automotive dataset, on which we have tested our work, was CamVid. It contains 367 training and 233 testing images [59]. There are eleven different classes such as building, tree, sky, car, road, etc. while the twelfth class contains unlabeled data, which we ignore while training. The original frame resolution for this dataset is 960×720 but we downsampled the images to 480×360 before training. In Table 2.5 we compare the performance of proposed architecture with existing state-of-the-art

algorithms. Our network outperforms other models in six classes, which are difficult to learn because they correspond to smaller objects. Its output for example images from the test set can be found in Figure 2.4.

Table 2.6.
SUN RGB-D test set results

Model	Global avg.	Class avg.	Mean IoU
SegNet	**70.3**	**35.6**	**26.3**
Proposed network	59.5	32.6	19.7

SUN RGB-D

The SUN dataset consists of 5285 training images and 5050 testing images with 37 indoor object classes. We did not make any use of depth information in this work and trained the network only on RGB data. In Table 2.6 we compare the performance of our network with SegNet [42], which is the only neural network model that reports accuracy on this dataset. Our results, though inferior in global average accuracy and IoU, are comparable in class average accuracy. Since global average accuracy and IoU are metrics that favor correct classification of classes occupying large image patches, researchers generally emphasize the importance of other metrics in case of semantic segmentation. One notable example is introduction of iIoU metric [50]. Comparable result in class average accuracy indicates, that our network is capable of differentiating smaller objects nearly as well as SegNet. Moreover, the difference in accuracy should not overshadow the huge performance gap between these two networks. The proposed network can process the images in real-time, and is nearly 20× faster than SegNet on embedded platforms. Example predictions from SUN test set are shown in Figure 2.5.

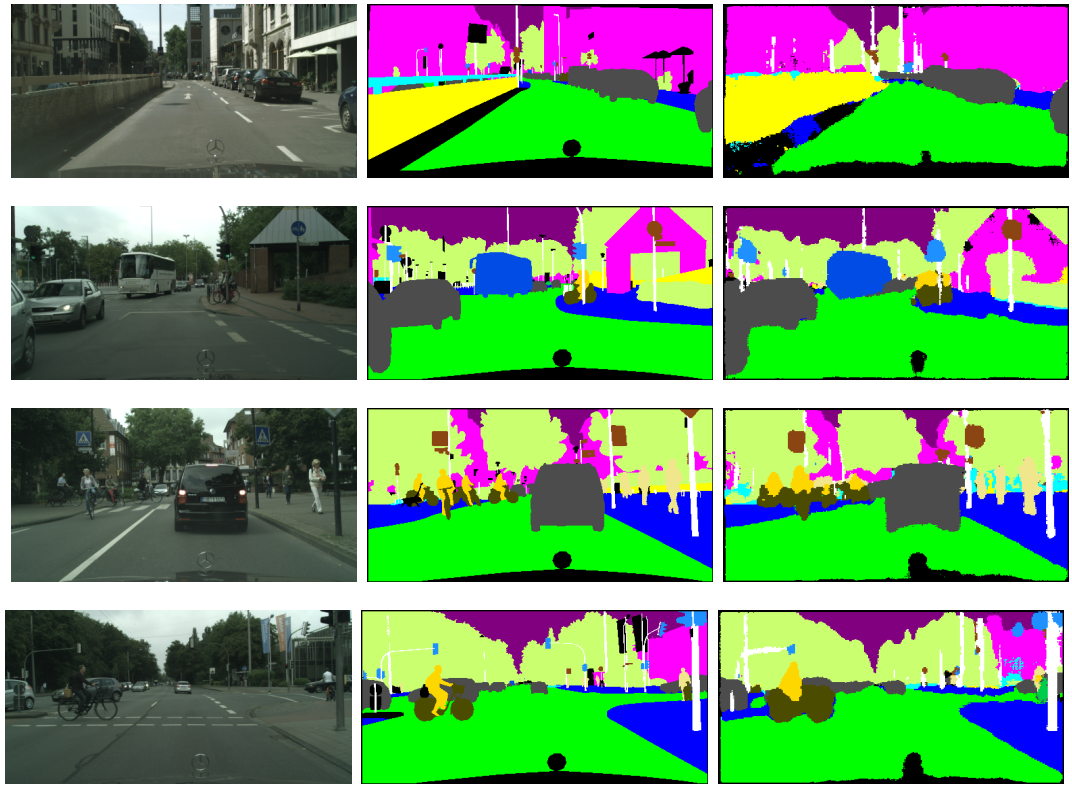

Fig. 2.3. Predictions on Cityscapes validation set [50]. From left to right the images belong to input image, ground truth, and output of our network respectively.

2.4 Conclusion

While designing neural network architecture, we must pay attention to complexity of each layer. Especially for applications which require model to run on the edge and in real-time, it becomes imperative that we do not indiscriminately add layers. In this chapter we provide an insight on ways to reduce number of parameters and operations. We tested our proposed model performance on various datasets and also on different hardwares.

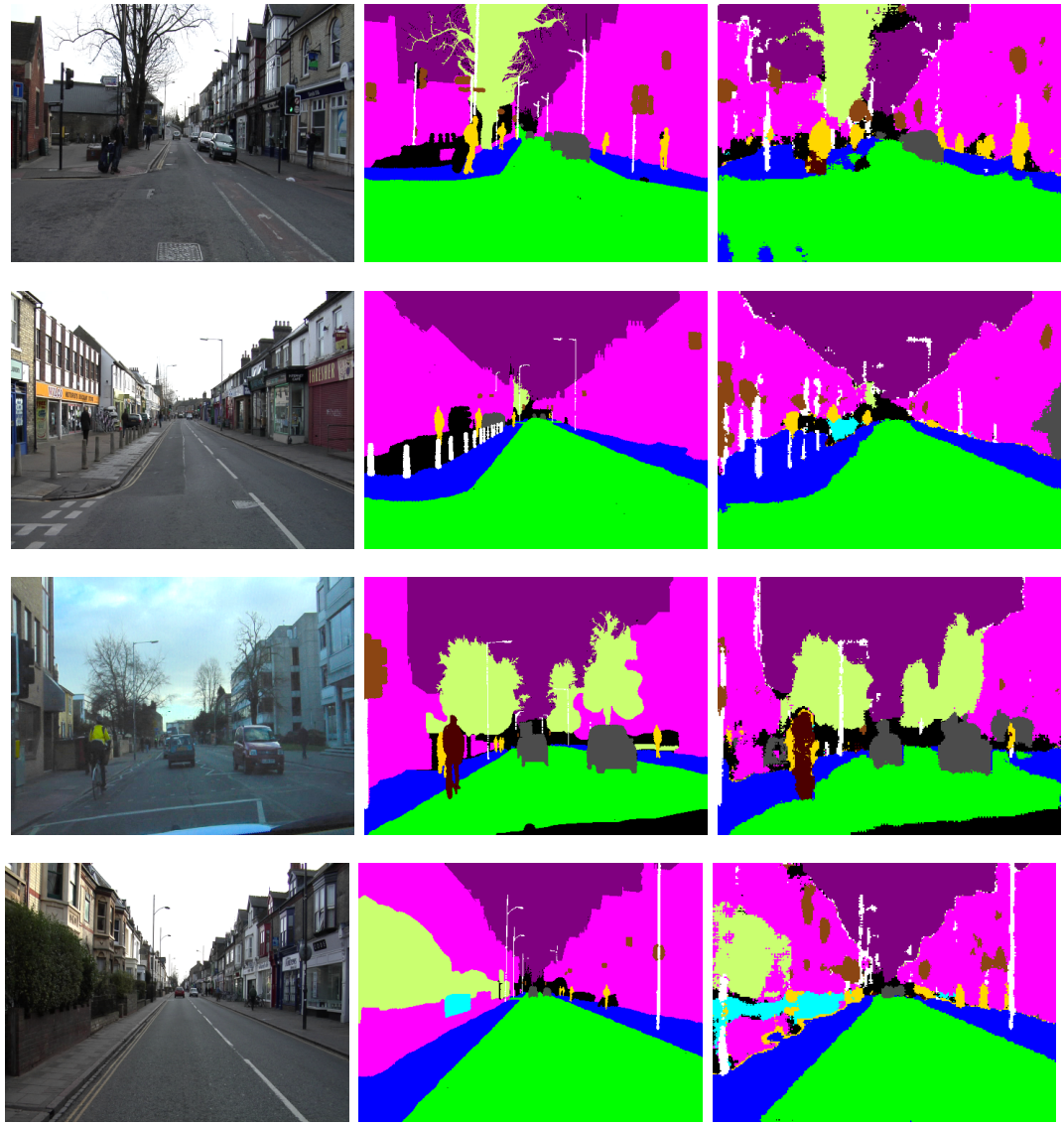

Fig. 2.4. Predictions on CamVid test set [59]. From left to right the images belong to input image, ground truth, and output of our network respectively.

