Bachelor Thesis

at the Department of Mathematics and Computer Science

Pixel-wise Semantic Segmentation for Low-power Devices

Albert Mkhitaryan
albert.mkhitaryan@fu-berlin.com
Matrikelnummer: 4812053

Supervisors: Prof. Dr. Daniel Göhring and Prof. Dr. Raúl Rojas
Advisor: Fritz Ulbrich

Berlin, November 6, 2017

Abstract

Academic research centers have lately become more interested in autonomous vehicles, particularly in pixel-wise semantic segmentation algorithms. The focus has been upon convolutional neural networks' accurate and efficient predictions of road scene objects as well as on the increase of pixel-wise annotated datasets which allow more accurate and well generalizing networks.
The aim of this study is to investigate how the modification of the existing efficient convolutional architectures can reduce the computational cost and become practically useful on low-power devices. A CNN was, therefore, developed for pixel-wise semantic segmentation of road images based on existing efficient architectures; namely SegNet, SegNet-basic and ENet for low-power devices, such as the Drive PX and Jetson TX series. All modified networks have been trained on mixed datasets made up of real and synthetic images of urban scenes, to produce better generalization and improved accuracy. These trained networks were tested both quantitatively on the Cityscapes validation dataset of 500 annotated images, and qualitatively, using a collection of personally taken photos from varied sources. The most accurate and better generalizing models were chosen for further training, which resulted in a model appropriate for pixel-wise labeling of images from different cameras, including the fisheye camera attached to the autonomous car belonging to the Free University of Berlin.

Declaration of Authorship

I certify under penalty of perjury, that the present work has been produced by me independently, all tools and sources are used as indicated and the work has not been submitted to any other institution for consideration.

Berlin, November 6, 2017 ________________________________

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass die vorliegende Arbeit von mir selbstständig angefertigt wurde, sämtliche verwendeten Hilfsmittel und Quellen angegeben wurden und die Arbeit an keiner weiteren Stelle zur Prüfung vorgelegt wurde.

Berlin, November 6, 2017 ________________________________

Acknowledgements

I would like to express my sincere gratitude to my supervisors Prof. Dr. Daniel Görhing and Prof. Dr. Raúl Rojas for their support, cooperation, and for providing me with all the necessary fascilities to carry out this research. I have the utmost appreciation for Prof. Daniel Görhing for his professional expertise and for the help provided during the development of this work.

I am very thankful to Fritz Ulbrich for his knowledge, friendly personal support and for the valuable and interesting discussions. It was very pleasant to be able to work in the friendly and productive atmosphere of the Dahlem Center for Machine Learning and Robotics. In particular, my deep appreciation to the researchers Zahra Boroujeni and Daniel Neumann for the valuable suggestions and for their personal and professional engagement in the writing of this thesis.

Last but not least, I would like to thank my fiancé Zhanna for her unending support, love and humor. This work would not be possible without you.

Contents

1 Introduction **1**

1.1 Related Work 2

1.1.1 CNNs for Image Classification 2

1.1.2 CNNs for Semantic Segmentation 3

1.2 Contribution 5

2 Background **6**

2.1 Neural Networks 6

2.1.1 Non-linearity or Activation Function 7

2.1.2 Training 7

2.2 Convolutional Neural Networks 8

2.2.1 Pixel-wise Semantic Segmentation 10

2.2.2 Dilated Convolutions 10

2.2.3 Batch-Normalization 10

2.2.4 Dropout and *SpatialDropout* 11

2.2.5 Transfer Learning 12

3 Methods **13**

3.1 SegNet, SegNet-basic and ENet 13

3.1.1 SegNet 13

3.1.2 SegNet-basic 15

3.1.3 ENet 15

3.2 Training Datasets 17

4 Experiments **21**

4.1 Networks: Training and Evaluation 23

4.1.1 SegNet 23

4.1.2 SegNet-basic 33

4.1.3 ENet 35

4.1.4 Chapter Summary 42

5 Conclusion 43

5.1 Future Work 44

Appendices 45

A Qualitative Results 45

B iIoU Scores on Cityscapes Validation Dataset 54

1 Introduction

Autonomous driving is described as a self-driving vehicle with a defined destination, which recognizes its environment and navigates without a human driver. Academic research centers, the automotive industry and software companies have lately become more interested in autonomous driving research. Low-priced mobile cameras, inexpensive hardware such as the graphics processing unit (GPU) and the availability of large datasets of annotated images are among some factors enabling the recent advancements in this field. In spite of all the recent developments, autonomous driving is still facing a number of challenges. One of which is the fast and accurate detection of all key objects from visual input signals on the road. Accurate detection is of vital importance for both passengers and pedestrians.

Object detection and localization in various scenarios through deep Convolutional Neural Networks (CNNs) is becoming progressively more efficient and accurate [24], [42], [34], [36], [15]. The accuracy records of the annual challenges in computer vision set by CNNs is good evidence of these advancements. These algorithms draw bounding boxes around the detected objects providing only coarse boundaries. To detect the exact boundaries of an object within an image, pixel-wise semantic segmentation algorithms were introduced. These algorithms classify every pixel in a given image to a certain semantic class. The issue of accuracy and efficiency also refers to pixel-wise segmentation algorithms, which are still in the development process.

In autonomous driving, the algorithm input is the image of the road from a camera installed on the vehicle. The output is also an image, usually with the same size as input, where every pixel represents a class i.e. car, road, tree, etc. (See, e.g., Figure 16). Pixel-wise segmentation of road images has the potential to make autonomous driving easier and safer. In this field, the ability to understand where the objects exact boundaries are or which parts of the road are suitable for driving is of great significance. The former is especially useful in the situation where the lane markings are not clear or do not exist. The time required from input to output called inference time is another important aspect of secure driving, especially when driving at higher speeds. The efficiency - decreasing the inference time - of the CNNs for semantic segmentation, especially for embedded and low-powered systems such as Drive PX, becomes more decisive when the number of cameras installed on a vehicle increases.

1.1 Related Work

Recently a number of successful techniques based on deep architectures have been developed to solve the problem of accuracy and efficiency in pixel-wise semantic segmentation [25], [4], [12], [22], [29], The instance segmentation, feature fusion, Recurrent Neural Networks (RNN) and Fully Convolutional Networks (FCN) are among several CNN-based techniques. Many of these approaches were developed using CNNs designed for image classification, which is the task of classifying a given image in a fixed set of defined classes.

1.1.1 CNNs for Image Classification

Neural networks (NNs), particularly deep CNNs, have progressively shown promising results in computer vision tasks such as image classification. Krizhevsky et al. (2012) [24] set a new record and won the annual Large Scale Visual Recognition Challenge (ILSVRC) 2012 [33] by outperforming the state-of-the-art (26.2% accuracy in top-5) in image classification with a large margin of 15.3% top-5 error. The success of this CNN named "Alexnet" demonstrated the appropriateness of CNNs for the classification of images, creating the basis for others to introduce new models. In 2013, Zeiler and Fergus [42] achieved a 11.7% rate by applying minor modifications to Alexnet: a smaller filter size (7×7 compared to 11×11) and smaller stride, among other changes. Building on this work, Clarifai [41] won the ILSVRC 2013 with a 11.2% error rate. A year later, Simonyan and Zisserman [34] set a new record (7.3%) by using deeper networks (19 layers) and an even smaller filter size of 3x3. Their 16-layer network named "VGG-16" (Figure 12) was later modified and applied to pixel-level segmentation tasks [25], [43], [2], [28]. The consecutive winners in the field were from Google in 2014 and Microsoft in 2015. Szegedy et al. [36] termed their architecture GoogLeNet, which consisted of 22 deep learning layers. This architecture set a record with 6.67% top-5 error while having 12 times less parameters than Alexnet. The network was structured using a new idea called "inception" module, which is a Network in Network (NiN). This module contained a max-pooling layer parallel to the convolutional layers with different filter sizes (1×1, 3×3, 5×5), which were then concatenated at the output (Figure 1).

Microsofts CNN called "ResNet" [15] won the ILSVRC 2015 classification task. It had 152 layers and reached 3.57%, thus outperforming human experts who achieved about 5% [33]. The novelty of this architecture was the "residual" block. The later reduced the redundant learning features by providing every following layer with the previous layers' input and output (Figure 2).

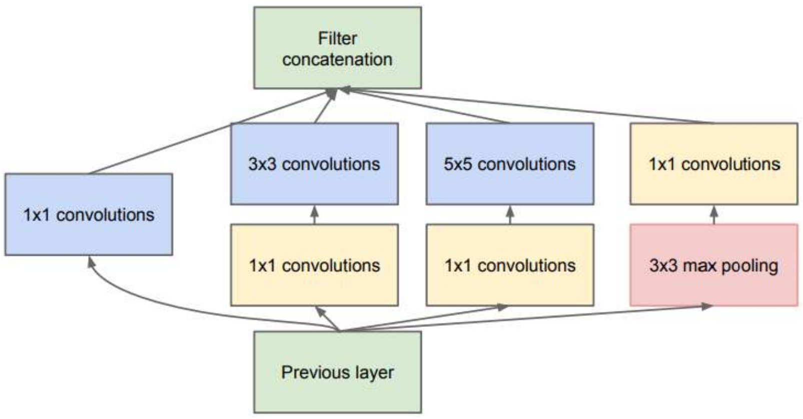


Figure 1: Inception module of GoogleNet. Figure extracted from [36]

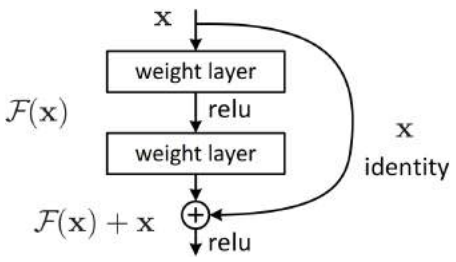


Figure 2: Residual block of ResNet. Figure extracted from [15]

1.1.2 CNNs for Semantic Segmentation

Long et al. [25] introduced a new technique to train Fully Convolutional Networks (FCN) end-to-end by adapting image classification networks like Alexnet [24], VGG net [34] and GoogLeNet [36] to pixel-wise segmentation networks. By replacing the fully connected layers of these networks with convolutional layers and by fine-tuning their learned representations, Long et al. achieved state-of-the-art performance both in accuracy and efficiency. They termed it *convolutionalization*, illustrated in Figure 3. This enabled the production of coarse output maps, which were then upsampled with the help of fractionally strided convolutional layers[1] to produce a dense labeled image, where every pixel represented a class. This model has become a building block for many CNNs using pixel-wise segmentation.

The early decoder networks in encoder-decoder architectures contained only one layer as shown in lower part of Figure 3 marked with "pixelwise prediction" [25], [43], [9]. Noh et al. [28] enlarged the decoder network to the size

[1]Also known as fractionally strided convolution, transposed convolution, upconvolution and deconvolution

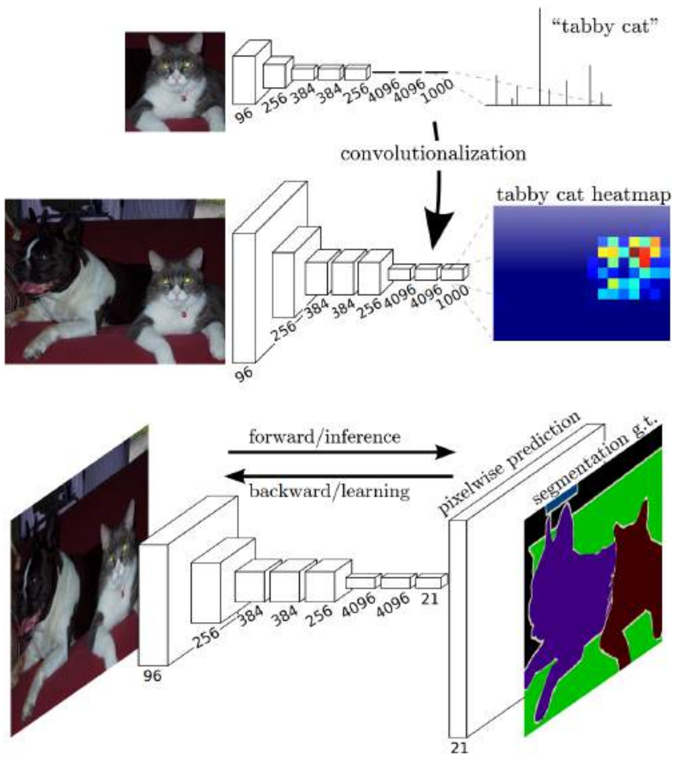


Figure 3: An example of convolutionalization of a network designed for image classification into FCN for pixel-wise classification. Upper: Replacing fully connected layers with convolutional layers outputs a heatmap. Lower: Adding 1×1 convolutional layer (decoder) with 21 filters to predict classes pixel-wise. Figure extracted from [25].

of the encoder network and added Batch Normalization (BN) layers to every convolutional layer of the network [18]. This network (Figure 4) achieved state-of-the-art accuracy in PASCAL VOC 2012 [11] segmentation benchmark. Later, Badrinarayan et al.[2] inspired by Ranzato et al. improved the decoder network, creating a network named "SegNet". They connected indices of max-pooling layers' max locations of encoder network to upsampling layers of the decoder network (Figure 13). By eliminating the fully connected layers from the original VGG-16 architecture, Badrinarayan et al. greatly reduced the number of parameters in the network from 134 million to 14.7 million.

To the best of my knowledge, all the CNNs for pixel-wise semantic segmentation were modified versions of image classification CNNs. That is until Yu and Koltun[40] designed a CNN specifically for semantic segmentation of dense images with dilated convolutional layers. They used dilated convolutional layers for their network module to increase the receptive fields of

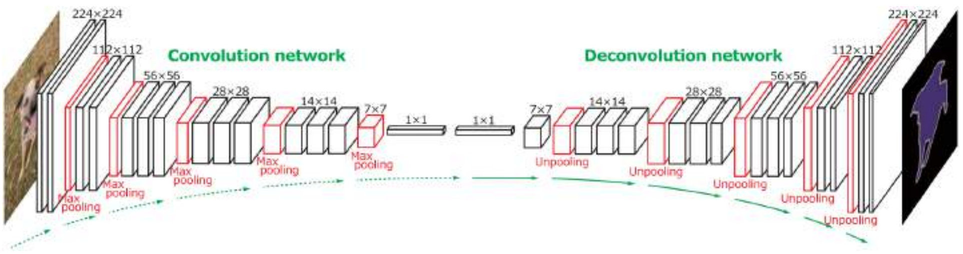


Figure 4: Encoder-decoder network proposed by [28] based on VGG-16. Figure extracted from the paper.

kernel filters so that the wider, global context could be taken into consideration[2]. The dilated convolutions also played an important role in designing an efficient network termed *ENet* introduced by Paszke et al. [29]. Inspired by ResNet [15] and [37] they introduced a new encoder-decoder neural network architecture for real time pixel-wise semantic segmentation with competitive accuracy.

1.2 Contribution

The aim of this work is to develop a CNN for pixel-wise semantic segmentation of road images based on existing efficient architectures; namely SegNet, SegNet-basic and ENet for low-power devices such as the Drive PX and Jetson TX series. All modified networks have been trained on the same mixed dataset of real and synthetic images of urban scenes to produce better generalization and improve accuracy. In contrast to ENet, the number of feature maps of SegNet architecture has been cut down in size to increase the number of processed images per second. Further, new layers were introduced to this reduced version of SegNet. ENet, which was originally trained on CityScapes, has been fine-tuned for smaller input images to decrease the inference time on Drive PX.

Another central aspect of this work is to train these networks and test them on a generalization set, that contains various sources of images not used in the training phase. The most efficient and better generalizing models (ENet and SegNet-reduced v1) were chosen for further training on bigger datasets, which are discussed in Section 4. For this Caffe [20], an open source deep learning framework was employed.

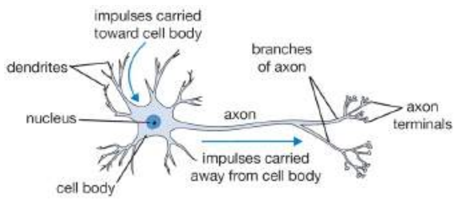
[2]Dilated convolutions are discussed more in depth in Section 2

2 Background

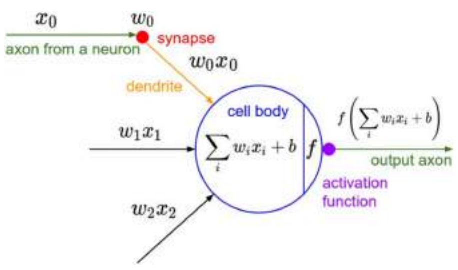
This chapter provides an insight into deep learning with particular focus on the terminologies and concepts used in this work. To clarify the workings of Convolutional Neural Networks (CNNs), an explanation of their main components is presented.

2.1 Neural Networks

In 1958 Rosenblatt introduced *perceptrons* as the "first learning neural computer" - a computer driven learning alternative to biological learning [31]. As Figure 5b illustrates, a neuron receives inputs $x_1, ..., x_n$ and multiplies these with corresponding weights $w_1, ..., w_n$. These multiplications are then summed and added to a bias. Before the neuron gives an output, an activation function is applied. First, the perceptrons were linear (binary) classifiers, as the *Heavyside* step function was used as an activation function, which outputs either 0 or 1. Later, to address the limitations of perceptrons, Multi-Layer Perceptrons (MLP), also known as Neural Networks (NN), with non-linear activation functions were introduced [13], [14].


impulses carried toward cell body
branches of axon
dendrites
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

(a) A drawing of a biological neuron.



(b) A mathematical model of a neuron, also called a *unit*

Figure 5: Images taken from [21]

NNs are a collection of artificial neurons stacked in layers with connections between them (Figure 6). All of the neurons in the layer l and $l-1$ are connected with each other, while the nodes in the same layers are not connected. The number of nodes in the output layer usually equals the number of object classes in the training dataset.

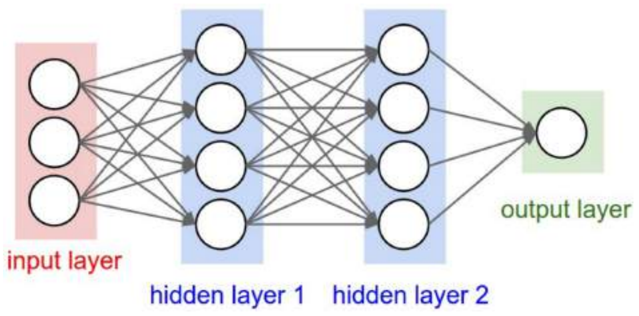


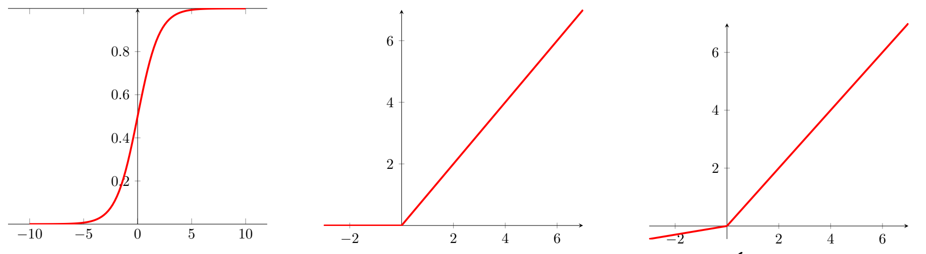
Figure 6: A 3-layer neural network, with 2 hidden layers and 1 output layer. Figure extracted from [21]

2.1.1 Non-linearity or Activation Function

Activation Functions are integral components of Neural Networks. An activation function takes a single number as an input and performs a mathematical operation. There are several types of non-linear activation functions such as the Sigmoid, the Hyperbolic Tangent (Tanh), the rectified linear unit (ReLU) [14], the Leaky ReLU, and the Parametric ReLU (PReLU) [16]. In comparison to Sigmoid and Tanh, which *squashe* the input number into a range of [0;1] and [-1;1], respectively, the ReLU activation function computes the function $f(x) = \max(0, x)$ and requires less computation as it does not contain exponentials (Figure 7). It also accelerates the convergence of stochastic gradient descent (SGD), due to its non-saturating property [24]. Since 2010 the ReLU has been widely used for deep NNs [27], however, it removes all the negative information coming from previous neurons and outputs zero instead. This state is called *dead ReLU* and can be irreversible [26]. Leaky ReLU fixes this problem by multiplying the negative inputs with a small constant number. As Figure 7 shows, the function has a negative slope, where a is a constant (usually 0.01). To take the idea one step further, He et al. [16] brought the parameter a into deep learning by making it trainable.

2.1.2 Training

To train the NNs, an algorithm called backpropagation is commonly used [32]. The algorithm first performs the *Forward Propagation* and then *Backward Propagation*. In the forward propagation, the data is propagated through the layers until it reaches the output layer. After computing the predictions, the loss function (also called objective function) computes the error comparing predictions with the ground truth (the right values of the output layer). The gradient of the loss function is then propagated backward to the first hidden layer. This step updates the weights of the neurons to reduce the



$$f(x) = \frac{1}{1+e^{-x}} \qquad f(x) = \max(0, x) \qquad f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

Figure 7: Sigmoid, ReLU and Leaky ReLU. PReLU has the same formula as Leaky ReLU, however in PReLU the parameter a is *trainable* whereas in Leaky ReLU it is a constant.

error at the output. Computing the gradient gives the objective direction at which the rate of error reduction is the highest. The step size in that direction defines the *learning rate*, which is an important parameter in NNs training. The learning rate is set at the beginning of a training, although, it can be reduced during the training as it is done in the trainings described in Section 4. To make the convergence faster, the backpropagation algorithm is used with momentum. Momentum "pushes" the learning rate, if the gradient from the previous step shows the same direction in the current step, thus increasing the sizes of the steps taken towards convergence.

A training is measured in epochs. An *epoch* is the amount of training when all samples of the training dataset have been "seen", in other words, all samples have been forward and backward propagated. The opportunity provided by GPUs to do parallel computing allows forward and backpropagation of many samples at once. This number of samples is called *batch size* and is limited by the GPU memory. When the number of used samples equals the batch size, an *iteration* is reached. For instance, if the number of training samples is 300 and the batch size is 10, it will take 30 iterations to achieve an epoch.

2.2 Convolutional Neural Networks

NNs have been successfully used for classifying smaller images, such as handwritten digits with a size of 32×32 [5], [7]. However, when the size of the input image increases, the number of the parameters grow to a point where it becomes impractical to train these networks. This problem was solved by deep CNNs inspired by animal visual cortex [17]. The NN algorithms, such as backpropagation, non-linear functions and data structures, such as neurons, are also used in CNNs. The later is a type of NN, which is designed specifically for classifying images. They are constructed from different layers, such as convolutional, pooling and Full Connected (FC). The convolutional layer is the main building block of CNNs. It consists of trainable filters, which are convolved with the input image to output feature maps. The main parameters of the convolutional layer are the filters' size, number, stride, and padding. Padding defines the number of pixels added to sides of the input, while the stride indicates the intervals at which the filter should be applied.

The Figure 8 shows the filters learned by the first layer of Alexnet [24]. The first layers usually recognize local features such as straight lines, curves, corners and edges.

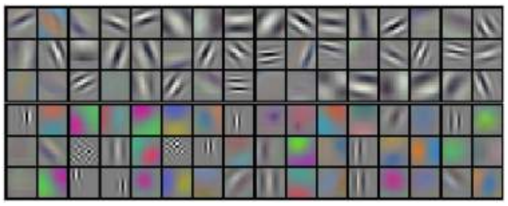


Figure 8: All 96 filters learned by the first layer in Alexnet. Each of these filters has a size of 11 × 11 and a stride of 4. Figure extracted from [24].

Filters in deeper layers use the low level features of previous layers to recognize objects such as circles, text, faces, etc., which in turn enable the last layers to recognize more complex objects such as cars, buildings, trees, and animals.

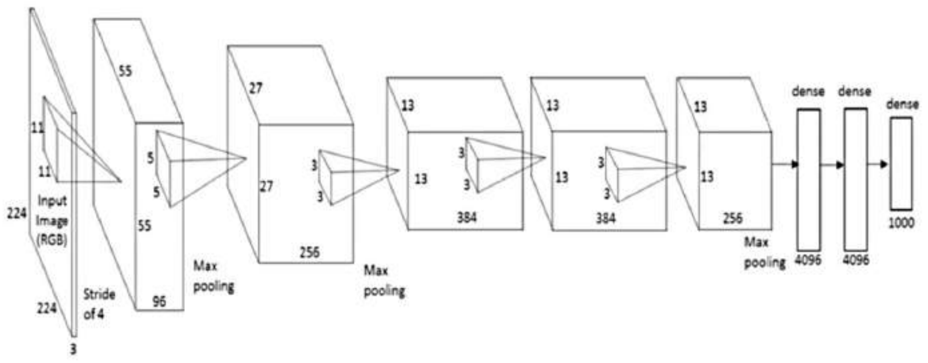


Figure 9: Illustration of Alexnet architecture, which was trained on ILSVRC-2012 dataset [33] to classify 1000 difference object classes. Figure extracted from [1].

Another commonly used layer type in CNNs are the pooling layers. They are used to reduce the spatial size of feature maps to reduce the computation in the training (Figure 9). A pooling layer defines the filter size, padding, stride and pooling type. The most common type of this layer is *max-pooling*, which performs a max operation in the receptive field. Usually max-pooling layers have a filter size of 2×2 with a stride of 2, meaning the maximum value in each 2×2 pixel block of input images is taken, thus halving the images size along the width and the height. The pooling layers are usually placed after or parallel to convolutional layers.

Fully connected layers, which are essentially regular NNs, are used in CNNs designed for classification tasks. As shown in Figure 9, the last 3 layers are fully connected and the very last layer has 1000 neurons, which decide the probability of a certain object class.

2.2.1 Pixel-wise Semantic Segmentation

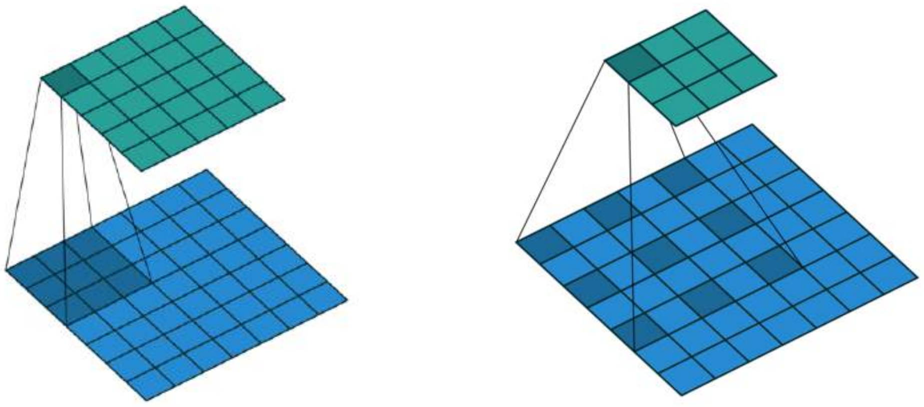
Pixel-wise semantic segmentation is the problem of assigning a certain label to every pixel in a given image, where each label corresponds to an object class, such as road, sidewalk, vehicle, building, and pedestrian. In CNNs designed for pixel-wise semantic segmentation, the last layer is a convolutional layer that produces feature maps. The number of feature maps is the number of class objects in the training dataset. In the Section 3, three architectures for pixel-wise segmentation are presented in more detail.

2.2.2 Dilated Convolutions

Regular convolutions have exclusively contiguous filters, which means that there are no gaps between their cells. Yu and Koltun introduced an extension to the regular convolutions called dilated convolution [40]. This new layer has only one additional parameter called *dilation*. Dilated convolutions are not contiguous, but have gaps between the filter cells. As illustrated in Figure 10b, 1-dilated convolution has 9 cells like the regular convolution, however the receptive field of the dilated convolution is not 3×3, but 5×5. This extension allowed filters to have an increased receptive field, without additional computation.

2.2.3 Batch-Normalization

Training deep NNs required a careful choice of learning rate and weights initialization. This was due to the fact that the distribution of layers input changes during the training, because of the changes in previous layers' parameters. To address this problem Ioffe and Szegedy introduced the batch normalization layer, which has quickly become popular in CNNs [18].



(a) A regular 3×3 convolution. **(b)** 1-dilated 3×3 convolution.

Figure 10: Illustration of a 1-dilated convolution. Dark blue areas are where the receptive field works. Figures extracted from [8].

By putting batch normalization layers after the convolutional layers the researchers performed the normalization of the layer inputs in each mini-batch. This enabled faster convergence through the allowance of higher learning rate and made the model more robust to weight initialization. By adding these layers to a network they achieved more accurate results than the original network while using 14 times less training steps.

2.2.4 Dropout and *SpatialDropout*

Overfitting and slowness pose serious obstacles to the successful operation of large deep neural networks. To overcome these obstacles, dropout is usually utilized [35]. It is a common regularization technique used during the training, that prevents NNs from overfitting to the training data by randomly dropping out units of NNs along with their incoming and outgoing connections. By dropping the connections of random nodes during the training, the rest of the nodes become immune to the weight changes of dropped nodes, thus making the model more robust (Figure 11). This technique results in not only a better generalizing model, but also improves the training efficiency, which allows faster prototyping. The success of NNs performances in various tasks i.e. in visual recognition is thus conditioned by the overthrow of overfitting.

When a standard dropout is used in a CNN, random pixels in feature maps are dropped out. Tompson et al. presented a new dropout technique for CNNs called *SpatialDropout* [38]. In contrast to regular dropout, where pix-

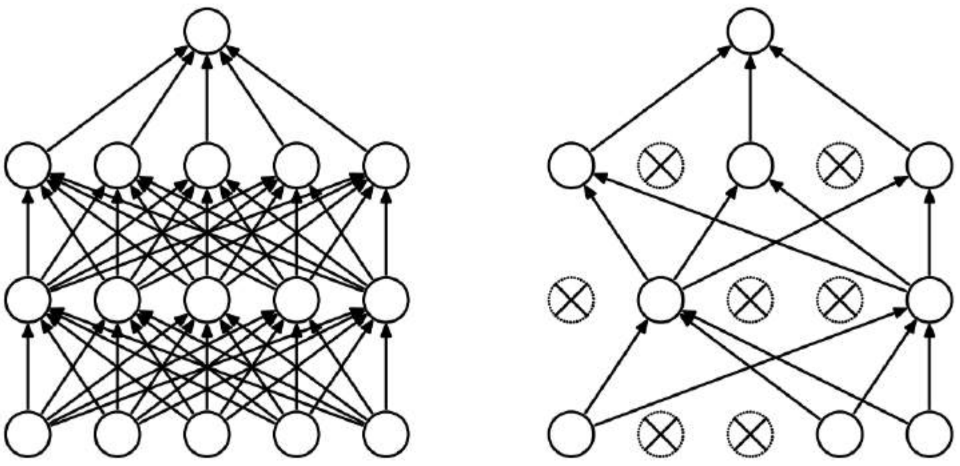


Figure 11: Left: A regular 2-layer NN. Right: After randomly dropping out neurons in a mini-batch during the training. Figure extracted from [35].

els are dropped, this technique drops entire feature maps during the training. Replacing the regular dropout in networks with *SpatialDropout* showed significant improvement in accuracy [38], [29].

2.2.5 Transfer Learning

Yosinksi et al. showed that the training of CNNs on natural images caused the networks to learn similar features regardless of the dataset and cost function [39]. These learned features are in the first layer and occur so often, that researchers called them general features, also known as low layer features. They called the last layers' features *specific*, also known as high level features, because these features specifically work on the dataset trained. This means, that copying low-level features from a model trained on one dataset to another model can save training time, because otherwise the neurons must learn these low-level features from scratch. This process of transferring the weights of one model to another one and then training it on a new dataset is called *fine-tuning*. This is especially useful, because training on a small dataset can easily result in overfitting and in improper learning of low-level features. When using this technique on a new dataset, the learning rate should be smaller than the rate used during the original training, as the weights should merely adapt to the new dataset and do not need to be changed too much.