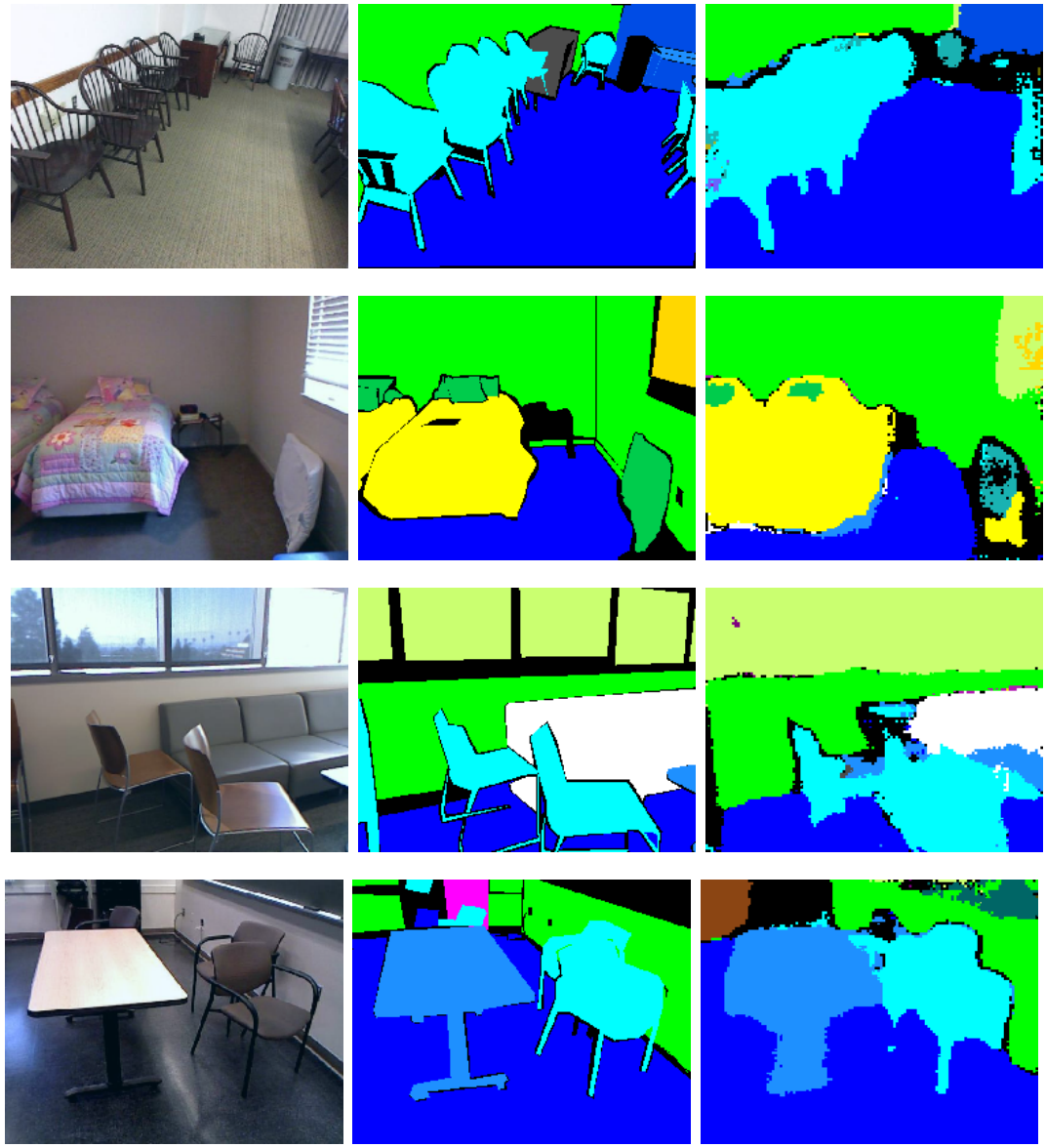

Fig. 2.5. Predictions on SUN RGB-D test set [60]. From left to right the images belong to input image, ground truth, and output of our work respectively.

3. EXPLORING ENCODER REPRESENTATIONS FOR EFFICIENT SEMANTIC SEGMENTATION

Pixel-wise semantic segmentation for visual scene understanding not only needs to be accurate but also efficient to find any use in real-time applications. Existing algorithms even though are accurate but they do not focus on utilizing the parameters of neural network efficiently. As a result, they are huge in terms of parameters and number of operations; hence slow too. In this chapter, we propose a novel deep neural network architecture which allows it to learn without any significant increase in the number of parameters. Our network uses only 11.5 million parameters and 21.2 GFLOPs for processing an image of resolution $3 \times 640 \times 360$. It gives a state-of-the-art performance on CamVid and comparable results on Cityscapes dataset. We also compare our network's processing time on NVIDIA GPU and embedded system devices with existing state-of-the-art architectures for different image resolutions.

Table 3.1.
Input and output feature maps

Block	Encoder		Decoder	
	m	n	m	n
1.	64	64	64	64
2.	64	128	128	64
3.	128	256	256	128
4.	256	512	512	256

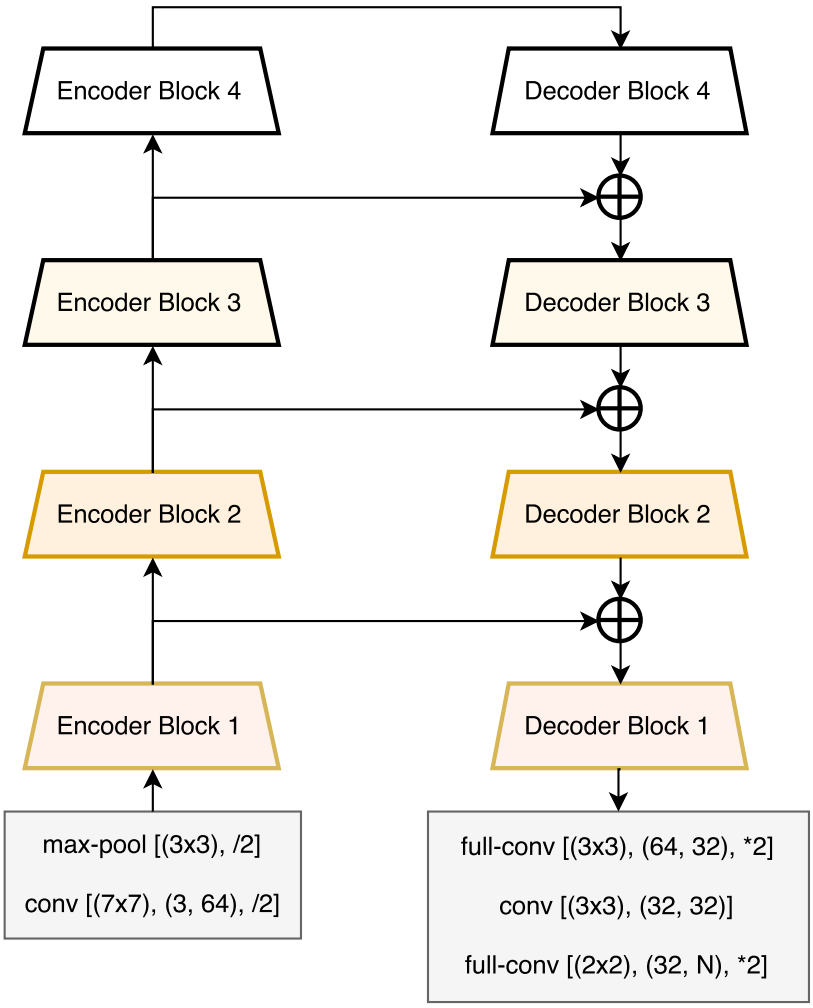

Fig. 3.1. Proposed network architecture

3.1 Network architecture, version 1

The architecture of LinkNet is presented in Figure 3.1. Here, `conv` means convolution and `full-conv` means full convolution [44]. Furthermore, /2 denotes downsampling by a factor of 2 which is achieved by performing strided convolution, and $*2$ means upsampling by a factor of 2. We use batch normalization between each convolutional layer and which is followed by ReLU non-linearity [54, 61]. Left half of the network shown in Figure 3.1 is the encoder while the one the right is the decoder. The encoder starts with an initial block which performs convolution on input image with

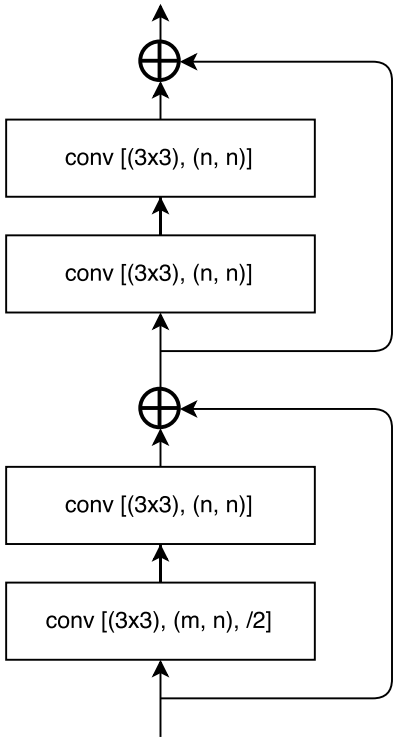

Fig. 3.2. Convolutional modules in *encoder-block (i)*

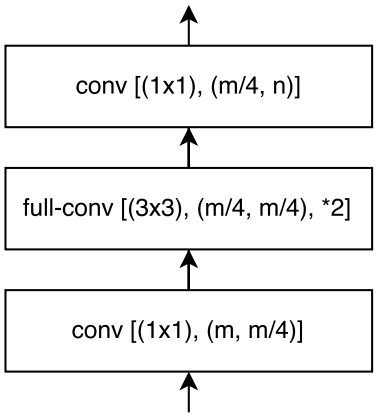

Fig. 3.3. Convolutional modules in *decoder-block (i)*

a kernel of size 7×7 and a stride of 2. This block also performs spatial max-pooling in an area of 3×3 with a stride of 2. The later portion of encoder consists of residual

blocks [22] and are represented as *encoder-block(i)*. Layers within these *encoder-block*s are shown in detail in Figure 3.2. Similarly, layer details for *decoder-block*s are provided in Figure 3.3. Table 3.1 contains the information about the feature maps used in each of these blocks. Contemporary segmentation algorithms use networks such as VGG16 (138 million parameters), ResNet101 (45 million parameters) as their encoder which are huge in terms of parameters and GFLOPs. LinkNet uses ResNet18 as its encoder, which is fairly lighter network and still outperforms them as evident from Section 2.3. We use the technique of full-convolution in our decoder as proposed earlier by [44]. Every `conv`$(k \times k)(im, om)$ and `full-conv`$(k \times k)(im, om)$ operations has at least three parameters. Here, $(k \times k)$ represent $(kernel - size)$ and (im, om) represent $(inputmap, outputmap)$ respectively.

Unlike existing neural network architectures which are being used for segmentation, our novelty lies in the way we link each encoder with decoder. By performing multiple downsampling operations in the encoder, some spatial information is lost. It is difficult to recover this lost information by using only the downsampled output of encoder. [41] linked encoder with decoder through pooling indices, which are not trainable parameters. Other methods directly use the output of their encoder and feed it into the decoder to perform segmentation. In this work, input of each encoder layer is also bypassed to the output of its corresponding decoder. By doing this we aim at recovering lost spatial information that can be used by the decoder and its upsampling operations. In addition, since the decoder is sharing knowledge learned by the encoder at every layer, the decoder can use fewer parameters. This results in an overall more efficient network when compared to the existing state-of-the-art segmentation networks, and thus real-time operation. Information about trainable parameters and number operations required for each forward pass is provided in detail in Section 2.3.

Table 3.2.
Performance comparison. Image size is W×H

Model	NVIDIA TX1						NVIDIA Titan X					
	480×320		640×360		1280×720		640×360		1280×720		1920×1080	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
SegNet	757	1.3	1251	0.8	-	-	69	14.6	289	3.5	637	1.6
ENet	47	21.1	69	14.6	262	3.8	7	135.4	21	46.8	46	21.6
LinkNet	108	9.3	134	7.8	501	2.0	15	65.8	53	18.7	117	8.5

Table 3.3.
Comparison on the basis of operations

	GFLOPs	Parameters	Model size (fp16)
SegNet	286.0	29.5M	56.2 MB
ENet	3.8	0.4M	0.7 MB
Proposed Net	21.2	11.5M	22.0 MB

3.2 Results

We compare LinkNet with existing architectures on two different metrics:

1. Performance in terms of speed:
   - Number of operations required to perform one forward pass of the network
   - Time taken to perform one forward pass
2. Performance in terms of accuracy on CamVid [59] dataset.