3 Methods

3.1 SegNet, SegNet-basic and ENet

In this study, the trained CNNs were based on three architectures: SegNet, SegNet-basic and ENet. Two different datasets - Cityscapes and GTA5 - were used for training and evaluation. In the following text, these architectures and dataset are introduced and elucidated.

3.1.1 SegNet

SegNet is a pixel-wise segmentation CNN designed by Badrinarayanan et al. [2]. They based SegNet on a CNN called VGG16, which was originally designed for the image-classification task. VGG16 is a 16-layer CNN (Figure 12) created by the Visual Geometry Group (VGG) [34], which uses only convolutions with a kernel size of 3×3 (pad and stride of 1) and max-pooling layers with a size of 2×2 and a stride of 2. All 16 convolutional layers are followed by a ReLU activation function. After each max-pooling layer, the number of feature maps doubles until the image size is downsampled to 28×28. The last max-pooling layer, that produces 512 feature maps with a size of 7×7, is followed by three Fully Connected (FC) layers: Two FC layers with 4096 feature maps and one with 1000 (number corresponds to 1000 classes of the ILSVRC-2012 dataset). The VGG16 was trained on the ILSVRC-2012 training dataset (1.3 million images) for about two weeks using a computer equipped with 4 NVIDIA Titan Black GPUs. The trained model achieved a top5-error rate of 7.4% on the test dataset and was uploaded to the Oxford University's VGG group website[3]. To implement the network and its training, the VGG group used a modified version of a deep learning framework called Caffe [20].

Badrinarayanan et al. adapted the VGG16 network to a semantic segmentation network by removing the FC layers along with the last max-pooling layer and adding a decoder network instead. As illustrated in Figure 13, the decoder network is the mirrored version of the encoder network, where instead of max-pooling layers, the decoder uses upsampling layers, which are connected to the corresponding max-pooling layers in the encoder network.

Decoder's purpose is to map the encoder's low resolution feature maps to higher resolutions up until the size of the input. The connection allows the decoder to receive the max locations (pooling indices) of max-pooling

[3] `http://www.robots.ox.ac.uk/~vgg/research/very_deep/`

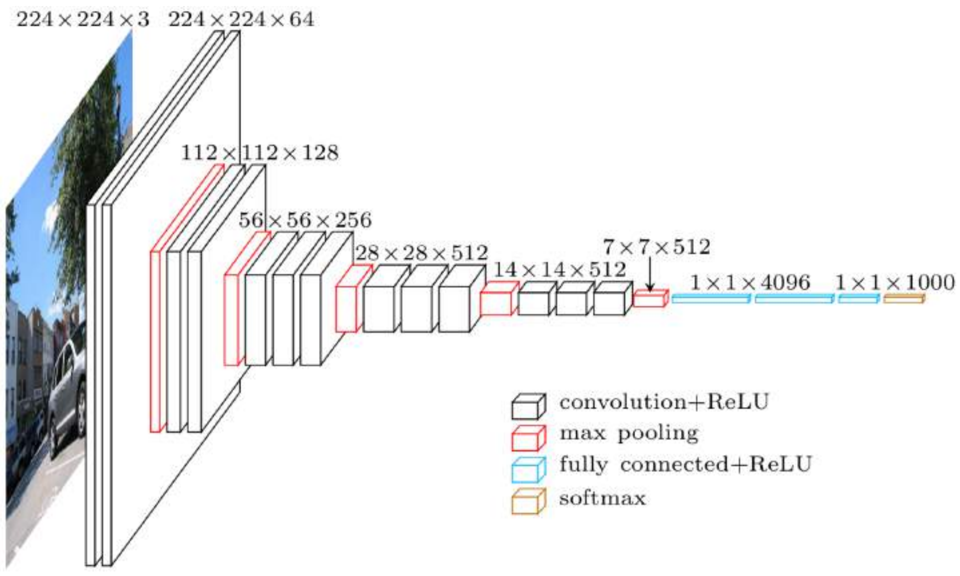


Figure 12: VGG16 architecture. Figure extracted from Matthieu Cord's website (`http://webia.lip6.fr/~cord/pdfs/news/TalkDeepCordI3S.pdf`)

layers to perform a non-linear upsampling. In the decoder, the convolutional layers with a trainable filter bank densify the feature maps. After the last convolutional layer, a Softmax classifier with cross-entropy loss function is placed. The number of feature maps of the last convolutional layer is the number of objects needing to be classified. It should also be noted, that in the SegNet Badrinarayanan et al. added a batch normalization layer after every convolutional layer. Removing the FC layers from VGG16 and creating connections between the encoder and the decoder allowed them to reduce the number of parameters from 134 million to 14.7 million.

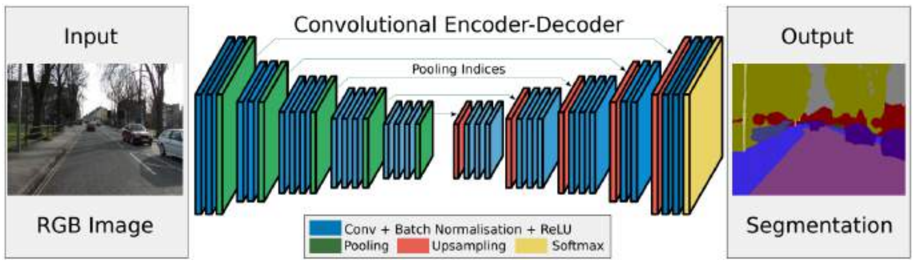


Figure 13: SegNet architecture [2] based on VGG-16. Figure extracted from the paper.

For training SegNet, they used the pre-trained weights of VGG16 in the encoder network of SegNet. Using the learned low features of the VGG16 and fine-tuning the high-level features on a small dataset called CamVid (367 images with a size of 360×480) enabled them to achieve the state-of-the-art accuracy in pixel-wise segmentation, while keeping the network efficient both in terms of inference time and memory.

3.1.2 SegNet-basic

SegNet-basic is a smaller version of Segnet with 4 encoders and 4 decoders, each containing a convolutional layer with 64 feature maps. All these convolutional layers are followed by batch normalization layers (See Table 2 for comparison). ReLU activation layers are only present in the encoder network. As in SegNet, the max-pooling and upsampling layers are connected with each other. For context awareness and smooth segmentation all convolutions have the same kernel size of 7×7.

3.1.3 ENet

Paszke et al. [29], inspired by ResNet [15], designed a pixel-wise semantic segmentation CNN for real-time applications following the common encoder-decoder technique. Unlike SegNet, where the decoder is a mirror of the encoder, the decoder of ENet, as illustrated in Table 1, is smaller than the encoder. Like in SegNet, the max-pooling indices in the encoder are saved and used in the decoder for upsampling.

The initial layer consists of a 3×3 convolutional layer with a stride of 2, which outputs 13 feature maps. Parallel to this layer, a max-pooling layer is placed outputing 3 feature maps, one for each color channel (RGB). As illustrated in Figure 14, the outputs of these layers are then concatenated, making 16 feature maps with half the input size of the originals. This early downsampling is done for optimization purposes. The rest of the network consists of bottleneck modules.

A bottleneck layer consists of three convolutional layers. As illustrated in Figure 15b, the first convolutional layer has a kernel size of 1×1, which is meant to reduce the number of feature maps passed to the following *main* convolutional layer. After the main convolution another 1×1 convolution is placed to increase the number of feature maps. The *conv* layer is either a regular, dilated or asymmetric convolution. If the module is in the decoder and is of the upsampling type, then the *conv* layer is a 2×2 deconvolutional with a stride of 2. In asymmetric bottleneck modules the main convolution consists of two asymmetric convolutional layers: 5×1 followed by an 1×5. These two asymmetric layers have a computational cost of one 3×3 convolution layer, however they provide an increased receptive field. Dilated convolutions with exponentially growing dilation parameters (Table 1) are

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

Table 1: ENet architecture. In this example the input image has a size of 512×512. The decoder network begins with the first upsampling module. C stays for the number of feature maps, which corresponds to the number of object classes needing to be classified. The table is extracted from [29].

also directed at global context integration. As Paszke et al. claim, adding dilated layers resulted in a 4% increase in accuracy on the Cityscapes dataset without additional computational costs.

Another type of module used by Paszke et al. is the downsampling bottleneck module (Figure 15a). One downsampling module was placed right after the initial layer to halve the size of the feature maps at an early stage. They suggested that the first layers did not directly participate in classification, but acted as feature extractors, that were supposed to prepare feature maps for later layers to classify. Accordingly, changing the number of feature maps in the initial layer from 16 to 32 did not affect the network accuracy and thus confirmed the suggestion.

Further, between all convolutions, a batch normalization and a PRelU activation layer were placed. For regularization, the bottleneck modules included a *SpatialDropout* layer. It had a value of $p = 0.01$ before and $p = 0.1$ after the *bottleneck2.0*. The choice of the PReLU was explained by experiments, which showed ReLU to be detrimental to the accuracy in the ENet.

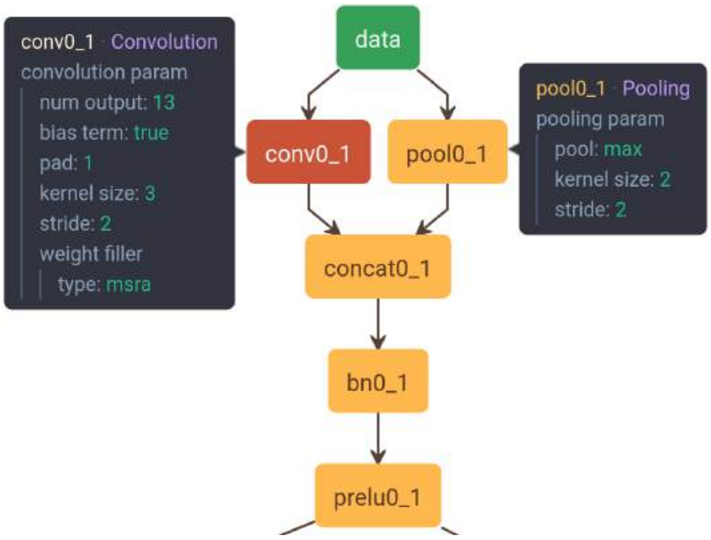


Figure 14: ENet initial block visualized with *Netscope*.

As illustrated in Figure 15, after the convolutions and regularization, the feature maps element-wise added to the main brunch of the network. The *element-wise addition* performs an element-wise addition of its inputs feature maps' values.

The last deconvolution (*fullconv* in Table 1) did not receive the initial module's max pooling indices, because it produces only 3 channels (the color channels of the input data), while the last deconvolution has more than 3 channels, because the number of object classes in a training dataset is usually higher than 3.

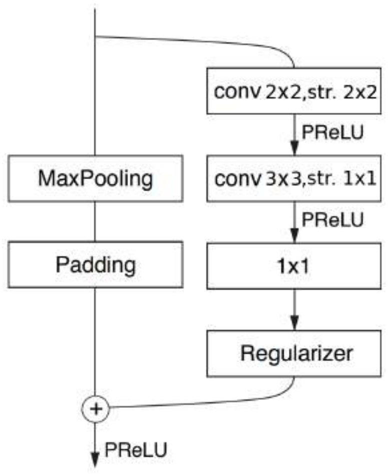
The ENet architecture can be visualizied with a web-based network visualizing tool called Netscope[4]. The training architecture in prototxt format is available in a Caffe implementation of ENet[5]

3.2 Training Datasets

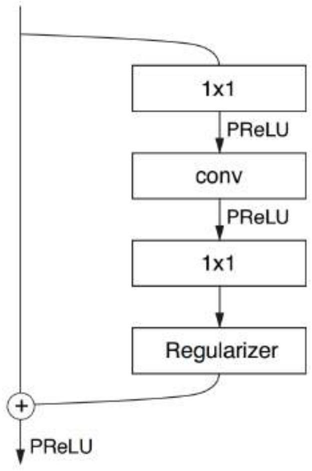
Two different datasets - a real-world called Cityscapes [6], and a synthetic dataset obtained from a game called GTA5 [30] - were used for the training of CNNs.

Cityscapes: The Cityscapes dataset contains 5000 finely and 20,000 coarsely annotated images of urban street scenes from 50 different cities in Germany

[4]`http://ethereon.github.io/netscope/`
[5]`https://github.com/TimoSaemann/ENet/tree/master/prototxts`



(a) Downsampling bottleneck module. The max indices of max-pooling layers are passed to the corresponding upsampling layer in the decoder.



(b) *conv* is either a regular 3×3, dilated convolution or a 2×2 deconvolution layer.

Figure 15: ENet bottleneck modules. The plus sign stays for element-wise addition, which is followed by a PReLU activation function. Figures are taken from [29] and presented with modifications

and Switzerland. Out of 5000 images with a size of 2048×1024, 2975 images are for training, 500 are for validation and 1525 for testing purposes. All images are annotated over 30 classes, from which only 19 were used for the trainings in this work. Ignored classes are annotated with the number 2 in Figure 16. The 500 images from the validation dataset were used to measure the accuracy of trained networks. The Github repository of the dataset[6] provides, among other scripts, a script for the calculation of the accuracy: the intersection-over-union (**IoU**) and instance-level intersection-over-union (**iIoU**) scores per class and per category (details are discussed in Section 4). The corresponding pixel-wise annotations of the remaining 1525 test images are not publicly available. The accuracy on test dataset is calculated when submitted to the server of Cityscapes dataset.

GTA5: Richter et al. provide a dataset of 24966 pixel-accurate labeled frames of urban street scenes extracted from a photo-realistic open-world computer game called Grand Theft Auto V (GTA5) [30]. The images are distributed in 10 equal parts and are annotated over 19 classes, which are compatible with Cityscapes class definitions. The frames were extracted in varying weather conditions and from different times of day (Figure 17).

[6]`https://github.com/mcordts/cityscapesScripts`

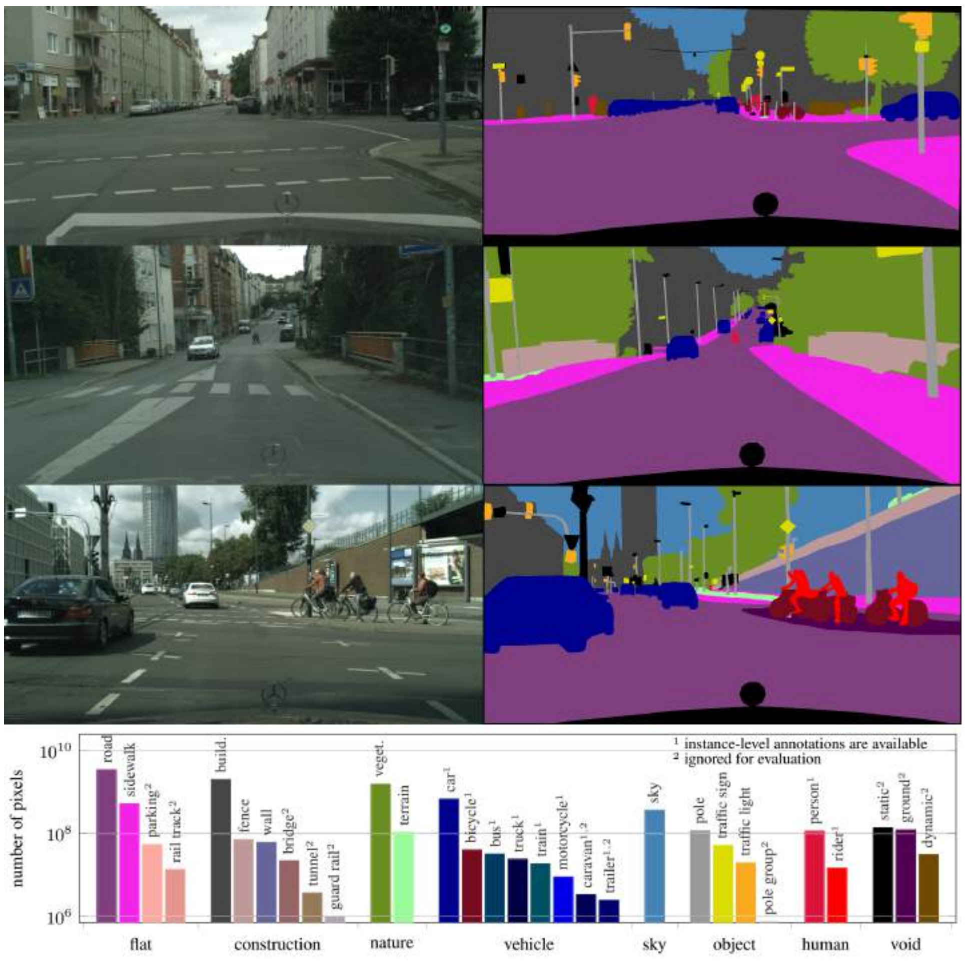


Figure 16: Upper: Examples of road images from the Cityscapes train dataset and their annotations. Lower: Corresponding colors of classes grouped in categories and number of pixels of each class (y-axis). Lower part is extracted from [6].

To make the combination of GTA5 and Cityscapes possible, I converted the pixel values of GTA5 images to be consistent with the pixel values of the Cityscapes dataset. The source images had a resolution of 1914×1052 which have been cropped to 1914×957 to have the same aspect ratio of 2:1 as the Cityscapes dataset. To minimize the number of removed annotated pixels, I removed the lower parts (1914×95) of all images, which contained the ego of the vehicle (annotated with black in Figure 17) and were not a part of 19 training classes.

During the training both datasets have been downsampled to 800×400 and 512×256 for the ENet and SegNet, correspondingly. For Enet, this downsampling allowed a batch size value of 4 (training with a smaller batch

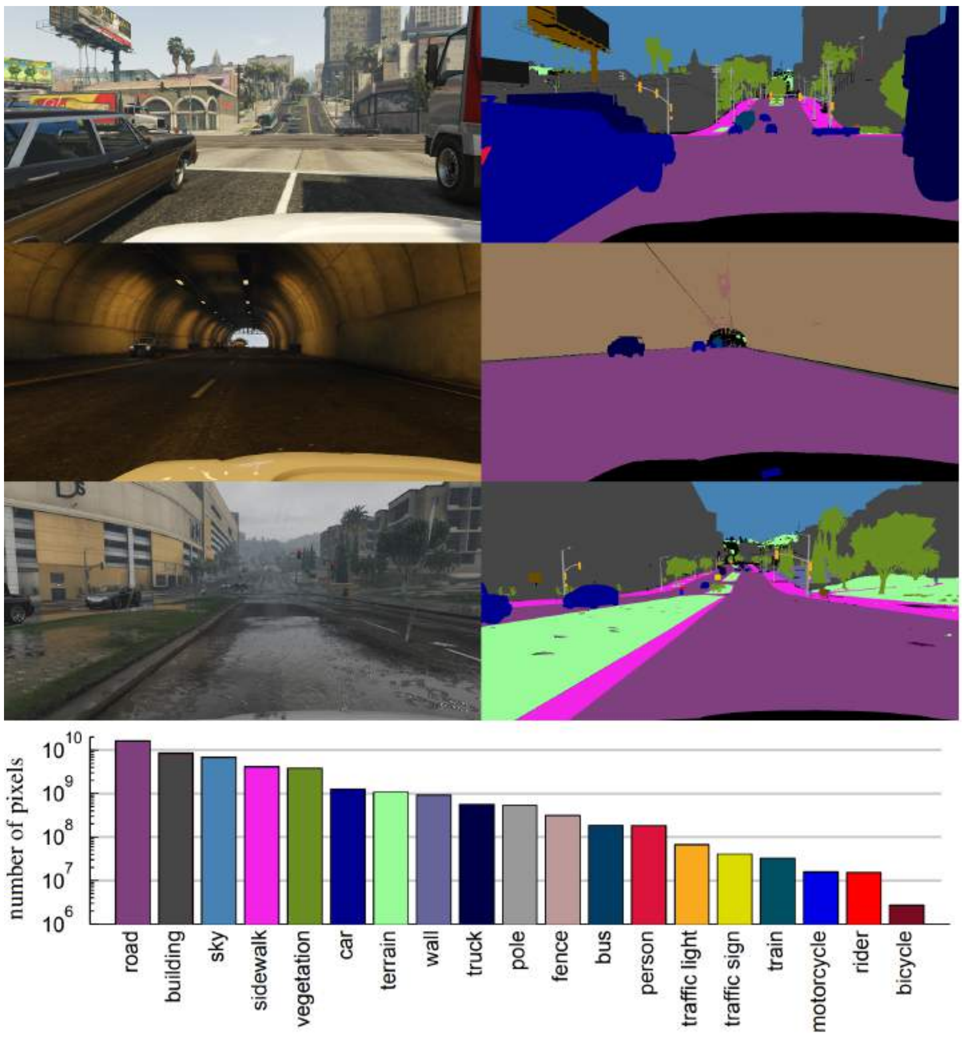


Figure 17: Upper: Examples of road images from the GTA5 dataset and their annotations used for training. Lower: Corresponding colors of classes and number of pixels of each class (y-axis). Lower part is extracted from [30].

size showed no convergence) and an increased number of processed images per second on Drive PX. For SegNet, this allowed a practical inference time of 560 ms on Drive PX.

4 Experiments

This chapter draws on the details of CNN trainings on the datasets discussed in Section 3. It includes tables and figures showing the differences within trained networks, their training parameters and qualitative and quantitative results. Finally, a brief summary of the main findings as well as a comparison of all trained networks, in terms of efficiency and quantitative results, is presented.

For this study, CNN architectures - ENet, modifications of SegNet and SegNet-basic - were trained on Cityscapes and GTA datasets to find a more efficient and better at generalizing network. ENet and a modified version of SegNet called *SegNet-reduced v1* were first trained on Cityscapes dataset, then on the combined dataset of Cityscapes and GTA parts from 1 to 9. The SegNet-basic and other modifications of SegNet were trained on Cityscapes dataset complemented with GTA part 1.

All networks were trained and evaluated on the same hardware and software. The main components of hardware were two nvidia GTX 1080 Ti GPUs with 11 GB of GDDR5 memory, a 12 core intel Xeon E5-1650 CPU and 32GB of RAM. For training the CNNs two slightly modified versions of Caffe deep learning framework were used. Both versions were compiled to use CUDA 8.0, CuDNN 5.1 and OpenBLAS with LAPACK 0.2.19-1.

All values of **inference time** of networks mentioned in this work refer to the time required by the CNN to label all the pixels of an input image on Drive PX. Drive PX is a mobile computer produced by Nvidia Corporation to provide autonomous driving functionality. It is equipped with two Tegra X1 SoCs (System on a Chip), that feature four ARM Cortex-A57 cores, four ARM Cortex-A53 cores and two Maxwell-based 256 core GPUs.

Benchmarking: For quantifying the accuracy of semantic segmentation, the most common metric, the Jaccard Index (also known as PASCAL VOC intersection-over-union [11] per-class metric) was employed. In this formula

$$IoU = \frac{TP}{TP + FP + FN}$$

TP, FP and FN are the numbers of true positive (intersection), false positive and false negative pixels, respectively, of a class over the evaluation dataset. Unlike the global accuracy measure, which merely calculates correctly classified pixels, this metric penalizes the false positive and false negative pixels. For the evaluation of the predicted labels Cityscapes implementation[7] of this

[7] `https://github.com/mcordts/cityscapesScripts/tree/master/cityscapesscripts/evaluation`

metric was used.

Additionally, to address the bias of the IoU measurement regarding larger object instances such as road, sidewalk, building, vegetation and sky, the authors of Cityscapes dataset offered instance-level intersection-over-union metric (iIoU)

$$iIoU = \frac{iTP}{iTP + iFP + iFN}$$

where iTP, iFP and iFN are weighted counts for classes: *person, rider, car, truck, bus, train, motorcycle* and *bicycle.* In the Cityscapes dataset these classes are annotated with instances and deemed to be crucial elements in road scene understanding. They weighted the contribution of each pixel by calculating the ratio of a given class' average instance size to respective ground truth instance size.

This implementation was used to calculate the IoU and iIoU scores of self-conducted training. This evaluation algorithm ignores pixels of classes that are annotated with number 2 in Figure 16.

Unlike the Cityscapes validation dataset, the annotations of test dataset are not publicly available and IoU scores of predicted images of test dataset are given on Cityscapes website upon submissions. Therefore evaluation and comparison of original architectures and their modified versions was done on validation datasets. The iIoU scores of all trained models can be found in appendix B.

Generalization: No pixel-wise annotated dataset of diverse road images compatible to the Cityscapes dataset was found to quantify the generalization of the trained networks. Instead, qualitative results were manually assessed through comparison of network outputs. For this objective 50 images from 4 different sources were used: Cityscapes, a GoPro HD Hero2 camera, the front fisheye camera of the Free University of Berlins autonomous car called MIG (Made In Germany), and the camera of a private mobile device (Samsung Galaxy S4). The pictures were taken at the campus area of FU Berlin (except Cityscapes) and were not pixel-wise annotated, hence the generalization was not quantified. The images from GoPro2 and the fisheye camera were cropped in the center with the aspect ratio of 2:1 to avoid the curvilinear effect of the fisheye lens and be compatible with the network input aspect ratio. This was done, because the predictions of rectified images of fisheye cameras turned out to be of poorer quality than the same pictures cropped in the center. The Galaxy S4 camera images were also cropped vertically to maintain the aspect ratio of 2:1. This set of 50 images is referred to as the generalization set.

4.1 Networks: Training and Evaluation

Class balancing: When the number of pixels of different classes vary greatly (Figur 16), the class balancing method called *median frequency balancing* is recommended [2]. Unlike the *natural class balancing*, where every class has the same weight in the loss function, the *median frequency balancing* assigns corresponding weight to every class depending on the frequency in which the class occurs in the train dataset.

The weight is calculated by dividing the median of class frequencies in the train dataset by the class frequency. Class frequency is the ratio of the number of pixels of a class to the sum of all images pixels, in which the class is present. A script was written to measure the class weights according to the following formula taken from [10].

$$\alpha_c = median\_freq/freq(c)$$

When the number of pixels of a class in the dataset is smaller compared to the other classes, then its weight gets a higher value in loss function. Conversely, the more frequent occurring classes get smaller weights than 1 and the highest frequency class (in this work the *road* class) gets the smallest weight.

Median frequency balancing was selected in this work as it *encourages* the network to learn less frequently occurring classes such as bicycle, rider, motorcycle, train, traffic light etc. which are key objects on the road scene. For this reason, the weights were calculated using the written script and assigned to every class in the loss function.

4.1.1 SegNet

All SegNet based networks were trained for about 150 epochs with the initial learning rate of 0.001. The learning rate was reduced by a factor of 10 after reaching convergence, usually about every 50 epochs. As in [2], the SGD solver [3] and the momentum of 0.9 were employed in all trained models. For initializing the weights, the *webdemo*[8] weights were used for all SegNet trainings. These weights were first trained by [2] over 11 classes on 3433 publicly available labeled images, which were combined from 4 different datasets including Cambridge-driving Labeled Video Database (CamVid) [19]. Next, the authors of SegNet used an additional publicly unavailable dataset to

[8]`https://github.com/alexgkendall/SegNet-Tutorial/blob/master/Example_Models/segnet_model_zoo.md`

train the network further[9]. The resulting weights are available at website[10] for demonstration purposes, hence the name *webdemo*.

651 ms were required to process one image with a size of 512×256 on Drive PX using the original SegNet architecture. After merging BN layers with the convolutional layers using the script provided by SegNets Github repository, the time reduced to 561 ms. According to the online benchmark of Cityscapes, SegNet has a mean IoU score of 56.1%[11] on **test dataset**.

To get the IoU scores on Cityscapes **validation dataset** the SegNet architecture with its original learning parameters was fine-tuned on Cityscapes training dataset. The *webdemo* weights were fine-tuned for approximately 150 epochs and resulted in a mean IoU score of 49.3% on the validation dataset (Table 3).

SegNet-reduced v1: To increase the number of images processed per second, the number of feature maps of convolutional networks were reduced as shown in Table 2. This reduced the inference time from 561 ms down to 395 ms and the inference memory from 1461 MB to 1241 MB. This architecture was then trained on Cityscapes training dataset using the *webdemo* initial weights. Due to the reduced number of feature maps, the IoU score decreased from 49.3% to 40.4%. This drop in quality was also easily noticeable on the generalization set. Further, this architecture was trained on a combined dataset of Cityscapes and GTA part 1 (overall 5460 images) for another 150 epochs. This raised the IoU on Cityscapes validation dataset by almost one percent (Table 3) whereas in the generalization set the training seemed to have greater success, especially in recognizing *road* pixels (Figure 18).

The same *webdemo* weights were then fine-tuned on an even bigger combined dataset; namely Cityscapes, GTA part 1 and part 2 containing 7960 pairs of labeled images. This fine-tuning resulted in a one percent increase on Cityscapes validation dataset compared to the previous fine-tuning. However, in comparison, the difference in accuracy on the generalization set was rather small and in some cases slightly lower, especially in images sourced from the MIG front camera. This could be explained by overfitting due to too many synthetic images.

[9]`http://mi.eng.cam.ac.uk/projects/segnet/tutorial.html`
[10]`http://mi.eng.cam.ac.uk/projects/segnet/`
[11]`https://www.cityscapes-dataset.com/benchmarks/`

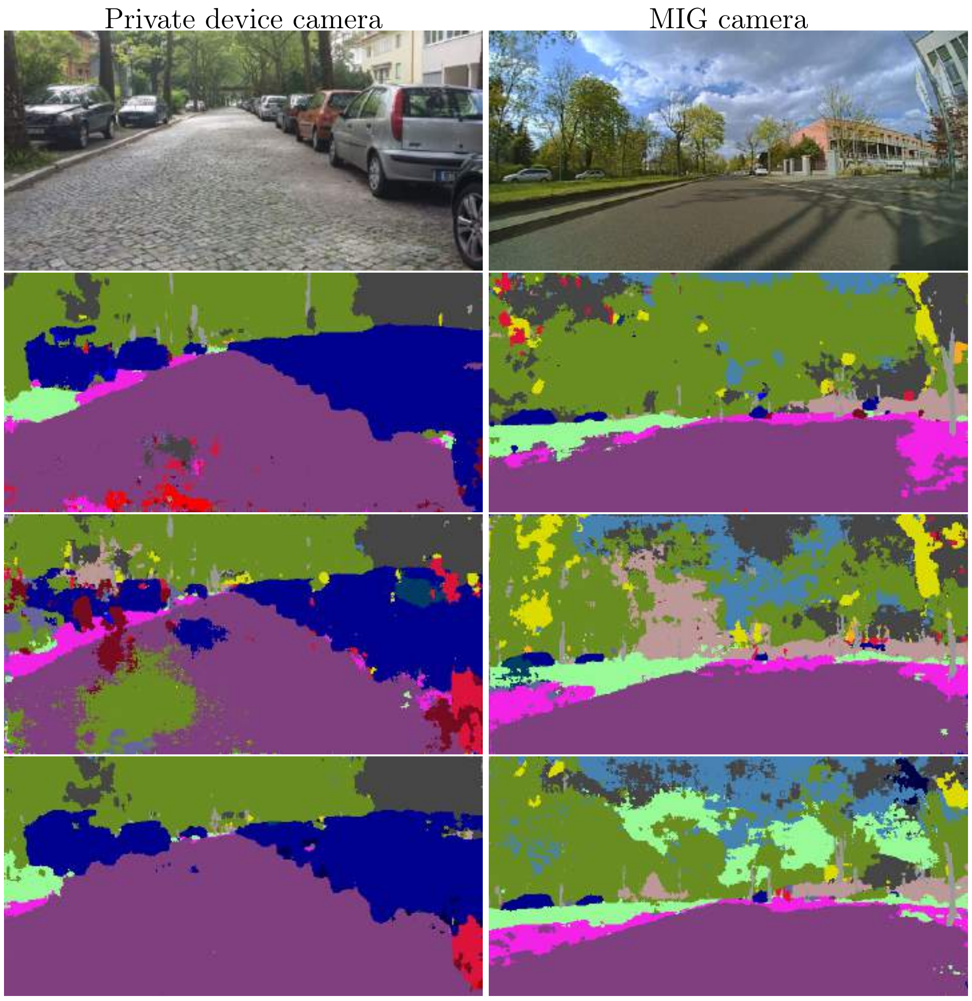


Figure 18: Qualitative results of SegNet and *SegNet-reduced v1*. From top to bottom: input image, original SegNet trained on Cityscapes, *SegNet-reduced v1* trained on Cityscapes, *SegNet-reduced v1* trained on Cityscapes and GTA part 1. For initializing the weights *webdemo* weights were used. Input size of images was 512 × 256 both for training and evaluation.