3.2.1 Performance Analysis

We report inference speed of LinkNet on NVIDIA TX1 embedded system module as well as on widely used NVIDIA TitanX. Table 3.2 compares inference time for a single input frame with varying resolution. As evident from the numbers provided, LinkNet can process very high resolution image at 8.5 fps on GPU. More importantly, it can give real-time performance even on NVIDIA TX1. '-' indicates that network was not able to process image at that resolution on the embedded device. Our network was also tested on Inference Engine (IE): an FPGA based neural network hardware accelerator [48, 49]. On IE4 which is a 250MHz 256 MACs 64KB maps buffer and 32KB kernel buffer system, our network's execution time for and image resolution of 480×320 was 168.1 ms. Whereas, on IE5 which has 1024 MACs and 512 KB maps buffer, proposed network took 82.58 ms for forward pass.

We choose 640×360 as our default image resolution and report number of operations required to process image of this resolution in Table 3.3. Number of operations determine the forward pass time of any network, therefore reduction in it is more vital than reduction in number of parameters. Our approach's efficiency is evident in the much low number of operations per frame and overall parameters.

3.2.2 Benchmarks

We use Torch7 [62] machine-learning tool for training with RMSProp as the optimization algorithm. The network was trained using four NVIDIA TitanX. Since the classes present in all the datasets are highly imbalanced; we use a custom class weighing scheme defined as $w_{\text{class}} = \frac{1}{\ln(1.02+p_{\text{class}})}$. This class weighing scheme has been taken from [63] and it gave us better results than mean average frequency. As suggested in Cityscapes [50], we use intersections over union (IoU) and instance-level intersection over union (iIoU) as our performance metric instead of using pixel-wise accuracy. In order to prove that the bypass connections do help, each table contains IoU and iIoU

Table 3.4.
Results on CamVid test set of (1) SegNet, (2) ENet, (3) Dilation8, (4) LinkNet without bypass, and (5) LinkNet

Model	Building	Tree	Sky	Car	Sign	Road	Pedestrian	Fence	Pole	Sidewalk	Bicyclist	IoU	iIoU
1	88.8	87.3	92.4	82.1	20.5	97.2	57.1	49.3	27.5	84.4	30.7	65.2	55.6
3	74.7	77.8	95.1	82.4	51.0	95.1	67.2	51.7	35.4	86.7	34.1	68.3	51.3
2	82.6	76.2	89.9	84.0	46.9	92.2	56.3	35.8	23.4	75.3	55.5	65.3	-
3	84.6	87.4	88.8	72.6	37.1	95.3	61.2	56.0	33.1	88.3	24.4	66.3	52.7
4	88.8	85.3	92.8	77.6	41.7	96.8	57.0	57.8	37.8	88.4	27.2	**68.3**	**55.8**

values with as well as without bypass. We also compare LinkNet's performance with other standard models such as SegNet [42], ENet [63], Dilation8/10 [33].

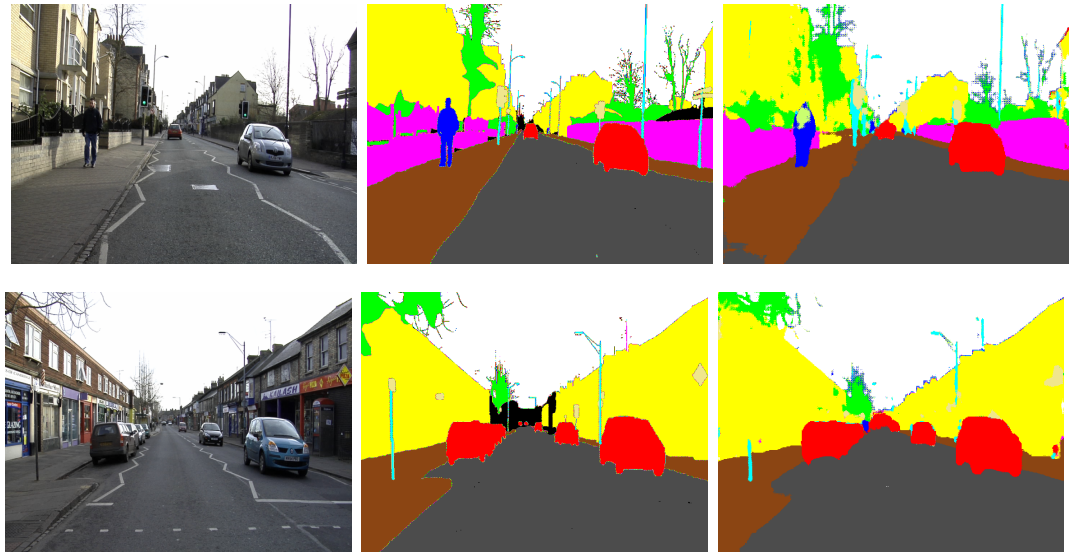

Fig. 3.4. LinkNet prediction on CamVid [59] test set. From left to right the images belong to input image, ground truth, and Linknet output respectively.

CamVid It is another automotive dataset which contains 367 training, 101 validation, and 233 testing images [59]. There are eleven different classes such as building, tree, sky, car, road, etc. while the twelfth class contains unlabeled data, which we ignore during training. The original frame resolution for this dataset is 960×720 (W,H) but we downsampled the images by a factor of 1.25 before training. Due to hardware constraint, batch size of 8 was used to train the network. In Table 3.4 we compare the performance of the proposed algorithm with existing state-of-the-art algorithms on test set. LinkNet outperforms all of them in both IoU and iIoU metrics. Segmented output of LinkNet can be seen in Figure 3.4.

3.3 Network architecture, version 2

We also propose a network with similar concept of bypass connections as shown in Figure 3.5. There are five key differences in this network as compared to the architecture in section 3.1.

- Input samples enter the network through two separate paths: start block 1 and 2. These two blocks are explained in Figure 3.6a. and b.
- Encoder of this network is inspired from dilated ResNet 34 [45].
- Bypass connection has a lateral module in it.
- Loss is calculated at different stages/layers of the model.
- Bilinear upsampling is used instead of transpose convolution.

Two different paths for the input and calculation of loss at different levels allow the network to learn at different scale. The encoder of this network uses dilated convolution, minimizing the loss in spatial information as data passes through it. Input and output maps and internal structure of encoder blocks are the same as shown in Table 3.1 and Figure 3.2. The difference is that after encoder block 2 there is no downsampling. Also, all the layers of encoder 3 and 4 have a dilation of 2 and

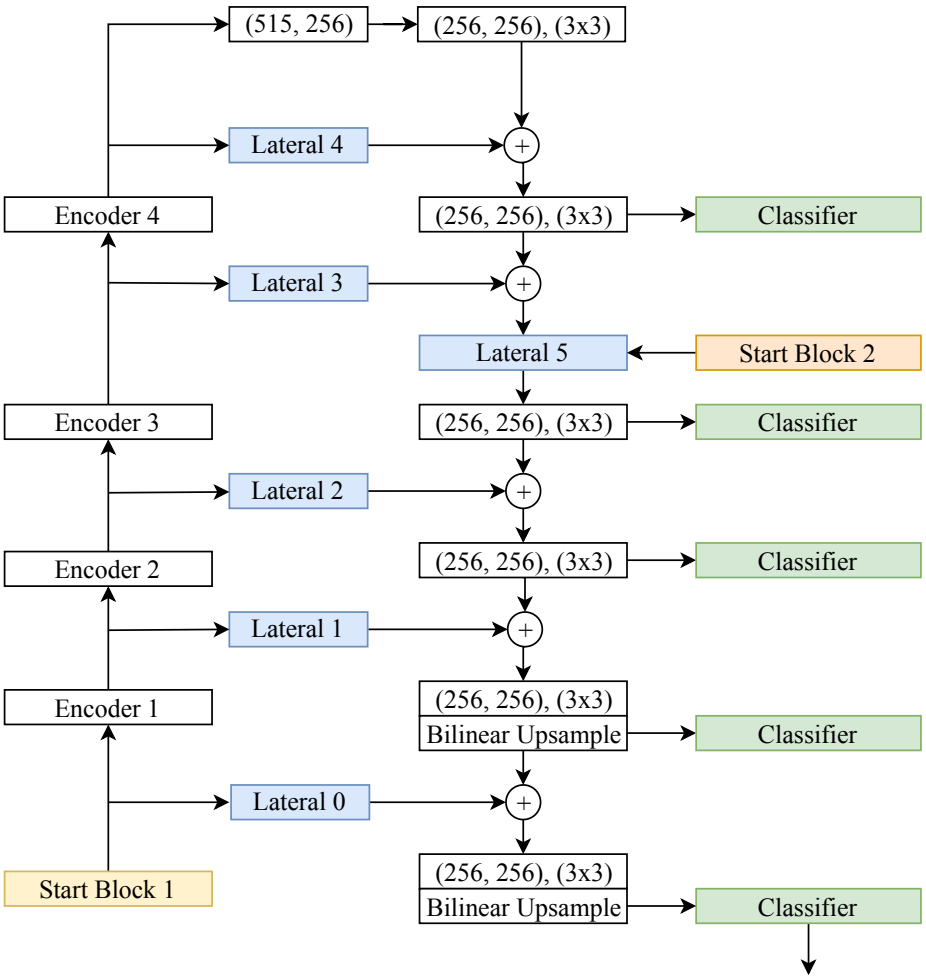

Fig. 3.5. Modified network architecture with branched encoder and multiple loss.

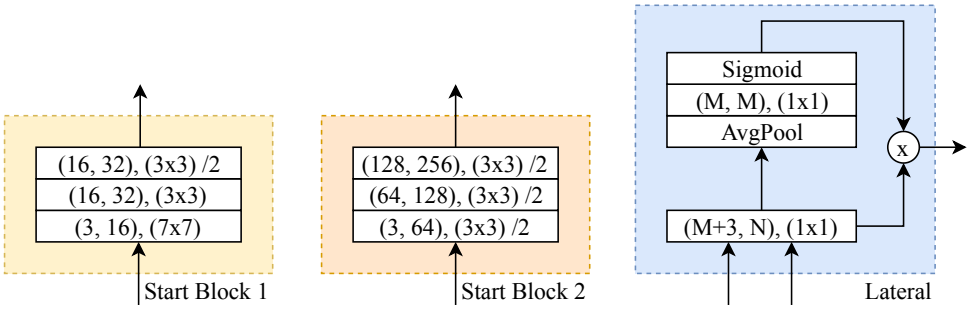

Fig. 3.6. (a) Start block 1 of encoder. (b) Start block 2 (c) Module for bypass connection. Both start block 1 and 2 take in the same input.

1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4
5	5	5	5	5

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

Fig. 3.7. (a) Positional encoding in x-direction. (b) Positional encoding in y-direction.

4 respectively. Moreover, encoder 4 contains two additional layers: the first with dilation of 2 and the second with dilation of 1.