The dataset was then extended to 25398 images: Cityscapes and the first 9 parts of GTA dataset. This training scored a mean IoU of 47.9%. However, the predictions of the generalization set turned out to be less accurate than the previous training. The main problem was the recognition of the road pixels from the MIG images, where about the half of the road pixels were classified as *sidewalk* (Figure 19). Nevertheless, the accuracy of predicting objects such as *bike*, *rider*, *vehicle* and *sky* were higher in most cases.

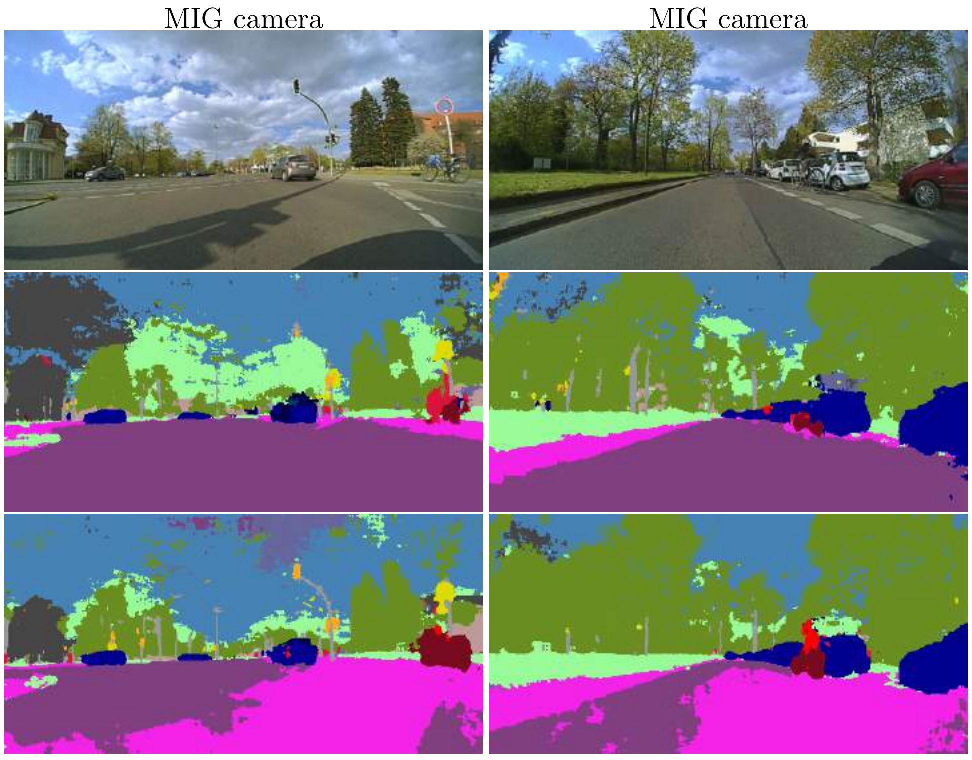


Figure 19: Qualitative results of *SegNet-reduced v1*. From top to bottom: input image, *SegNet-reduced v1* trained on Cityscapes plus GTA part 1 and part 2, *SegNet-reduced v1* trained on Cityscapes and GTA parts from 1 to 9. Both were trained and evaluated using an image size of 512 × 256 and used *webdemo* model as initial weights.

	SegNet	SegNet-reduced-v1	SegNet-reduced-v2	SegNet-wide-dropout	SegNet-basic
conv1_1	64	64	64	32	64
conv1_2	64	64	64	32	-
pool					
conv2_1	128	128	128	128	64
conv2_2	128	128	128	128	-
pool2					
conv3_1	256	256	128	128	64
conv3_2	256	256	128	128	-
conv3_3	256	256	128	128	-
pool3					
conv4_1	512	128	128	128	64
conv4_2	512	128	128	128	-
conv4_3	512	128	128	512	-
pool4					
conv5_1	512	128	128	128	-
conv5_2	512	128	128	128	-
conv5_3	512	128	128	3072	-
pool5					-
upsample5					-
conv5_3_D	512	128	128	-	-
conv5_2_D	512	128	128	-	-
conv5_1_D	512	128	128	512	-
upsample4					
conv4_3_D	512	256	128	-	64
conv4_2_D	512	256	128	-	-
conv4_1_D	256	256	128	128	-
upsample3					
conv3_3_D	256	256	128	-	64
conv3_2_D	256	256	128	-	-
conv3_1_D	128	128	128	128	-
upsample2					
conv2_2_D	128	128	128	-	64
conv2_1_D	64	64	64	32	-
upsample1					
conv1_2_D	64	64	64	32	64
conv1_1_D	19	19	19	19	19
Kernel size of convs	3	3	3	3	7
Infer memory (MB)	1461	1241	1077	869	563
Infer time (ms)	561	395	284	214	300

Table 2: The architecture of SegNet, SegNet-basic and modifications of SegNet. Values of convolutional layers correspond to the number of feature maps produced by that layer, which are followed by a batch normalization and a ReLU activation layer. All pooling layers are max-pooling operations with a stride and kernel size of 2×2. Upsampling layers *scale* parameter also has value of 2. The inference time refers to the processing time of one 512×256 image on Drive PX.

	SegNet-Original	SegNet-reduced v1			
Dataset	C	C	C+G1	C+G2	C+G9
Infer memory (MB)	1461	1241			
Infer time (ms)	561	395			
IoU on class-level					
Mean IoU	49.3	40.4	41.3	42.3	47.9
Road	94.6	92.1	93.2	92.9	93.6
Sidewalk	65	57.5	61.2	60.4	63.9
Building	81.2	76.8	78	76.3	77.7
Wall	28	19.1	20.2	23.2	32.5
Fence	28.8	19.0	20.4	21.2	27.5
Pole	31.4	26.1	29.1	28.1	31.2
Trafficlight	26.5	17.7	20.3	18	21.7
Trafficsign	36.8	27.6	26.6	26.8	28.3
Vegetation	82.3	79.9	81	80.6	83
Terrain	46	37.5	37.5	38.5	43.4
Sky	86.5	85.0	86.5	86	86.7
Person	51.6	40.5	43.6	41.8	47.6
Rider	30.9	17.3	17.6	20.4	28.6
Car	82.9	77.4	78	78.7	82
Truck	32.8	14.8	16	18.7	36.5
Bus	38.2	17.9	13.3	23.3	41
Train	27.5	8.2	7.5	12	16.3
Motorcycle	15.3	11.4	11.7	13.2	19.1
Bicycle	49.6	42.7	43.1	43.8	48.6
IoU on category-level					
Mean IoU	74.1	69.6	71	70.2	72.5
Flat	96.5	95.9	95.9	95.8	95.9
Nature	82.4	80	81.2	80.7	83.1
Object	36.2	29.5	31.5	30.6	33.2
Sky	86.5	85	86.5	86	86.7
Construction	81.2	76.5	77.9	76.1	77.9
Human	54.6	44.5	47.5	45.7	50.7
Vehicle	81.3	75.9	76.4	76.4	80

Table 3: IoU scores of *SegNet-reduced v1* trained on different datasets compared to IoU scores of SegNet original architecture fine-tuned on Cityscapes. Both networks used *webdemo* weights to initialize the weights. The letter C stands for Cityscapes and G1 for GTA part 1. G2 includes GTA part 1 and part 2. G9 includes all parts from part 1 to part 9. The inference time refers to the processing time of one 512×256 image on Drive PX.

SegNet-reduced v2: This network had only the first 4(conv1_1 to conv2_2) and last 5(conv3_1_D to conv1_1_D) layers of the original SegNet network and in the rest of the convolutional layers the number of feature maps was reduced to 128 (Table 2). Compared to *SegNet-reduced v1* this network had a shorter inference time of 284 ms per image and required 1077 MB of memory. Training on mixed dataset of Cityscapes and GTA part 1 resulted in a mean IoU score of 35.2% and quite low accuracy in the generalization set.

SegNet-wide-dropout: This architecture compared to *SegNet-reduced v2* was significantly wider in the middle (3072 feature maps at the end of the encoder and 512 in the beginning of decoder network) and narrow elsewhere (first and last 2 layers had 32 and the rest of the layers had 128 feature maps). It also had a much shorter decoder as shown in Table 2. Because the number of feature maps of convolutional layers was smaller, the overfitting risk was higher. To reduce the risk, dropout layers with probability $p = 0.5$ were added after pool3, pool4, pool5, conv5_1_D, conv4_1_D and conv3_1_D layers.

This modification led to a decrease of the inference time down to 214 ms and inference memory to 869 MB. Training this network on a combined dataset of Cityscapes and GTA part 1 resulted in a mean IoU score of 18%. The probable reason for this drop is the low number of feature maps used in the first two layers. Changing the number of feature maps meant that the pre-trained *webdemo* weights could not be transferred to these layers, thus the filter bank had to be trained from "scratch", which is a challenge taking into account the low number of training samples.

Segnet-dilated: This network is also a clone of *Segnet-reduced v1*, but with dilated convolutional layers at the beginning of the network. As shown in Figure 20, 4 dilated layers, along with an identity layer (convolution with kernel size of 1×1), were added parallel to the original conv1_1 layer and then concatenated at the output and followed by ReLU activation function. The concatenation layer (concat1_1) concatenates its input feature maps, so that the next layer can be applied to all features maps of previous layers. Each dilated layer outputs 8 feature maps and has following parameters:

	dil_1	dil_2	dil_3	dil_4
Kernel size	3	3	5	7
Dilation	2	4	2	3
Pad	2	4	4	9

This addition increased the inference time to 451 ms and the mean IoU to 43%. The quality of predictions in the generalization set was only slightly better compared to *SegNet-reduced v1* trained on the same mixed dataset.

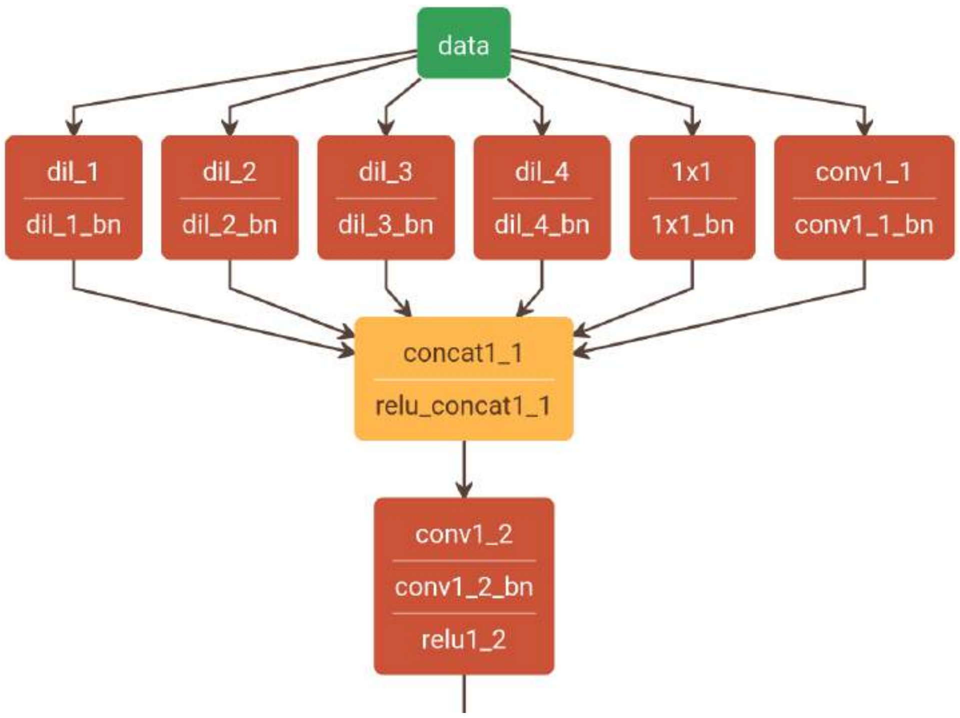


Figure 20: Dilated convolutional layers of *Segnet-dilated* followed by batch normalization layers, which are then concatenated at output and followed by a ReLU activation function. Network visualized using Netscope (`http://ethereon.github.io/netscope/`)

Segnet-SpatialDropout: In this network dropout layers were added to *Segnet-reduced v1*. In contrast to Bayesian SegNet [22], where Kendall et al. used the "standard" dropout in Bayesian version of SegNet-basic, this architecture used only *SpatialDropout* layers [38]. Overall 5 *SpatialDropout* layers with probability $p = 0.3$ were placed after pool4, pool5, conv5_1_D, conv4_1_D and conv3_1_D layers. This increased the mean IoU to 45.2% (Table 4), but the inference time also increased from 395 ms to 461 ms. Although this training resulted in higher IoU scores than *Segnet-dilated* in almost all classes, the accuracy was lower in the generalization set. As illustrated in Figure 21 the network did not recognize a big portion of the road pixels on images from the Galaxy S4 and and MIG cameras.

The mean IoU scores of all trained models and their efficiency are summarized in Table 5. The inference time was calculated on a computer designed for autonomous cars called Drive PX.

	SegNet-reduced-v2	SegNet-wide-dropout	SegNet-dilated	SegNet-SpatialDropout
Dataset	C+G1			
Infer memory (MB)	1077	869	1543	1023
Infer time (ms)	284	214	451	461
IoU on class-level				
Mean IoU	35.2	18	43	45.2
Road	90.8	81.5	92.8	92.9
Sidewalk	54.3	26.8	59.5	62.5
Building	71.6	26.6	76.7	75.7
Wall	14.5	1.7	19.9	31.1
Fence	13.4	2.7	16.8	24.4
Pole	22	3.5	26.1	29
Trafficlight	10.9	2.1	19.8	18.1
Trafficsign	15.7	3.5	24.2	27.7
Vegetation	78.1	63.9	80.1	81
Terrain	35.5	10.6	41.3	42
Sky	83.6	58.1	86.6	84.9
Person	33.5	13.9	41.4	44.9
Rider	9	0	19.4	28.3
Car	72.6	44.3	79.3	80.4
Truck	10.8	1.8	33.4	25.3
Bus	12.3	0	35.1	35.8
Train	0.8	0	11.8	12.4
Motorcycle	4.6	0	8.9	14.4
Bicycle	35.1	0	43.5	47.4
IoU on category-level				
Mean IoU	65.7	44.6	70.1	70.8
Flat	95	91.3	95.7	95.8
Nature	78	63.6	80.1	81.5
Object	22.9	8.6	29.2	31.2
Sky	83.6	58.1	86.6	84.9
Construction	71.6	28.2	76.5	76
Human	37.4	15.7	44.8	47.8
Vehicle	71.1	46.5	77.5	78.1

Table 4: IoU scores of SegNet-reduced v2, SegNet-wide-dropout, SegNet-dilated, SegNet-SpatialDropout all fine-tuned (using *webdemo* weights) on combined dataset of Cityscapes and GTA part 1. C stands for Cityscapes and G1 for GTA part 1. The inference time refers to the processing time of one 512 × 256 image on Drive PX.

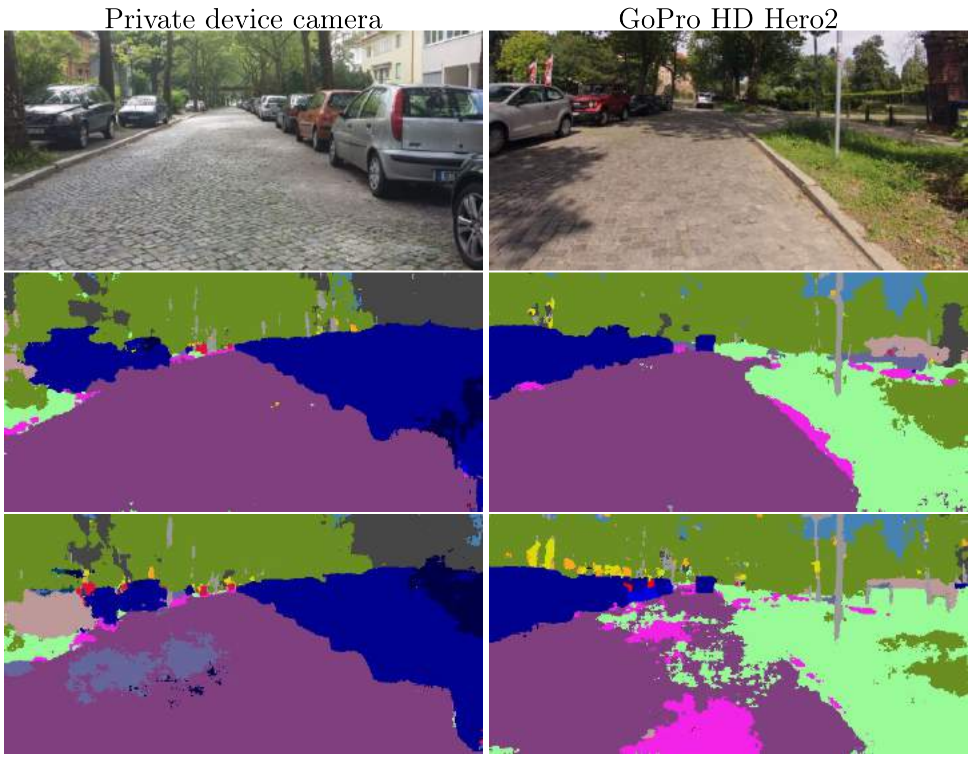


Figure 21: Qualitative results of *Segnet-dilated*(top) and *SpatialDropout*. Both were trained and evaluated on the same combined dataset (Cityscapes plus GTA part1) with an input image size of 512 × 256 and used *webdemo* model as initial weights.

	Original	v1	v1	v1	v1	v2	wide-dropout	dilated	SpatialDropout
Dataset	C	C	C+G1	C+G2	C+G9	C+G1	C+G1		
Input size	512x256								
Inf. time(ms)	561	395				284	214	451	461
Mean IoU class-level	49.3	40.4	41.3	42.3	47.9	35.2	18	43	45.2
Mean IoU cat.-level	74.1	69.6	71	70.2	72.5	65.7	44.6	70.1	70.8

Table 5: Comparison of mean IoU scores of all trained SegNet models and their inference time on Drive PX. The models were evaluated on Cityscapes validation dataset.

4.1.2 SegNet-basic

Cordts et al. [6] presented 3rd party baselines, that claim a mean IoU score of 57.0% for SegNet-basic evaluated Cityscapes **test dataset**. As the training details were not discussed in that paper, the training parameters of this network have been taken from the [2] and the pre-trained weights (uploaded by SegNet[12]) were fine-tuned on the Cityscapes training dataset. After training for about 50 epochs a mean IoU score of only 34.2% was achieved. Further, the initial learning rate of 0.1 was reduced by a factor of 10 twice every 50 epochs. This resulted in 36.6% on Cityscapes **validation dataset** (Table 6). As expected, the SegNet-basic, which has 17 convolutional layers less and produces far less feature maps than SegNet, did not score close to the SegNet accuracy. The quality in the generalization set was also low (See appendix A).

SegNet-basic was then trained on the combined dataset of Cityscapes and GTA part 1. With an initial learning rate of 0.1 the network was fine-tuned for about 150 epochs and achieved a mean IoU score of 35.6% on the validation dataset. This training, compared to the previous training on only Cityscapes dataset, recognized objects in the generalization set better in some parts of some images while worse in other parts, keeping quality overall the same.

SegNet-basic-SpatialDropout: In this experiment, 3 *SpatialDropout* layers were employed. Two, with dropping probability of $p = 0.2$, were placed after the first and second convolutional layers in the decoder network and the other one, with $p = 0.05$, after the third convolutional layer also in decoder. Training this network on the mixed dataset of Cityscapes and GTA part 1 neither generated better results in validation dataset (35.2%) nor in the generalization set. This meant dropping out channels did not create better results and the reason for this can be the high number of classes compared to small number of convolutional layers and feature maps.

[12]`https://github.com/alexgkendall/SegNet-Tutorial/blob/master/Example_Models/segnet_model_zoo.md`

	Basic-original (test-set)	Basic-original	Basic-original	Basic-SpatialDropout
Dataset	C		C+G1	
Infer memory (MB)	-	563	563	563
Infer time (ms)	-	300	300	377
IoU on class-level				
Mean iIoU	57	36.6	35.6	35.2
Road	96.4	91.4	91.8	90.7
Sidewalk	73.2	53.7	55.3	52.2
Building	84	73.3	73.1	70.5
Wall	28.5	11.9	12.5	12.4
Fence	29	10.4	12.3	12.5
Pole	35.7	23.1	23.5	21.7
Trafficlight	39.8	15.4	13.5	12.9
Trafficsign	45.2	24.6	22	21
Vegetation	87	79.1	79.1	77.2
Terrain	63.8	38	36.1	35.5
Sky	91.8	78.6	81.3	78.9
Person	62.8	37.9	35.4	35.7
Rider	42.8	13.2	12.7	16.4
Car	89.3	74.3	72.3	71.2
Truck	38.1	6.2	5.2	6.3
Bus	43.1	18.9	9.4	9.9
Train	44.2	4.5	3.4	2.3
Motorcycle	35.8	4	3.7	7.2
Bicycle	51.9	36.1	34.3	35
IoU on category-level				
Mean IoU	79.1	66.4	66	64.5
Flat	97.4	94.9	94.9	94.3
Nature	86.7	79.4	79	77.3
Object	42.5	26.4	25.8	24.3
Sky	91.8	78.6	81.3	78.9
Construction	83.8	73.2	72.8	70.1
Human	64.7	40.5	38.1	38
Vehicle	87.2	71.9	70.3	68.6

Table 6: IoU scores of SegNet-basic and SegNet-basic-SpatialDropout fine-tuned on Cityscapes and on mixed datasets of Cityscapes and GTA part 1. C stands for Cityscapes and G1 for GTA part 1. The inference time refers to the processing time of one 512 × 256 image on Drive PX.

4.1.3 ENet

Originally ENet was implemented in a machine learning library known as Torch [29]. However, it was also implemented in other frameworks, namely in Keras and Caffe. The Caffe implementation called caffe-enet[13] was used for this work for all of the ENet trainings. One disadvantage of this implementation was a lack of the multiple GPU support when training the ENet. This was due to the constraints of the *SpatialDropout* layer implementation in Caffe framework. Further, this implementation included scripts for training and testing the trained networks: A script for computing the batch normalization statistics, a script for merging batch normalization and *SpatialDropout* layers with preceding convolutional layers and a script to visualize the network output compatible to class values of Cityscapes dataset. Moreover, it also provided model weights trained on Cityscapes dataset and the training parameters (solver.prototxt) such as a base learning rate of $5e-4$, a weight decay of $2e-4$ and a momentum of 0.9. To train the ENet from scratch, without using pre-trained weights, the authors of the network, Paszke et al. recommended to train the encoder first and then, using the weights of trained encoder, train the decoder to perform upsampling. For fine-tuning, however, the encoder-decoder network was taken as a whole. In all of the trained models the provided weights were fine-tuned for 200 epochs using a single GPU. The batch size was set to 4 due to GPU memory constraints and the base learning rate to $5e-5$, which was reduced by a factor of 10 about every 50 epochs of training. As in the original paper [29], the Adam optimization algorithm [23] was used.

Paszke et al. trained ENet with an input and output size of 1024×512, which resulted in a mean IoU score of 58.3% on Cityscapes **test dataset** [14]. Because the ENet implementation in Caffe did not support multiple GPUs, the training with input size of 1024×512 could be done with a maximum batch size of 2 due to GPU memory limits. This batch size however was not enough for proper training, as the model did not converge. Instead the input size was reduced and the weights of the Caffe implementation were used as initial weights for later training[15]. Initially, these weights were evaluated on Cityscapes *validation dataset* which showed a mean IoU score of 53.3%. The IoU values of classes *Bus* and *Train* were, however, either 0 or near 0 (Table 7). This problem was later gradually solved as the training dataset became larger and larger.

To reduce the inference time and assure a reasonable mini-batch size of 4 for training, the input size of the images was downsampled to 800×400. This

[13]`https://github.com/TimoSaemann/ENet`

[14]`https://www.cityscapes-dataset.com/detailed-results/`

[15]`https://github.com/TimoSaemann/ENet/tree/master/enet_weights_zoo`

setting was then fine-tuned on Cityscapes training dataset, which resulted in a mean IoU score of 48%, however the problems with recognizing *buses* and *trains* persisted. As this and the provided weights were only trained on the Cityscapes dataset, they had similar results on the generalization dataset.

Next, GTA part 1 was added to the training dataset. Training on these 5460 images resulted a slightly smaller mean IoU score of 47.4%, however on the generalization dataset, the quality of this training was much higher (Figure 22). Nevertheless, the clouds in the images obtained from the MIG front camera were mostly recognized as *terrain* and the rest of the sky mostly as *vegetation*.

Afterwards, the training dataset was complemented with the GTA part 2 bringing the total to 7960 annotated images. This setting achieved a mean IoU of 51.3%, but most importantly the *bus* class was recognized with 46.6% accuracy (Table 7). In the generalization set the difference, compared to the previous training, was not significant except that this time the sky in MIG images was classified slightly better.

Then, the GTA parts from 3 to 9 were added to the training dataset. This amounted to 25398 images and after 200 epochs of training on this dataset a mean IoU score of 52% was achieved. This model, unlike previous models, predicted the *train* class. It resulted a score IoU of 31% and 56.6% on *train* and *bus* classes, respectively. So far this model produced the highest mean IoU score on the validation dataset, however in the generalization set, presumably because of overfitting to the GTA dataset, the prediction of road pixels demonstrated poor accuracy. However, predictions of images of other sources, which were usually of lower quality in previous trained models, turned out to be easier for this model (Figure 23).

To address the previous model's overfitting (fine-tuned on the combined dataset of Cityscapes and GTA part 1 to part 9), it was fine-tuned on the combined dataset of Cityscapes and GTA part 1. The idea behind this was that the model trained on the bigger dataset has *learned* both low and high level features, however as the dataset consisted mostly of synthetic data, the model's highest level features needed to be fine-tuned on a more diverse dataset.

Input size	1024 x512 (test set)	1024 x512	800x400			
Dataset	C		C	C+G1	C+G2	C+G9
Infer memory (MB)	-	2255	1733			
Infer time (ms)	-	260	173			
IoU on class-level						
Mean IoU	58.3	53.3	48.0	47.4	51.3	52
Road	96.3	95.4	94.1	93.7	93.5	93.8
Sidewalk	74.2	71.7	67.5	66	65.4	64
Building	85	85.5	78.7	78.7	79	77.2
Wall	32.2	37.3	45.2	43.5	42.6	33.8
Fence	33.2	43.6	39.3	36.8	35.7	34.1
Pole	43.5	43.6	30.1	30.7	31.4	31.2
Trafficlight	34.1	42.9	25.1	21.3	22.6	21.4
Trafficsign	44	52	36	34.9	35	28.7
Vegetation	88.6	86.5	83.4	83.5	83.8	83.4
Terrain	61.4	52.8	46.8	46.1	46.3	42.8
Sky	90.6	88.5	84.9	86.7	87.6	86.5
Person	65.5	65.8	57.1	54.9	54.2	53.5
Rider	38.4	47.6	41.4	42.4	41.7	40
Car	90.6	87.1	84.3	84.1	84.6	83.7
Truck	36.9	25	26.4	26.5	47.2	49.9
Bus	50.5	0.1	0.1	0	46.6	56.5
Train	48.1	0	0	0	0	31
Motorcycle	38.8	28.6	21.2	20.8	23.8	25.8
Bicycle	55.4	58.9	49.5	50.6	53	51.5
IoU on category-level						
Mean IoU	80.4	80.1	74.3	74.2	74.4	73.5
Flat	97.3	97.2	96.4	96.3	96.3	95.9
Nature	88.3	87.1	84.2	84	84.3	84.1
Object	46.8	49.2	34.3	34.1	34.8	32.9
Sky	90.6	88.5	84.9	86.7	87.6	86.5
Construction	85.4	85.7	79.3	79.1	79.3	78.1
Human	65.5	68.1	59.4	57.5	56.7	56.3
Vehicle	88.9	85	81.6	81.6	81.6	81

Table 7: IoU Scores of ENet fine-tuned on different datasets and input sizes. All values refer to the evaluation on the Cityscapes validation dataset, except the first column which shows IoU scores from the original ENet model evaluated on the test dataset. C stands for Cityscapes and G1 for GTA part 1. G2 includes GTA part 1 and part 2. G9 includes all parts from part 1 to part 9. The inference time refers to the processing time of one image on Drive PX.

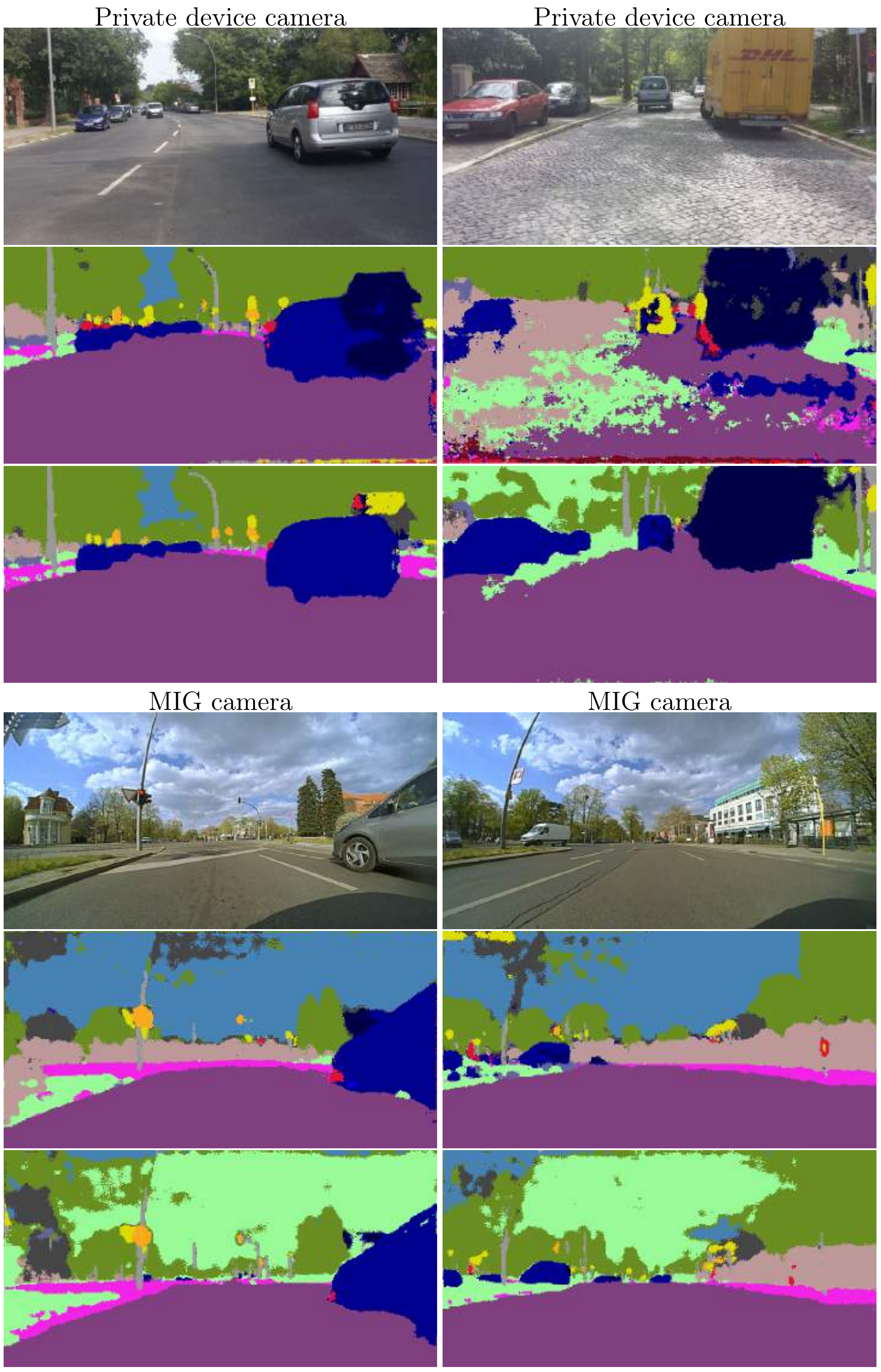


Figure 22: Qualitative results of ENet. From top to bottom: input image, provided weights fine-tuned only on Cityscapes and fine-tuned on Cityscapes and GTA part 1. Both were fine-tuned and tested using an image size of 800 × 400.