The lateral module as shown in Figure 3.6.c. takes in two inputs. Input 1 will be output from the corresponding encoder block, while input 2 is fixed value with 3 feature maps. These feature maps contain positional encoding in x, y, and radial direction. An example of input of size 5×5 is displayed in Figure 3.7. This lateral module performs 1×1 convolution and then uses the attention mechanism (later part of the module) to extract only important features from encoder blocks.

Table 3.5.
Incremental improvement in performance as a result of addition of individual modules. A: Encoder with dilated convolution, B: Loss at multiple level, C: Positional encoding in lateral blocks, D: Multiple encoder paths

Module(s)	Class IoU
A	65.98
A + B	68.37
A + B + C	69.42
A + B + C + D	72.20

Cityscapes This dataset consists of 5000 fine-annotated images, out of which 2975 are available for training, 500 for validation, and the remaining 1525 have been selected as test set [50]. We trained on our network on 19 classes that was provided in the official evaluation scripts [50]. Input image of resolution 1024×512 was used for training the network. Table 3.5 shows the effect of addition of each module on network's performance.

We report performance values for our network calculated on validation set in Table 3.6. PSPNet [51] and Deep-Lab v3 [52] have better accuracy than our network, but as we said earlier, this number in itself is not sufficient. Our network is way smaller as compared to these networks. SegNet [42] and ENet [63] are way smaller than us but their accuracy is in 50s which is very low. That is why, we only compare the proposed network's performance with rest of the state-of-the-art networks. Among these models, our network architecture performs the best in terms of mean IoU per giga operations (GOps). Figure 3.8 shows the predicted segmented output on couple of cityscapes validation images. Confusion matrix of our network on cityscapes is shown in Figure 3.9.

Table 3.6.
Performance on cityscapes dataset [50].

Model	Class IoU	Parameters (M)	Operations (GOps)	IoU/GOps
SegNet [42]	57.0	39.8	315.1	0.180
ENet [63]	58.3	0.4	3.8	15.34
FCN 8 [33]	65.3	134.5	2685.1	0.024
PSPNet [51]	80.2	68.1	4344.7	0.018
Deep-Lab v3 [52]	81.3	59.3	1419.5	0.057
FRRN B [53]	71.8	23.9	915.5	0.078
Proposed network	72.2	25.1	679.9	0.106

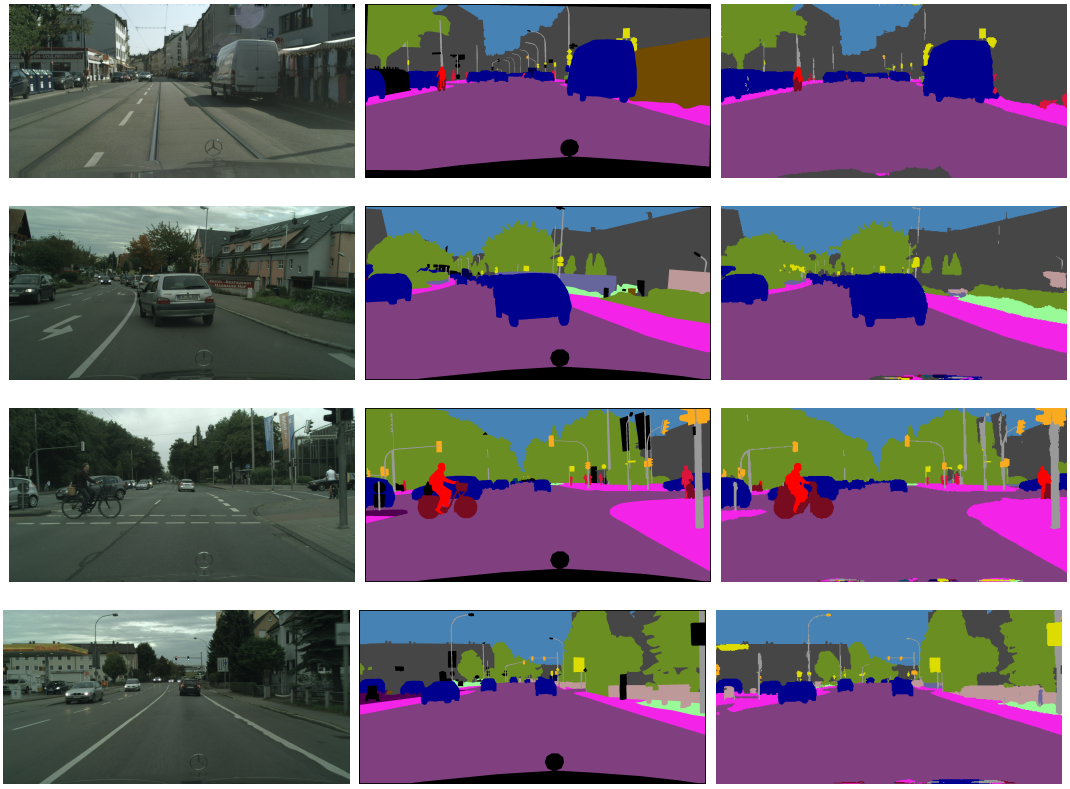

Fig. 3.8. Our network prediction on Cityscapes [50] validation set. From left to right the images belong to input image, ground truth, and network output respectively.

3.4 Conclusion

This chapter presents a few ways (bypass connection, multiple loss, and encoder path, etc.) to tackle the problem of loss in spatial information because of downsampling layers present in the encoder part of existing networks. As a result, we were not only able to achieve better accuracy, but we also reduced number of parameters of our decoder. In section 3.2 we also show that our network can be run even on edge devices in real-time.

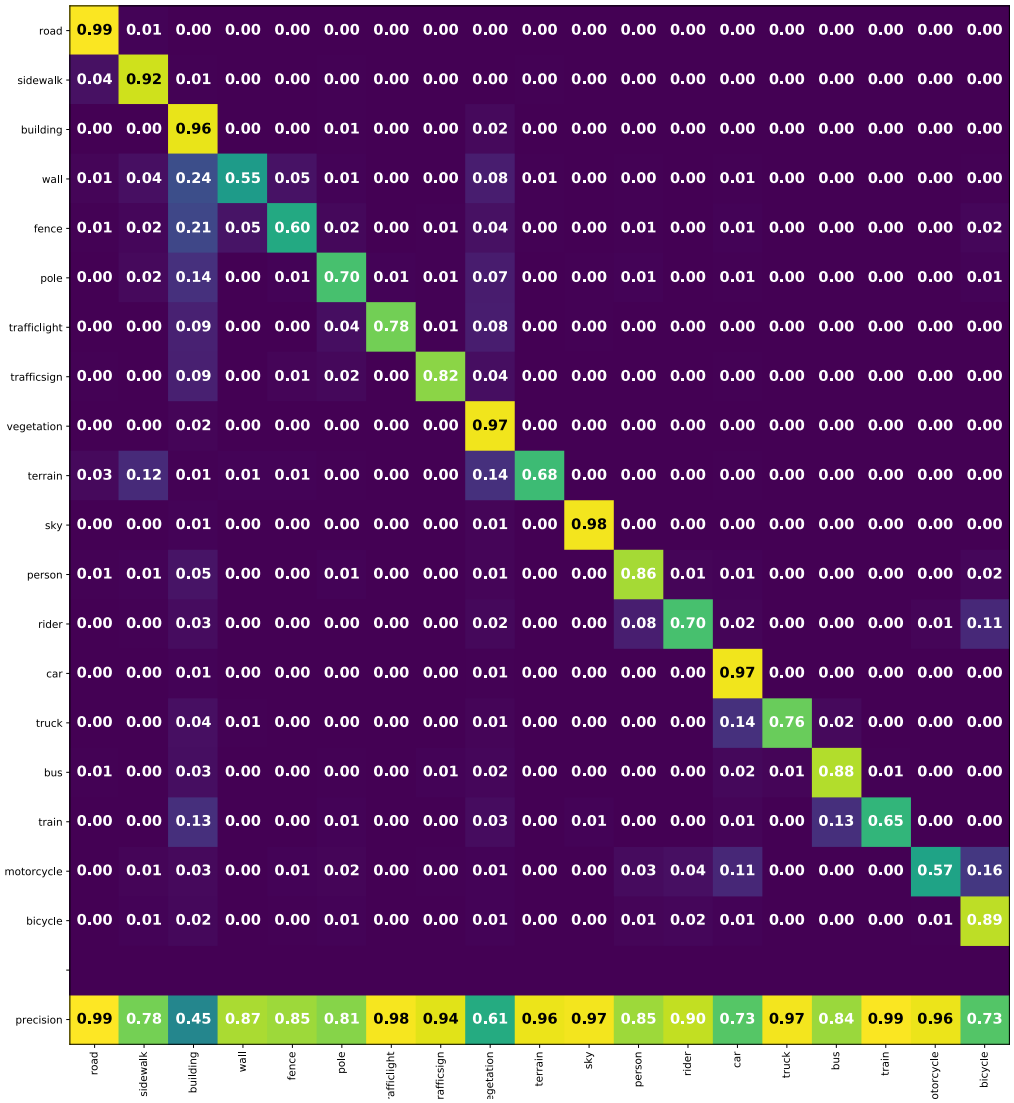

Fig. 3.9. Confusion matrix of proposed network on cityscapes [50] validation data.

4. LANE FOLLOWING SYSTEM FOR AUTONOMOUS DRIVING

Autonomous driving is no more just a concept for the future. Researchers are working day in and day out on building systems such as parking assistance, lane detector, 3-D mapping and localization, driver attention warning, etc. First of all, a route has to be decided to go from point A to B. This can be done using several existing simultaneous localization and mapping (SLAM) techniques [64]. After path planning, a vehicle needs to navigate through traffic. For this, it needs to have an idea about where cars, pedestrians and other objects are located. Even though segmentation masks and localization algorithms provide us valuable information such as the location of traffic signs and symbols, presence of cars and pedestrians in a 2D plane, we need to know exactly where an object is located in a 3D point cloud [65–67]. Then comes the need for lane awareness [68] which includes lane following as well as switching lanes when required. An ensemble of many more tasks like these when put together make autonomous driving a reality.

A lot of success in all the aforementioned areas can be attributed to the success of deep neural networks. Segmentation networks alone can serve as a solution to most of these problems, but still, they are only one of the many cogs in the wheel of autonomous driving. The segmentation mask in itself is not sufficient unless there is a way to process and translate that information to vehicle actuators. This decision making step is an integral part of self-driving vehicles that cannot be ignored. A network with low accuracy and very small memory and computation footprint might be sufficient for a strong and robust control system. Moreover, these controllers are the only tools that can clearly tell us how is our system going to perform while interacting in the real world.