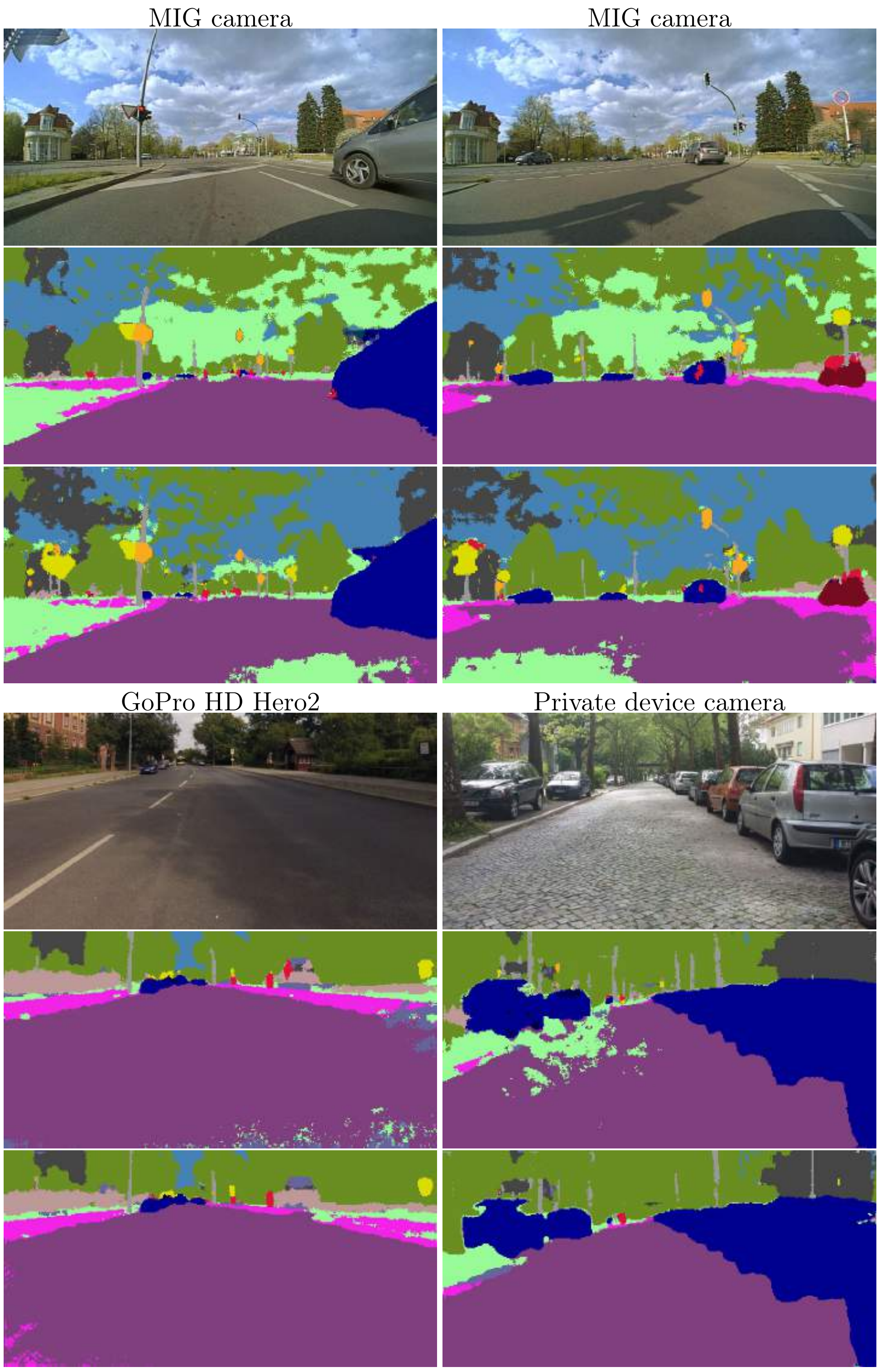


Figure 23: Qualitative results of ENet. From top to bottom: input image, fine-tuning only on Cityscapes plus GTA part 1 and part 2, fine-tuning on Cityscapes plus GTA from part 1 to part 9. Both were fine-tuned and tested with an image size of 800 × 400.

Two attempts, one with a learning rate of $5e-4$ and the other with $5e-5$, were made. Both lasted 200 epochs and the learning rate was reduced by a factor of 10 every 50 epochs. The first one resulted a mean IoU score of 54% and the second one 55% (Table 8). Not only did these two models score the highest IoU scores on the validation dataset, these were also best at generalizing: The first one being more accurate at recognizing images of the MIG camera while the second one at recognizing the rest (Figure 24).

Input size	800x400
Dataset	C+G1
Infer memory (MB)	1733
Infer time (ms)	173

Learning rate	5e-4	5e-5
IoU on class-level		
Mean IoU	54	55
Road	93.5	94
Sidewalk	65.6	66.7
Building	79.8	80.7
Wall	42.3	42
Fence	38.6	39.4
Pole	32.6	33.2
Trafficlight	24.8	26
Trafficsign	35	36.9
Vegetation	84.3	84.9
Terrain	46.1	49.8
Sky	88.1	88.1
Person	55.8	57
Rider	42.3	42.5
Car	85	85.7
Truck	58.2	55.5
Bus	60.5	59.2
Train	13	20.6
Motorcycle	27.7	28.9
Bicycle	53.1	53.7

	5e-4	5e-5
IoU on category-level		
Mean IoU	75.2	75.8
Flat	96.4	96.6
Nature	85	85.4
Object	35.8	36.9
Sky	88.1	88.1
Construction	80.5	81.4
Human	58	59.2
Vehicle	82.5	83

Table 8: IoU scores of two ENet models first fine-tuned on combined dataset of **Cityscapes and GTA part 1 to part 9**, then on **Cityscapes and GTA part 1**. *Learning rate* refers to the learning rate at the beginning of each training. The model was evaluated on Cityscapes evaluation dataset. C stands for Cityscapes and G1 for GTA part 1. The inference time refers to the processing time of one image on Drive PX.

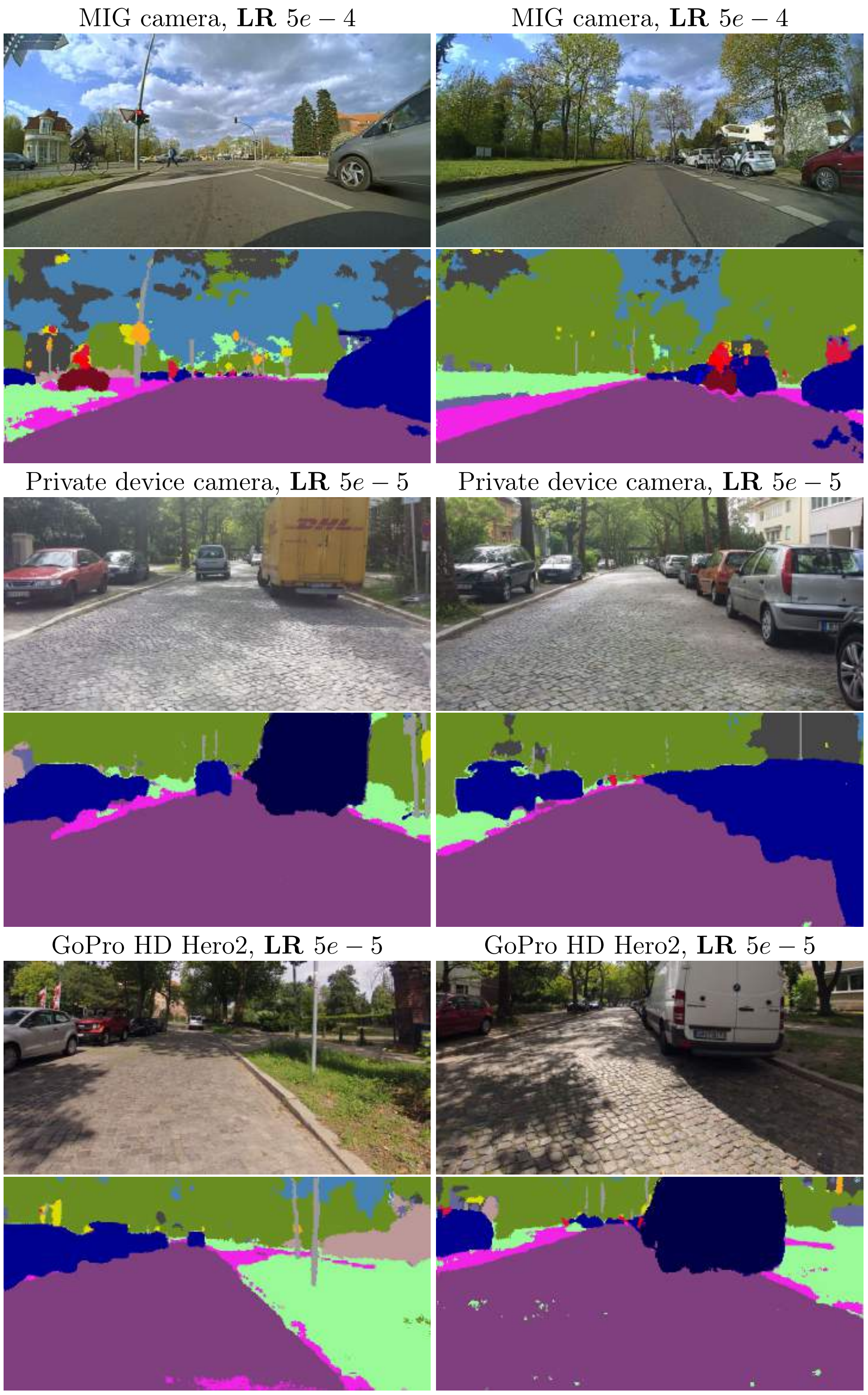

MIG camera, **LR** $5e-4$
MIG camera, **LR** $5e-4$
Private device camera, **LR** $5e-5$
Private device camera, **LR** $5e-5$
GoPro HD Hero2, **LR** $5e-5$
GoPro HD Hero2, **LR** $5e-5$

Figure 24: Qualitative results of two ENet models first fine-tuned on the combined dataset of **Cityscapes and GTA part 1 to part 9**, then on **Cityscapes and GTA part 1**. *LR* refers to the learning rate at the beginning of each training.

4.1.4 Chapter Summary

In an attempt to train an efficient and better generalizing network, 18 experiments were carried out. This chapter presented the information describing the experiment networks, their training and evaluation, both quantitatively and qualitatively. The quantitative results were calculated on the Cityscapes validation dataset of 500 pixel-wise annotated images. The qualitative results were manually assessed using collected images from different cameras. First, a reduced version of SegNet called *SegNet-reduced v1* was trained on 3 different combined datasets: **Cityscapes and GTA part 1**; **Cityscapes and GTA part 1 and part 2**; **Cityscapes and GTA part 1 to part 9**. Then, different versions of SegNet (more reduced versions along with versions complemented with new layers) were trained on combined datasets of **Cityscapes and GTA part 1**. A more simple network called SegNet-basic and a modification of it were also trained on **Cityscapes and GTA part 1**, however the results of this training were not sufficient. At the end, a network named ENet, with a reduced input size of 800×400, was first fine-tuned on combined datasets of **Cityscapes and GTA part 1**, then on **Cityscapes and GTA part 1 and part 2** and finally on **Cityscapes and GTA from part 1 to part 9**.

Both SegNet and Enet trainings yielded an increase of IoU scores on the validation dataset due to which the number of synthetic images increased in the training dataset. Unfortunately, in the generalization set, the trainings with 9/10th synthetic datasets could not properly recognize some very important classes such as the *road* and the *sidewalk* due to overfitting.

The ENet model trained on **Cityscapes and GTA part 1 to part 9** was then fine-tuned again on the combined dataset of **Cityscapes and GTA part 1** to address the overfitting to the synthetic dataset. This resulted in not only better generalization capabilities, but also better quantitative results on the Cityscapes validation dataset.

The iIoU scores of trained models along with more pixel-wise predictions on the generalization set are displayed in the appendix. The trained models, their parameters and architectures were written to a CD and are attached to this work.

5 Conclusion

The primary aim of this study was to modify existing pixel-wise segmentation CNNs and make them more efficient so they could be of practical use on low-powered devices such as Drive PX. Another objective of this work was to make the network generalize well allowing appropriate pixel-wise predictions on images from different cameras, including the fisheye camera attached to the autonomous car (MIG) belonging to the Free University of Berlin. To achieve this goal, modifications of efficient CNNs were designed. Furthermore, two datasets - one containing only real-world images and the other only synthetic ones - were made compatible to make the training dataset bigger and more diverse. During this research, the networks that performed well on a smaller part of the mixed dataset, were chosen to be trained on bigger parts of the training dataset. Two networks, *SegNet-reduced v1* and ENet (input size 800×400), were chosen to be trained further on a bigger dataset, because of their efficiency and accuracy.

The last training of ENet resulted the highest IoU score on the validation dataset but performed poorly in the generalization set in recognizing some vital classes, such as road and sidewalk. To address this overfitting to the synthetic dataset, the model was then trained with two different learning parameters on a dataset with less synthetic images. The model with learning rate of $5e-4$ turned out to have the highest mean IoU score of 55% and also was more accurate at predicting images obtained from the fisheye camera of the MIG autonomous car.

During the research, the accuracy of all trained models was analyzed quantitatively and qualitatively. To quantify the accuracy, the Cityscapes validation dataset of 500 annotated images was used. For qualitative assessment a (generalization) set of road images from different sources was collected. This showed that having thousands of mostly synthetic images in the training dataset can increase prediction accuracy in some datasets of real-world images, but the network generalization might suffer, regardless of the applied techniques against overfitting. The overfitted model trained on this big dataset, however, can be fine-tuned on its smaller parts, in which the real-world and synthetic images are about equal. This underlined the importance of diverse images and image sources in the training dataset - for better generalization capability - as well as in the validation dataset for better measurement of generalization.