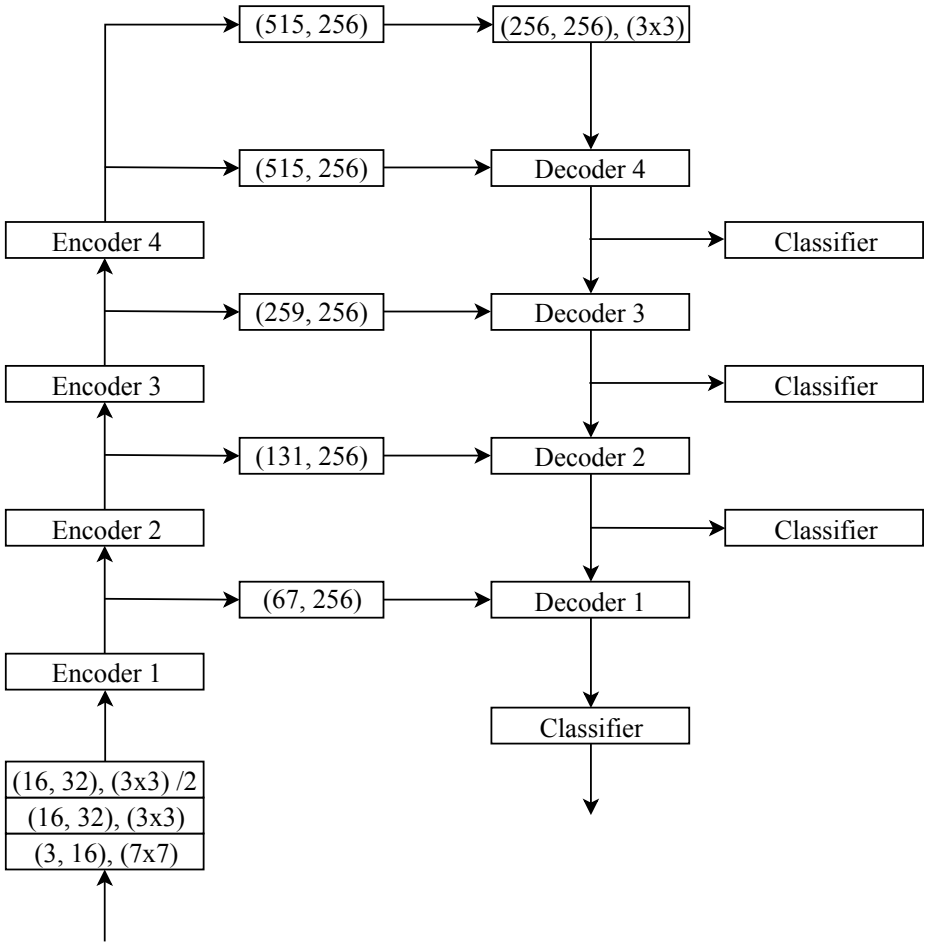

Fig. 4.1. Proposed network architecture for lane marking detection. Encoder of this network is similar to dilated ResNet 34 [45] and bypass connections are point-wise convolutions.

In this chapter we will select one of the many intermediate blocks required for autonomous driving: lane following. We design a segmentation model (Figure 4.1) capable of detecting lane markings in the real world. Some work has been done [69] in this area but on simple datasets such as CULane [68]. Then, we also propose a system which can take in segmentation maps and predict optimal steering angle and vehicle speed.

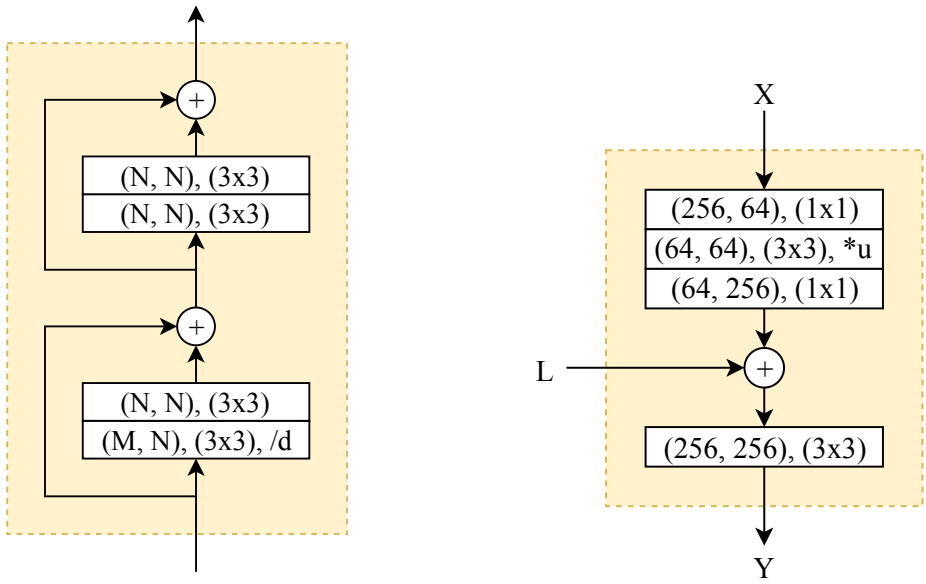

Fig. 4.2. (a) Encoder block (b)Decoder block. Each block represents a convolution layer followed by batch-normalization and ReLU.

4.1 Lane detection

4.1.1 Model architecture

The proposed network architecture is shown in Figure 4.1. Each block has entries like: $(M, N), (k \times k), x$. Here M and N represent input and output feature maps respectively, k denotes the kernel size, and in some blocks x when present represents downsampling or upsampling factor. $/2$ means convolution with a stride of 2 was used, whereas $*2$ indicates the use of transpose convolution with an upsampling factor of 2.

The encoder of this model is inspired by dilated ResNet 34 [45]. Dilation allows us in retaining the receptive field of a convolution layer with strides. As a result, we avoid losing spatial information which otherwise would have been the case with strided convolution. The first layer of encoder 1 and 2 have convolution with a stride of 2. Also, all the layers encoder 3 and 4 use dilated convolution. Encoder 3 has dilation of 2, and encoder 4 has a dilation of 4. The last layer of encoder 4 also has

Table 4.1.
Detailed encoder parameters

	M	N	Downsampling	Dilation
Encoder 1	32	64	2	1
Encoder 2	64	128	2	1
Encoder 3	128	256	1	2
Encoder 4	256	512	1	4

two extra layers with input and output feature maps equal to 512. The first layer out of these two extra layers has dilation of 2 and the last layer has no dilation. Details of each encoder block can be seen Figure 4.2.a and table 4.1. Dilation of 1 means simple convolution with no dilation.

All the bypass connections linking encoder blocks to decoder blocks are point-wise convolutions. The number of input feature maps of these layers does not match the number of corresponding encoder's output feature maps. It is because of the use of matrix with positional encoding [70]. We want our network to retain positional information as it goes through the encoder blocks. We hypothesize that there is a correlation between the position of each object in a given image, especially in the case of driving scenarios. In Figure 3.7 we show positional encoding in x and y direction for an input of size 5×5. Values of each cell can be calculated using the following equation:

$$z_{ij} = \sqrt{i^2 + j^2}$$

Size of this matrix is the same as that of the input of a given layer. This matrix is normalized and then concatenated with the input matrix along the feature dimension and then sent as an input to the layer in bypass connection.

Figure 4.2.b shows a detailed breakdown of decoder blocks. X is the output from the previous decoder and input of the current decoder block. L is the link coming from bypass connection and Y is the output of the decoder. Only decoder blocks 1

Fig. 4.3. Validation loss curve per epoch.

performs upsampling. The layer with $*u$ is the layer upsampling input X by a factor of 2 using transpose convolution. Encoder already has an understanding of what is in the image (it was pretrained on imagenet [12] dataset. Therefore, the task of the decoder is only to upsample the embedding of the encoder. For this simple task, we claim that a decoder with very few parameters will be sufficient. That is why in the bottleneck architecture used in decoder blocks, we reduce the number of feature maps by a factor of 4 using point-wise convolution and then perform 3×3 convolution on this reduced feature space. The output of each decoder block is fed into classifier blocks. A classifier block is a single convolution layer with a kernel size of 1×1. In our architecture, all these four classifiers share the same parameters. Classifiers at different stages ensure that our network will have the ability to identify objects at different scales and hence, give better performance.

4.1.2 Results

We used cross entropy as our loss function and Adam [58] as the optimization algorithm. Learning rate was set to $1e-3$ and weight decay parameter was set to $5e-4$. We trained our network using PyTorch framework [71] with a batch size of

16 for 200 epochs (Figure 4.3. The proposed network has 19.53 million parameters and 326.31 GOps for an image of size 910×512. Encoder was first pretrained on imagenet [12] dataset.

Dataset

Berkley Deep Drive (BDD) dataset [72] has 70000 training and 10000 validation samples for lane markings. These images were collected from various cities during different weather conditions and times of the day. That is why this is a very rich and challenging dataset. It has 10 categories: crosswalk, road curb, solid and dashed for single white, double white, single yellow, and double yellow. Annotations for this dataset include single lines along edges of lane markings. For training, we modify these annotations by passing them through a Gaussian kernel. Input images are downsampled to size 910×512 before feeding them to the network.

Network performance

In Figure 4.6 we show detected lane markings as a heat map on an image from the validation set. The first image shows the neural network generated lane markings overlayed over the input image. This input has four different lane markings present in it: crosswalk, road curb, broken white and solid white line. Left images are the output of the neural network while the right images are their corresponding ground truths. This example demonstrates the richness and in the meantime difficulty level of the dataset. As it can be seen in Figure 4.6, lane markings in the ground truth are not continuous lines. Sometimes they are just small fragments because of either how far they are or because of occlusions. Still, the network is able to detect them as it can be seen in the case of crosswalks. All three crosswalk markings are successfully detected by the proposed model. Figure 4.7 shows the output of the neural network on images from the validation set in the form of heat maps. All these images contain

different driving situations such as day, night and twilight time, straight, curved, city and highway road.

From heat map to precise position of lane marking

The position of the camera and the angle at which it is mounted on the vehicle is a critical piece of information required to find the location of lane markings. That is why we first calibrate our system in the beginning by selecting four points on the input image such that when seen from the top, those four points will form corners of a rectangle. From this, we get a region of interest (ROI) indicated by the red box in Figure 4.8. This ROI will remain the same until the mounted camera is moved.

Since BDD dataset [72] has 10 classes, our network generates 10 separate output maps for each class. We process each map individually during the inference step. Steps involved in generating the final output shown in Figure 4.8 are as follows:

- Get heat-maps from the neural network.
- Extract region from the predictions belonging to the ROI.
- Perform thresholding on this cropped prediction to remove background from lane markings. This step will give Figure 4.8.b.
- Use the planar projection transformation matrix obtained using the four points selected during the calibration step to get the top-down perspective of the detection (Figure 4.8.c).
- Now find the center of these generated blobs and x-coordinate of the blobs will tell us position of the lane markings in the horizontal direction.

Empirically we can see the robustness of the proposed network in detecting the position of lane markings. Figure 4.9 contains few images from validation set and performance of our proposed system. There are very few samples of double solid lines in the training set and still, it gets detected in Figure 4.9.e. Apart from this, it

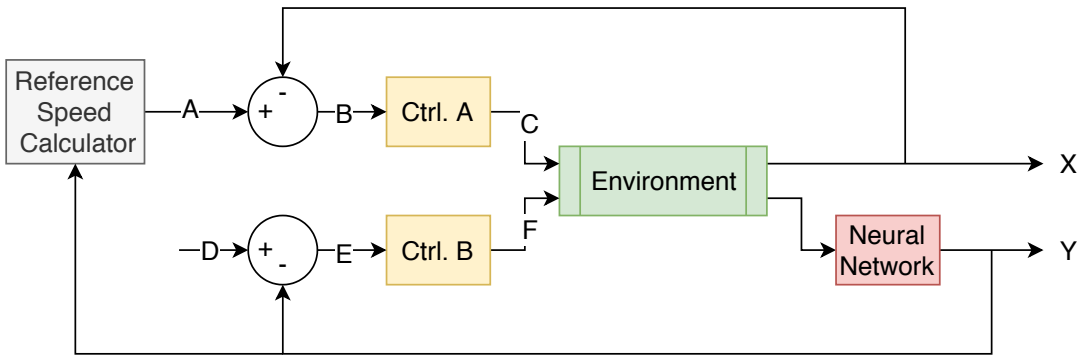

Fig. 4.4. Proposed feedback control system for lane following assistant. **B** is error in speed, **E** is positional error. **C** and **F** represent suggested throttle and steering angle respectively. Similarly, **X** is actual speed and **Y** contains vehicle's current position.

successfully detects yellow solid line and road curb even when they are very close by (two solid yellow lines) in Figure 4.9.f. As it can be seen, our lane marking detection system works well for multiple lanes. This can prove helpful in tasks involving lane switching.

4.2 Lane-following system

Once we know the position of lanes, we can find out the vehicle's current lane based on lane positions with respect to (w.r.t.) the camera. The difference between lane center and vehicle center can then be used to keep the vehicle within the lane. This type of system can prove helpful in providing feedback to drivers once they start deviating from their lane, but it is not sufficient in itself to move from one point to another.

The block diagram of our proposed lane following assistant is shown in Figure 4.4. Ctrl. A and Ctrl. B are two proportional controllers. While Ctrl. A predicts throttle of the vehicle, Ctrl. B predicts the steering angle. These values are sent to the actuators and then they interact with the environment. After that, sensors gather the vehicle's positional information and speed. Positional information includes vehicle

orientation and location w.r.t. the current lane. A segmentation network capable of detecting lane as proposed in section 4.1.1 can give us the location of the vehicle in the lane. To calculate the angle between lane marking and vehicle, we can do curve fitting over segmented output and then draw tangent on that curve. Value **D** contains reference angle θ and reference position α. θ can be calculated while installing the camera and it is the angle between lane marking and vehicle when the vehicle is moving forward in a straight line. Reference position α is the center of the vehicle.

Figure 4.5 shows the angle calculation in detail. Region of interest is extracted from input image (indicated by red box in Figure 4.5).a. Segmented output of lane marking (Figure 4.5.b) is then generated for that ROI. For angle calculation, we only use the immediate left lane marking. So we calculate centroids of both edges and pick the left one. A non-linear curve is fit on top and bottom region of the detected lane marking, depicted by red and green lines respectively in Figure 4.5.c. Tangents on these two curves give is angles θ_1 and θ_2. The difference between these angles is indicative of incoming curb or straight lane. Bigger $\Delta\theta$ means a curb is ahead while smaller $\Delta\theta$ indicates a straight lane.

Total positional error **E** is then fed to controller B which eventually predicts the optimum steering angle. The segmented output is also passed to the reference speed calculator. In case of incoming turns or when the vehicle is at an offset with the lane center, the reference speed calculator block suggests reducing the speed. Reference speed **A** is then used to calculate the difference between expected (**A**) and current (**X**) speed and is then feed to proportional controller A. After this, **C** finally calculates the required amount of acceleration or deceleration.

To show the effectiveness of our driver assistant for lane following, we test it on CARLA which is an open-source simulator for autonomous driving [73]. All the lane markings in CARLA are labeled as one class and important classes such as road curb are missing in this simulator. That is why we did not train our network on CARLA data. Instead, we directly use the lane marking label provided by the simulator and

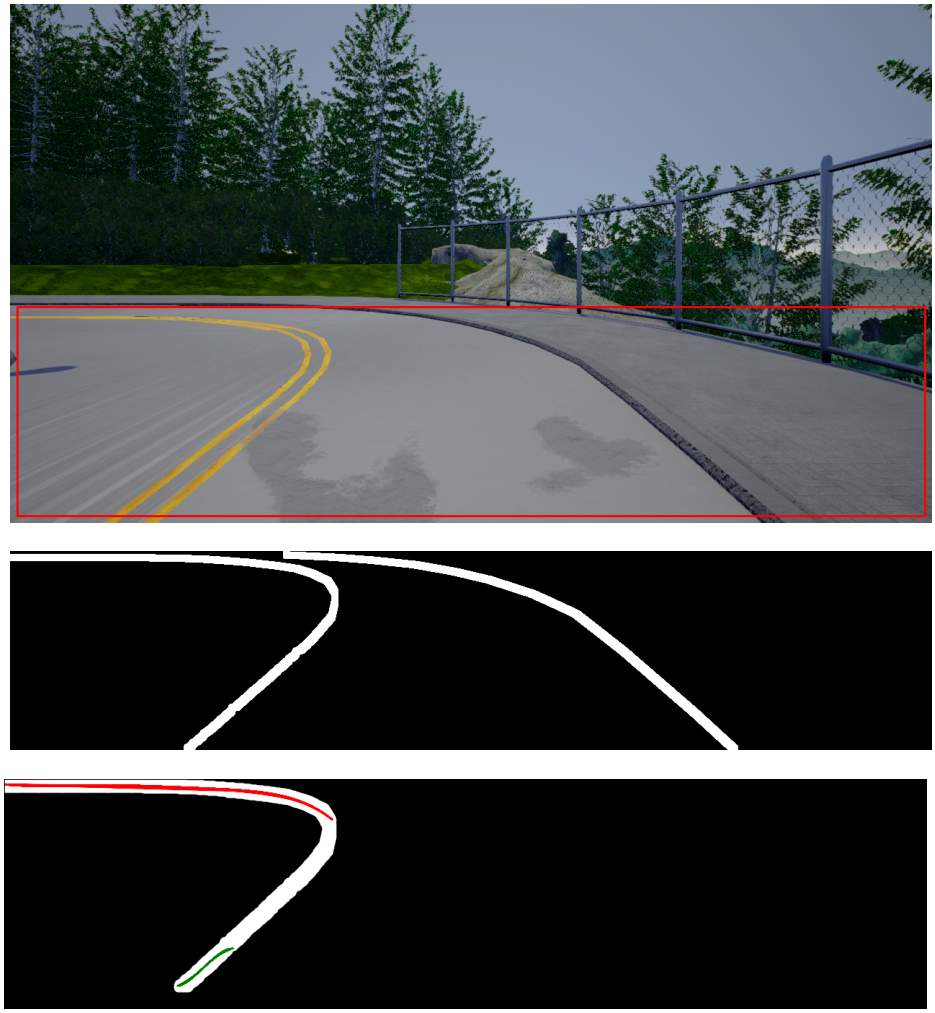

Fig. 4.5. Top to bottom: (a) Input image with ROI from CARLA [73]. (b) Lane marking provided by the simulator and road curb extracted using edge detector. (c) Curve fit on the left lane marking in order to make a decision about incoming turn.

check our proposed system's performance. Our designed control system's response to an approaching turn is shown in Figure 4.10.

The top horizontal bar shows the steering angle, while the vertical bar on the right is for throttle. The green dot at the bottom is the center point of the lane where the vehicle should ideally be, and the arrow is showing all the information in a combined form. Initially, when there is a straight road, the steering direction is close to zero. Since the current speed is 10mph and the reference speed is 30mph, the vertical bar

is full. This means that the vehicle needs to accelerate. On the other hand in the next image, a turn is detected. That is why the arrow at the bottom of the image becomes smaller and the vertical bar starts filling downwards, indicating the vehicle needs to decelerate.

4.3 Conclusion

In this chapter, we present an algorithm using a deep neural network to detect lane markings. The proposed network was successfully able to learn even difficult instances such as almost overlapping road curb and yellow line, distinguish between double and single solid lines. We also showed that using our approach multiple lanes were also detected, that too in various driving conditions. Later we develop a feedback control system using a proportional controller which utilizes the lane marking information provided to it. Based on that information, it suggests the steering and throttle values for a vehicle.

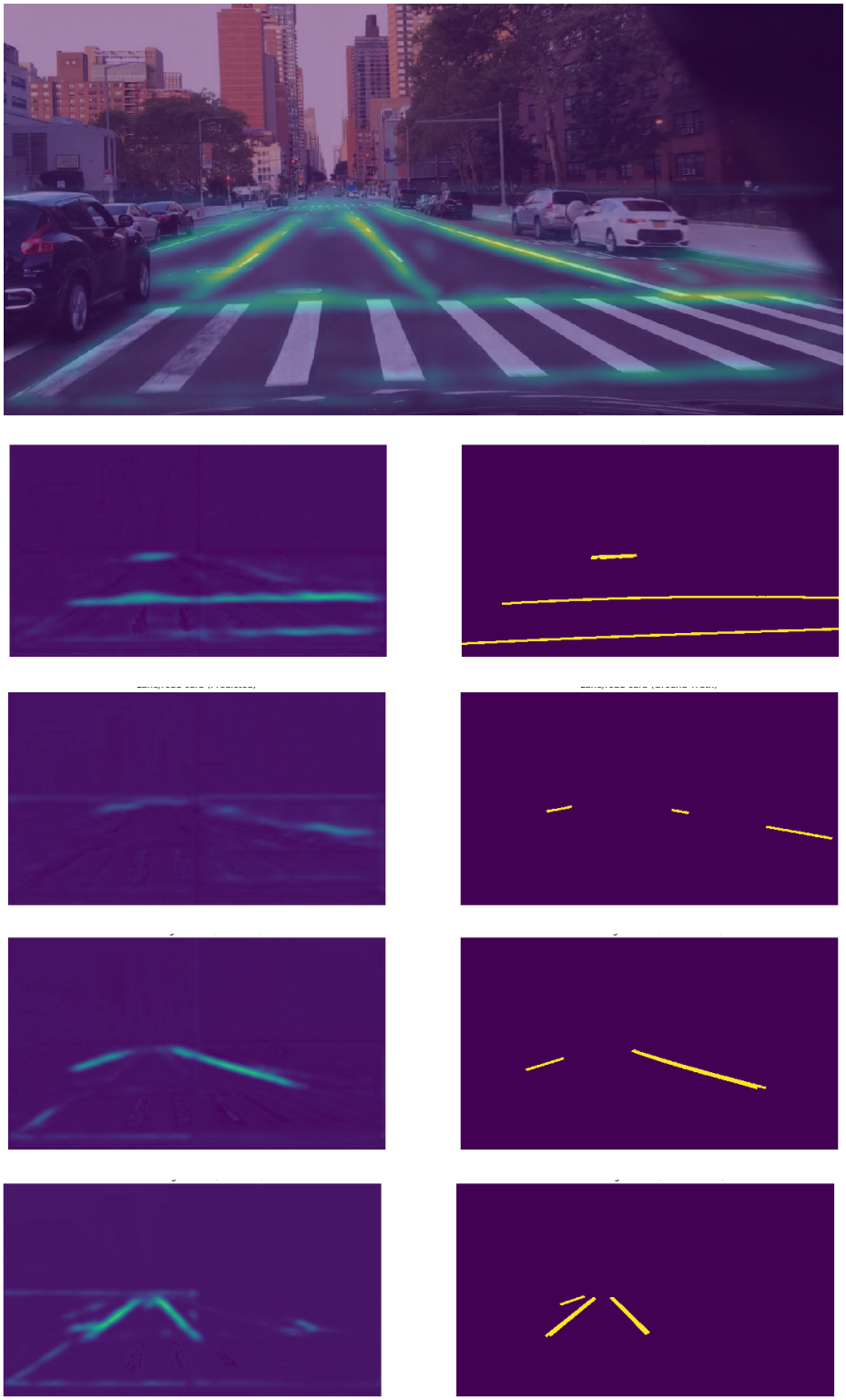

Fig. 4.6. Heat map showing detected lane markings and their corresponding ground truths.

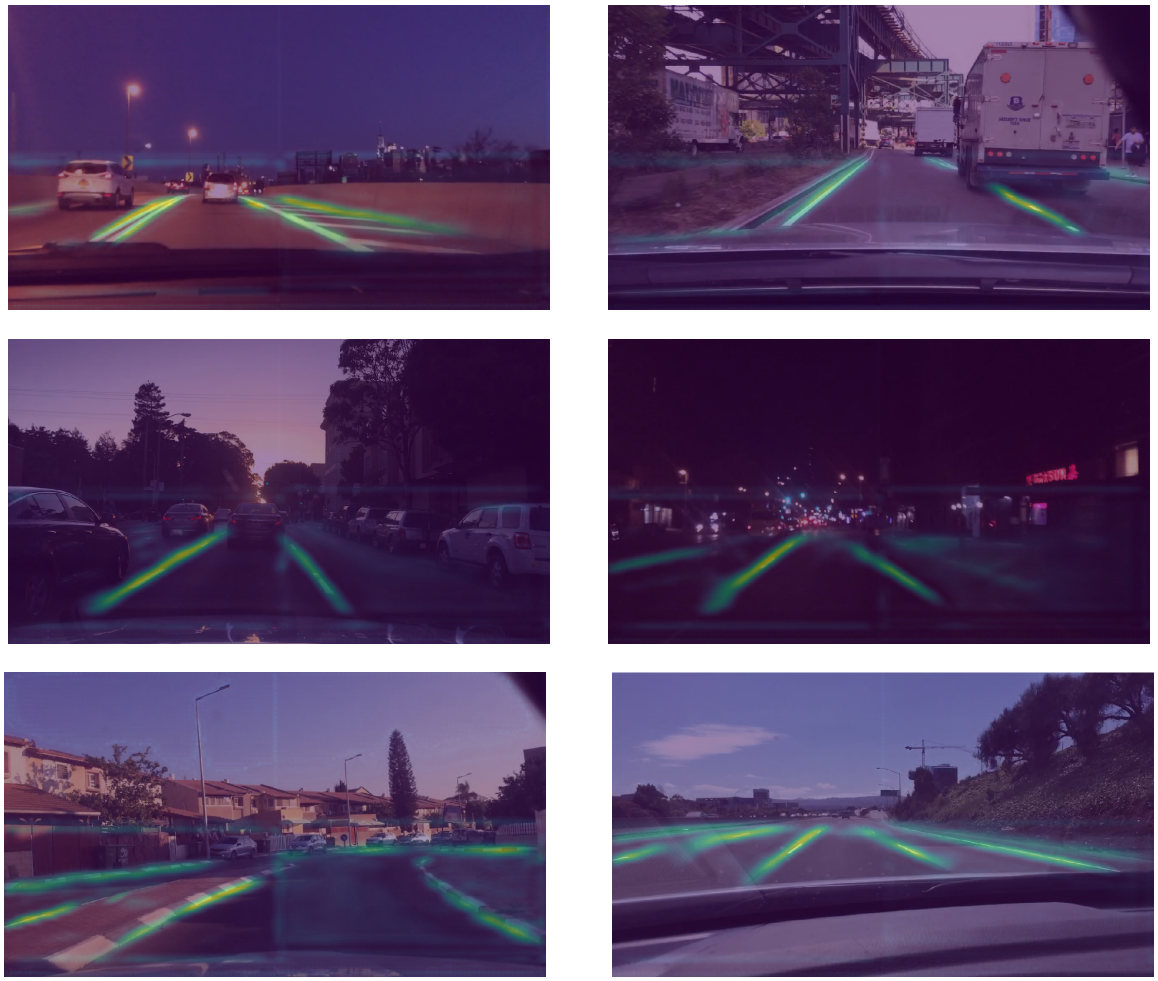

Fig. 4.7. Output of proposed network on images from validation set. These images include day and night time, highway as well as busy city road, and straight and curve roads.

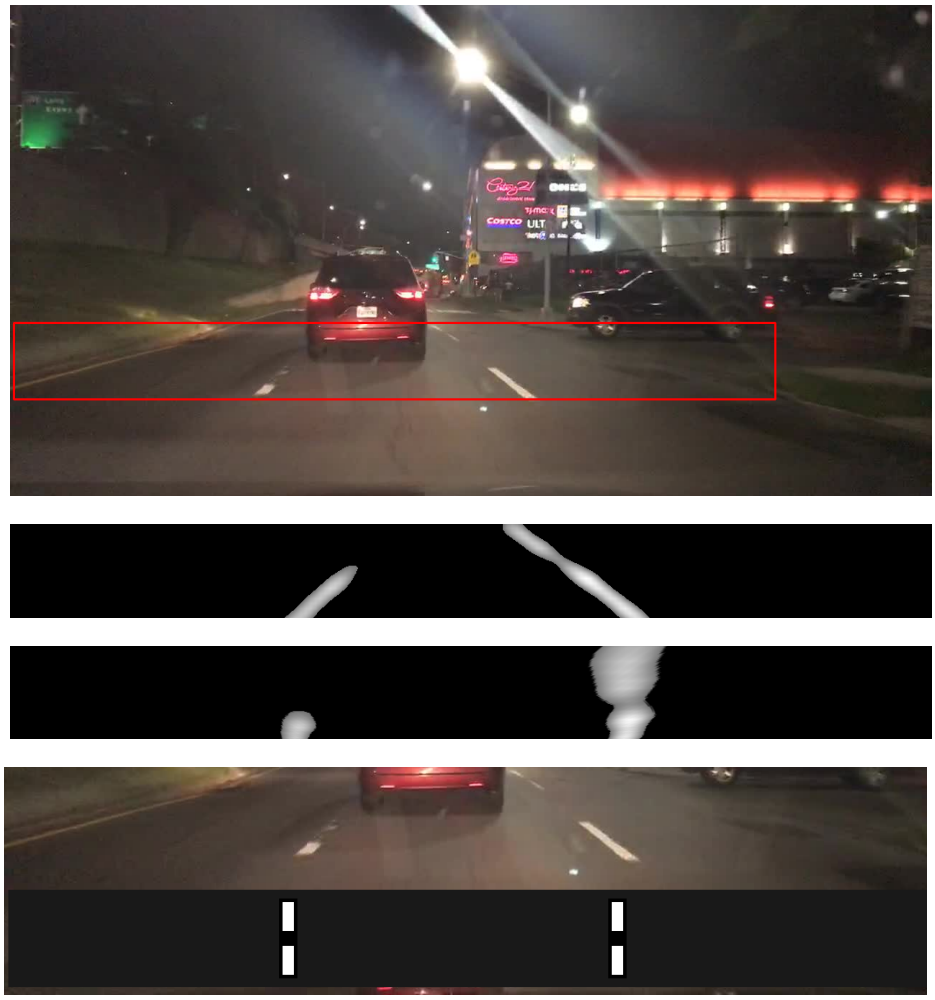

Fig. 4.8. Top to bottom: (a) Input image with region of interest. (b) Neural network output. (c) Detected lane marking when viewed from top perspective. (d) Broken white line as final output with their position on region of interest.

Fig. 4.9. Moving left to right and top to bottom, detected lane markings are: (a) road curbs and broken white line, (b) road curbs and broken white line, (c) broken white lines, (d) solid yellow and broken white lines, (e) double solid white line and broken white line, (f) road curb, solid yellow and white lines. Here, both road curbs as well as solid yellow lines are shown using yellow lines.

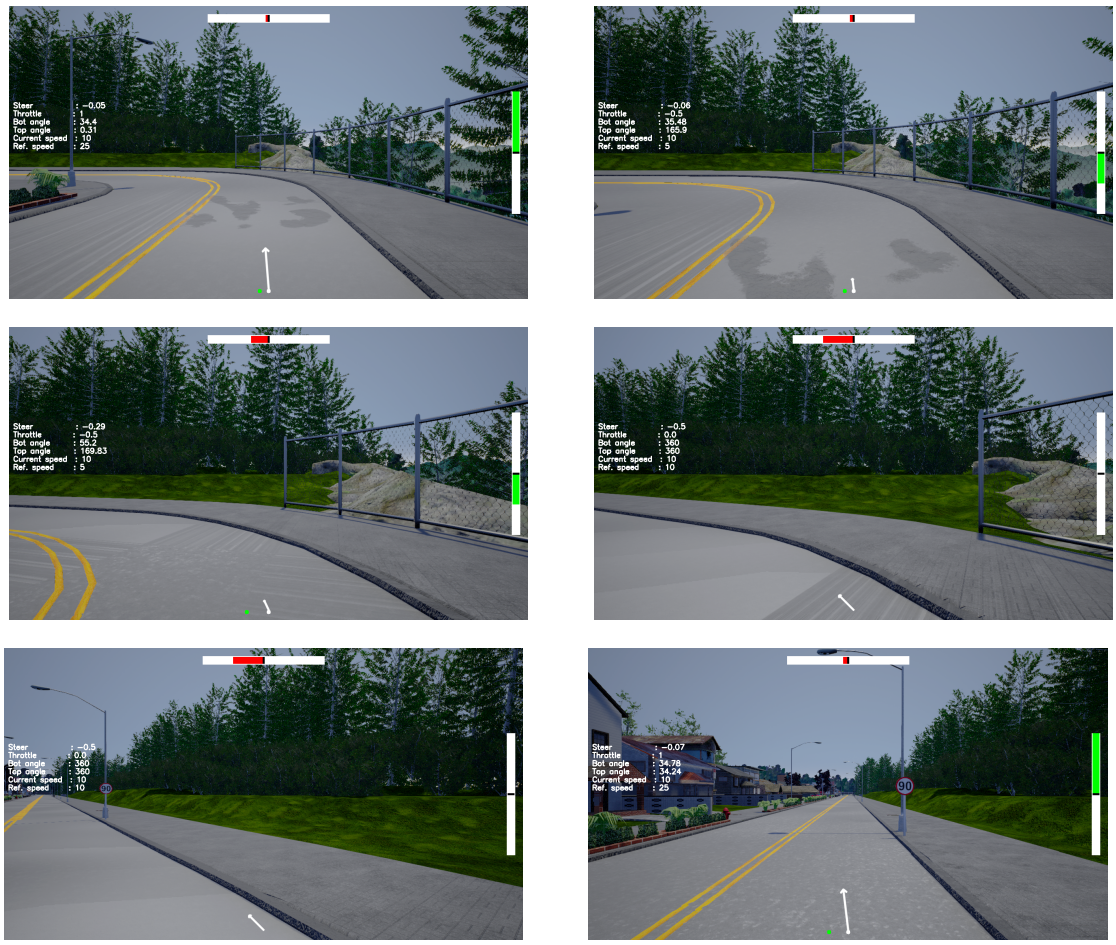

Fig. 4.10. Suggested steering angle and throttle value by proposed method while performing a sharp turn. Vertical bar on the right is for throttle and horizontal bar on top is for steering angle. White arrow is a vector using both the values. Green dot at the bottom represents center of the lane.

5. SUMMARY

This dissertation focuses on designing efficient deep neural networks for semantic segmentation. We show the capability of these models on benchmarks such as accuracy, and memory and computation footprint.

5.1 Conclusion

Chapter 2 explores existing neural network modules/layers and uses them to build a new architecture. It highlights the fact that it is possible to achieve the desired performance in terms of accuracy without compromising performance in terms of processing time. The proposed network has a very high density of information as compared to existing models. This can be inferred from its comparable performance on accuracy scale while having the number of parameters as low as 0.37 million. Devices on which state-of-the-art networks did not even fit, we show that our network can run on them in real-time.

Encoders of most of the existing segmentation networks contain downsampling layers because of which important spatial information is lost. As a result, decoders need to be strong enough to relearn this lost information and bring back encoder output to input space. Chapter 3 uses a novel approach to share information between encoder and decoder at each layer of abstraction, minimizing the loss in information because of downsampling. This allows the decoder to be small and yet give comparable results. Apart from bypass connections, dilated convolution and branched encoders were also used to preserve spatial context. Multiple loss at different levels in the network were also shown to help learn better features. This chapter also highlights the effect of individual modules on network's performance. Especially mul-

tiple loss and multiple encoder paths help in improving performance of our proposed architecture.

Segmentation networks are extensively used in the are of autonomous driving. Chapter 4 puts forth a novel segmentation network to detect lane markings in a variety of driving situations. The model is successfully able to detect markings from different lanes and decide the vehicle's position with respect to the current lane. Finally, a driver assistance system is proposed, capable of navigating a vehicle from point A to B by controlling its speed and steering angle. A driving simulator is used to demonstrate the capabilities of the designed system while navigating on straight tracks as well as during sharp turns.

5.2 Limitations

Networks proposed in this thesis were trained on separate images and not on a sequence of images. That is why, even though these networks have good representation for 'discrete' scenes; they do not have any temporal information in them. As a result, the output of these networks varies noticeably for even two consecutive frames, giving a perception of jittery output on videos.

The current lane detector requires an initial setup to get the perspective transformation matrix. This might become problematic if the position of the mounted camera changes due to vibrations in the vehicle. Sometimes, the width of lanes varies even in close by areas. In such situations, the reference angle used in the lane following system will have to be manually tuned.

5.3 Future work

- In order to store temporal information, some memory modules such as recurrent neural networks can be added to the network.

- At present, only the encoders are pretrained and decoders are trained from scratch for semantic segmentation. Since getting segmented data is a daunting task, unsupervised techniques to pre-train a whole network is worth exploring.

- Instead of perspective transform, Hough transform can be used. This might allow us to skip the calibration step for lane detection. It might also help us in the adaptive calculation of the reference angle. To use the Hough transform, the segmented output of the network will have to be improved.

- The proposed lane following assistant does not take traffic or pedestrians into consideration. A deep neural network capable of giving segmentation mask with depth information can potentially help in designing a proportional-integral-derivative controller to respond to such situations.

- The updated driver assistant can later be used as a method to verify if mean IoU of 70% is sufficient or a heavier segmentation network with a 90% mean IoU is required.

REFERENCES

REFERENCES

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *international Conference on computer vision & Pattern Recognition (CVPR'05)*, vol. 1. IEEE Computer Society, 2005, pp. 886–893.

[2] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*. IEEE, 1998, pp. 555–562.

[3] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, no. 3, pp. 32–57, 1973.

[4] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep 1995. [Online]. Available: https://doi.org/10.1007/BF00994018

[5] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, pp. 255–258, 1998.

[6] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[7] D. O. Hebb, *The Organization Of Behaviour*. Wiley, 1949.

[8] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[9] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct 1990.

[10] J. J. Weng, N. Ahuja, and T. S. Huang, "Learning recognition and segmentation using the cresceptron," *International Journal of Computer Vision*, vol. 25, no. 2, pp. 109–143, Nov 1997. [Online]. Available: https://doi.org/10.1023/A:1007967800668

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

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

[14] 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*, 2015, pp. 1–9.

[15] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context," *Int. Journal of Computer Vision (IJCV)*, January 2009.

[16] F. Perronnin, Y. Liu, J. Snchez, and H. Poirier, "Large-scale image retrieval with compressed fisher vectors," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 3384–3391.

[17] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers, and A. W. M. Smeulders, "Segmentation as selective search for object recognition," in *IEEE International Conference on Computer Vision*, 2011.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[19] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Neural Networks: Tricks of the Trade*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.

[20] M. A. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, 2007, pp. 1–8.

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

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[23] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[24] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[25] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 648–656.

[26] P. Sturgess, K. Alahari, L. Ladicky, and P. H. Torr, "Combining appearance and structure from motion features for road scene understanding," in *BMVC 2012-23rd British Machine Vision Conference*, 2009.

[27] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2650–2658.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *arXiv preprint arXiv:1603.05027*, 2016.

[29] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv preprint arXiv:1512.00567*, 2015.

[30] I. Sobel, "An isotropic 3x3 image gradient operator," *Presentation at Stanford A.I. Project 1968*, 02 2014.

[31] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," *arXiv preprint arXiv:1412.5474*, 2014.

[32] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

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

[34] X. Ren, L. Bo, and D. Fox, "Rgb-(d) scene labeling: Features and algorithms," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 2759–2766.

[35] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, Aug 2013.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[37] 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*, 2015, pp. 91–99.

[38] 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*. Springer, 2016, pp. 21–37.

[39] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "Icnet for real-time semantic segmentation on high-resolution images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 405–420.

[40] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 689–696.

[41] V. Badrinarayanan, A. Handa, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling," *arXiv preprint arXiv:1505.07293*, 2015.

[42] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *arXiv preprint arXiv:1511.00561*, 2015.

[43] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1520–1528.

[44] 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*, 2015, pp. 3431–3440.

[45] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014.

[46] F. Visin, M. Ciccone, A. Romero, K. Kastner, K. Cho, Y. Bengio, M. Matteucci, and A. Courville, "Reseg: A recurrent neural network-based model for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016, pp. 41–48.

[47] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr, "Conditional random fields as recurrent neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1529–1537.

[48] A. X. M. Chang, A. Zaidy, L. Burzawa, and E. Culurciello, "Deep neural networks compiler for a trace-based accelerator (short wip paper)," in *Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. ACM, 2018, pp. 89–93.

[49] A. Zaidy, A. X. M. Chang, V. Gokhale, and E. Culurciello, "A high efficiency accelerator for deep neural networks," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE, 2018, pp. 9–13.

[50] 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.

[51] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network." *arXiv preprint arXiv:1612.01105*, 2016.

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

[53] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe, "Full-resolution residual networks for semantic segmentation in street scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4151–4160.

[54] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[55] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[56] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep networks with stochastic depth," *arXiv preprint arXiv:1603.09382*, 2016.

[57] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

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

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

[60] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 567–576.

[61] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[62] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

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

[64] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, June 2006.

[65] P. Wang, R. Yang, B. Cao, W. Xu, and Y. Lin, "Dels-3d: Deep localization and segmentation with a 3d semantic map," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5860–5869.

[66] D. Maturana, P.-W. Chou, M. Uenoyama, and S. Scherer, "Real-time semantic mapping for autonomous off-road navigation," in *Field and Service Robotics*. Springer, 2018, pp. 335–350.

[67] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3061–3070.

[68] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial cnn for traffic scene understanding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[69] R. F. Berriel, E. de Aguiar, A. F. D. Souza, and T. Oliveira-Santos, "Ego-lane analysis system (ELAS): dataset and algorithms," *arXiv preprint arXiv:1806.05984*, 2018.

[70] R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski, "An intriguing failing of convolutional neural networks and the coordconv solution," in *Advances in Neural Information Processing Systems*, 2018, pp. 9605–9616.

[71] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS Autodiff Workshop*, 2017.

[72] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving video database with scalable annotation tooling," *arXiv preprint arXiv:1805.04687*, 2018.

[73] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

VITA

VITA

Abhishek received his Bachelors degree in Electronics and Communication Engineering from the Indian Institute of Technology Guwahati, India in 2012. As an undergraduate, most of his work was in robotics and computer vision. In summer 2011, he did his internship at Hanyang University under Prof. Frank Chung-Hun Rhee. After that, he was offered a scholarship for Masters at Hanyang University. He went ahead with it and received his Masters degree in Electronics and Communication Engineering from Hanyang University, South Korea in 2014. His Masters thesis was on clustering techniques using interval type-2 fuzzy logic. He joined Purdue University in Fall 2014 and is currently a Ph.D. student in Electrical and Computer Engineering as a part of e-Lab under Dr. Eugenio Culurciello. Abhishek's research is mainly focused on developing deep neural network architectures for real-world interaction/application, using supervised, unsupervised and reinforcement techniques.