5.1 Future Work

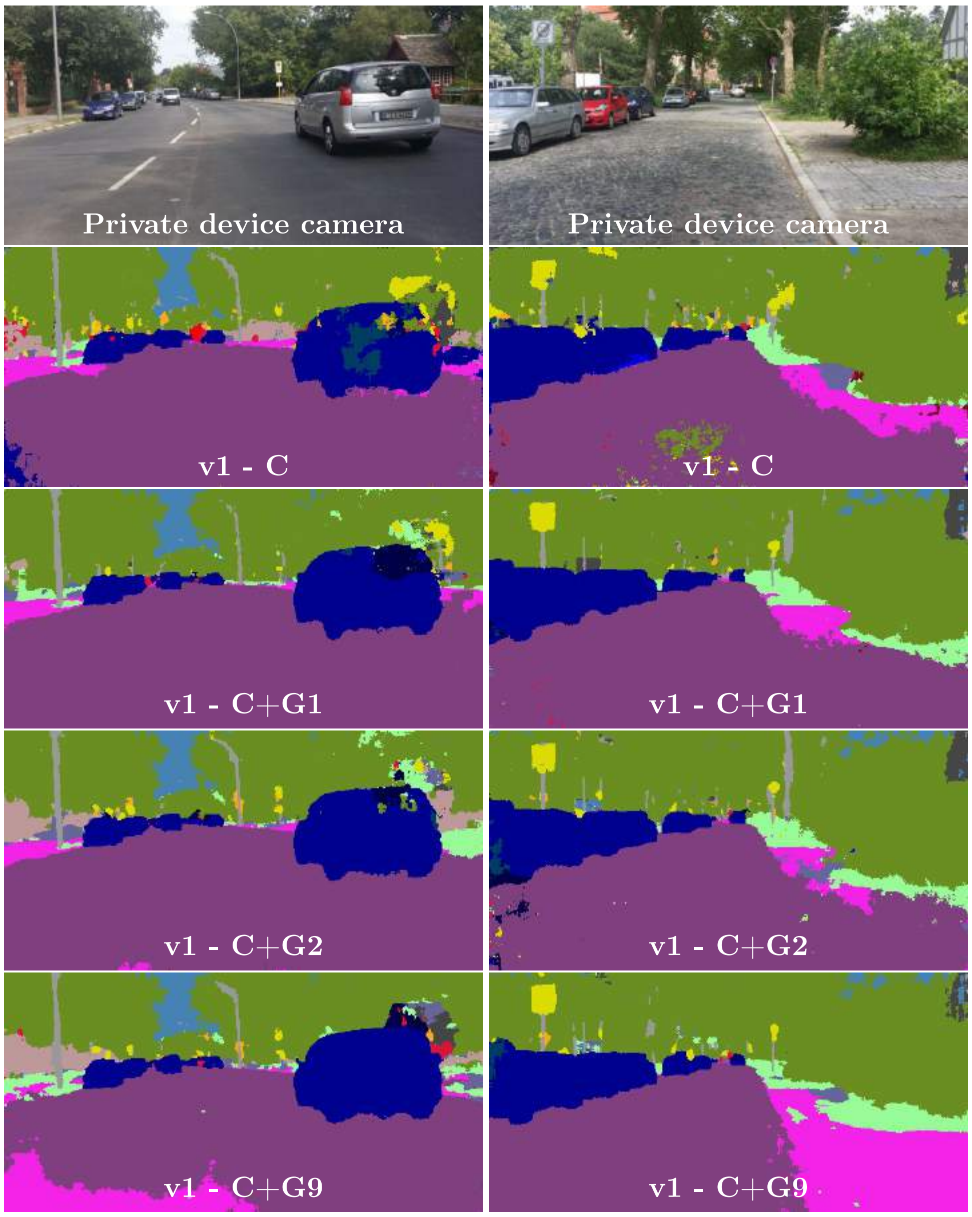
The scope of this research was restricted by the amount of computing power, time and availability of diverse training datasets. Below, several ideas for future work, that could make the network predictions more accurate and better at generalizing, are suggested.

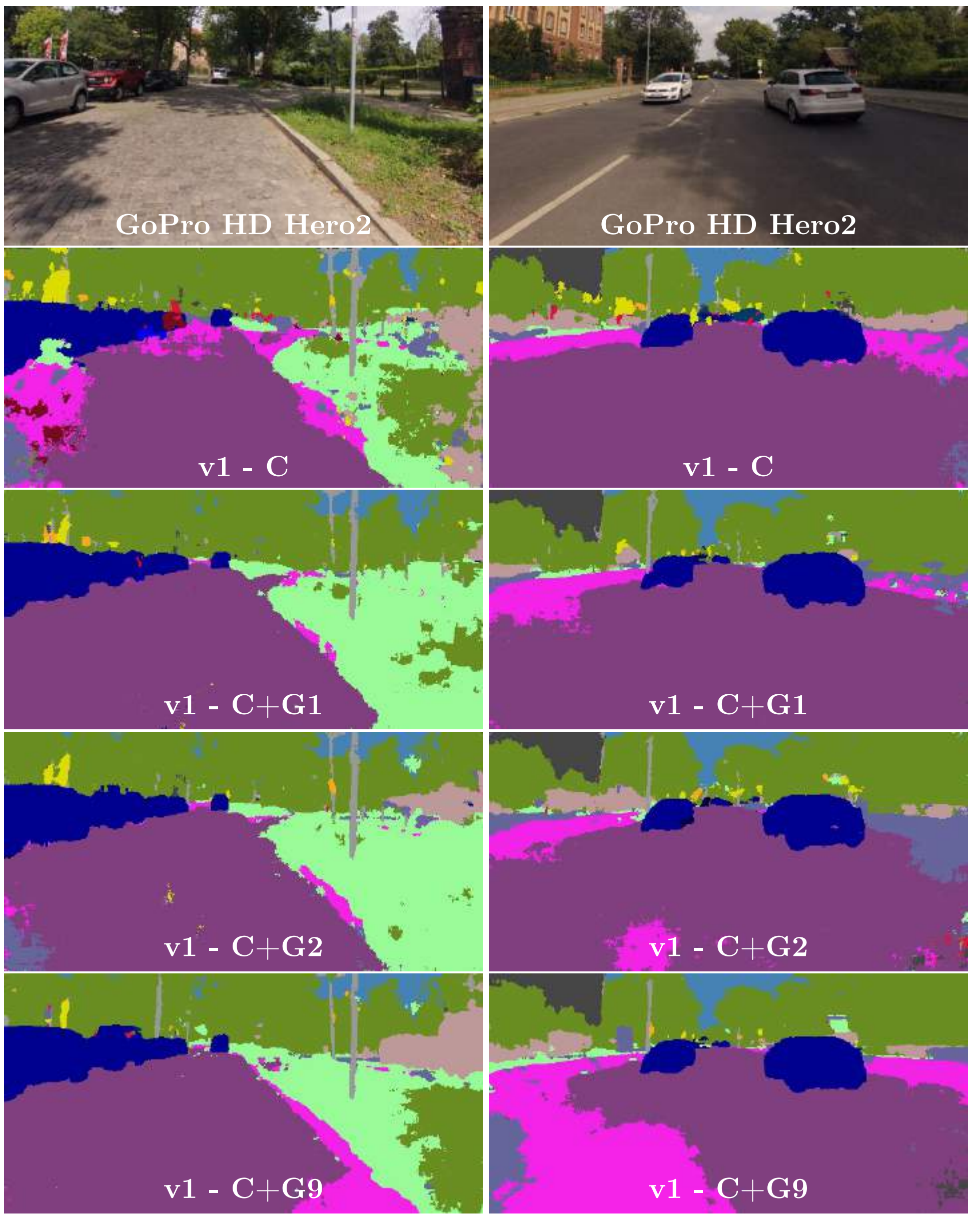
- Adding only one source of synthetic dataset to the real-world training dataset made the models more accurate and better at generalizing. With publishing the *GTA5* dataset, Richter et al.[30] also published their code used for extracting frames from the game, which can also be applicable to other photo-realistic games. This script can be used for adding more sources of synthetic datasets to the training dataset. This has the potential to make models more accurate and especially good at generalizing.
- To train a model capable of accurately labeling images from cameras with different lenses, data augmentation can be applied to parts of existing datasets. Simulating a fisheye (curvilinear) effect on images and on corresponding annotations could possibly make the network immune to lens changes. Adding this kind of augmented dataset to existing ones can potentially make the model more accurate on rectilinear images.
- Adding dilated layers of different sizes in the first encoder of SegNet showed an increase in accuracy and made the network better at generalizing. The same can be said about the SegNet version with *SpatialDropouts* layers. Adding dilated convolutional layers deeper in the encoder and decoder, rather than only in the first encoder, and combining them with *SpatialDropouts* layers can potentially generate better pixel-wise predictions. Having dropout layers should also reduce the risk of over-fitting to big synthetic parts of training dataset.

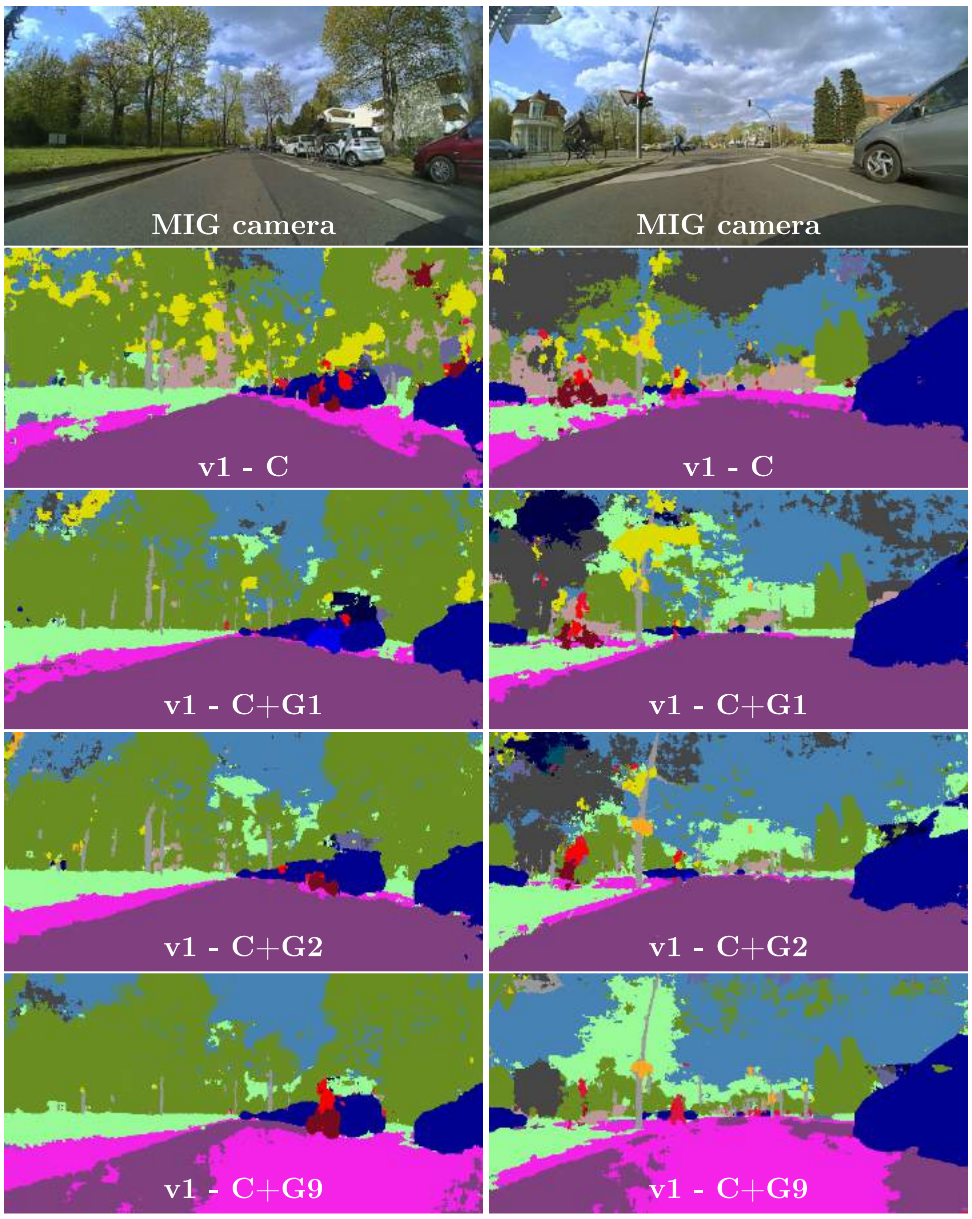
Appendices

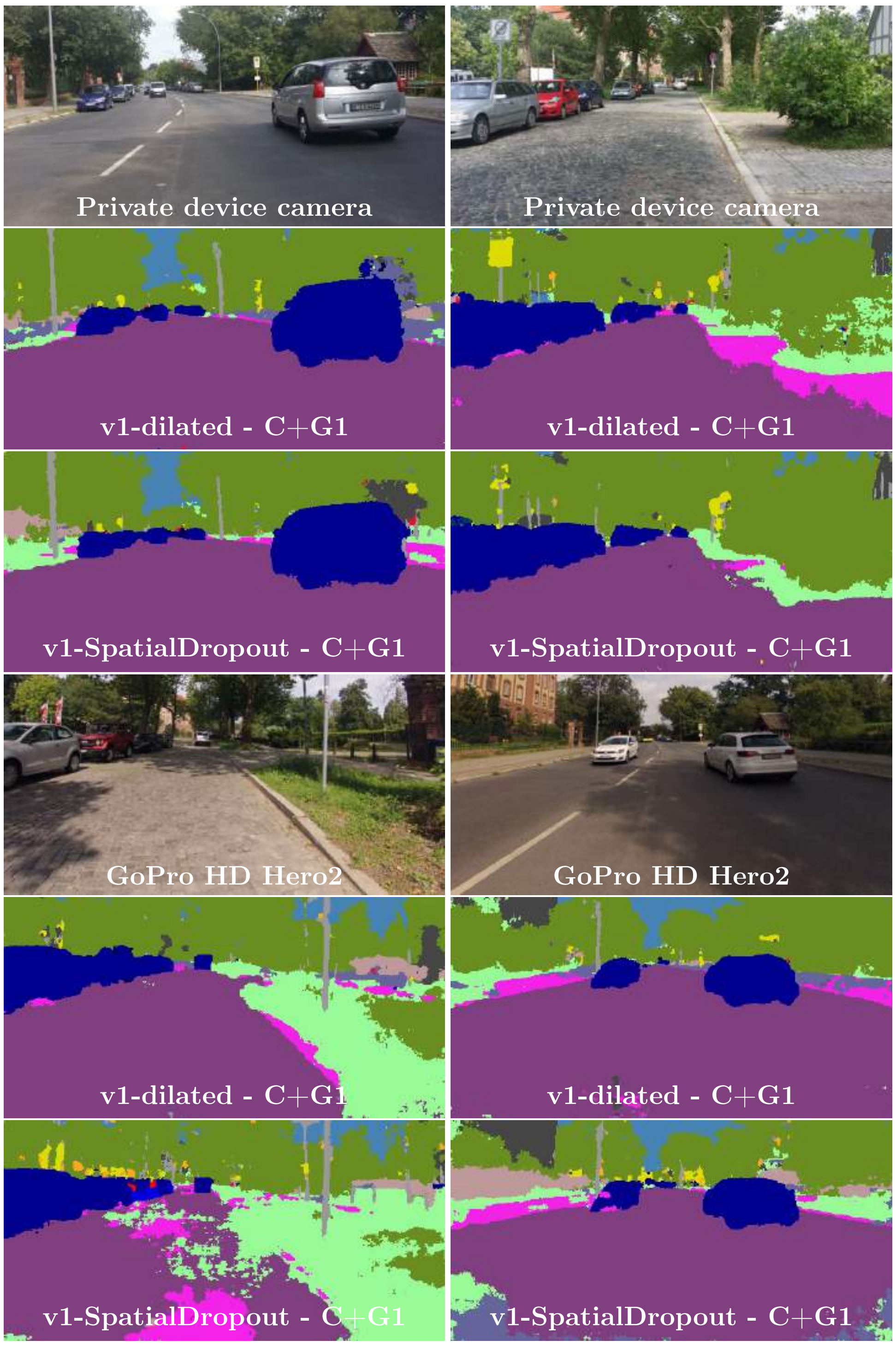
A Qualitative Results

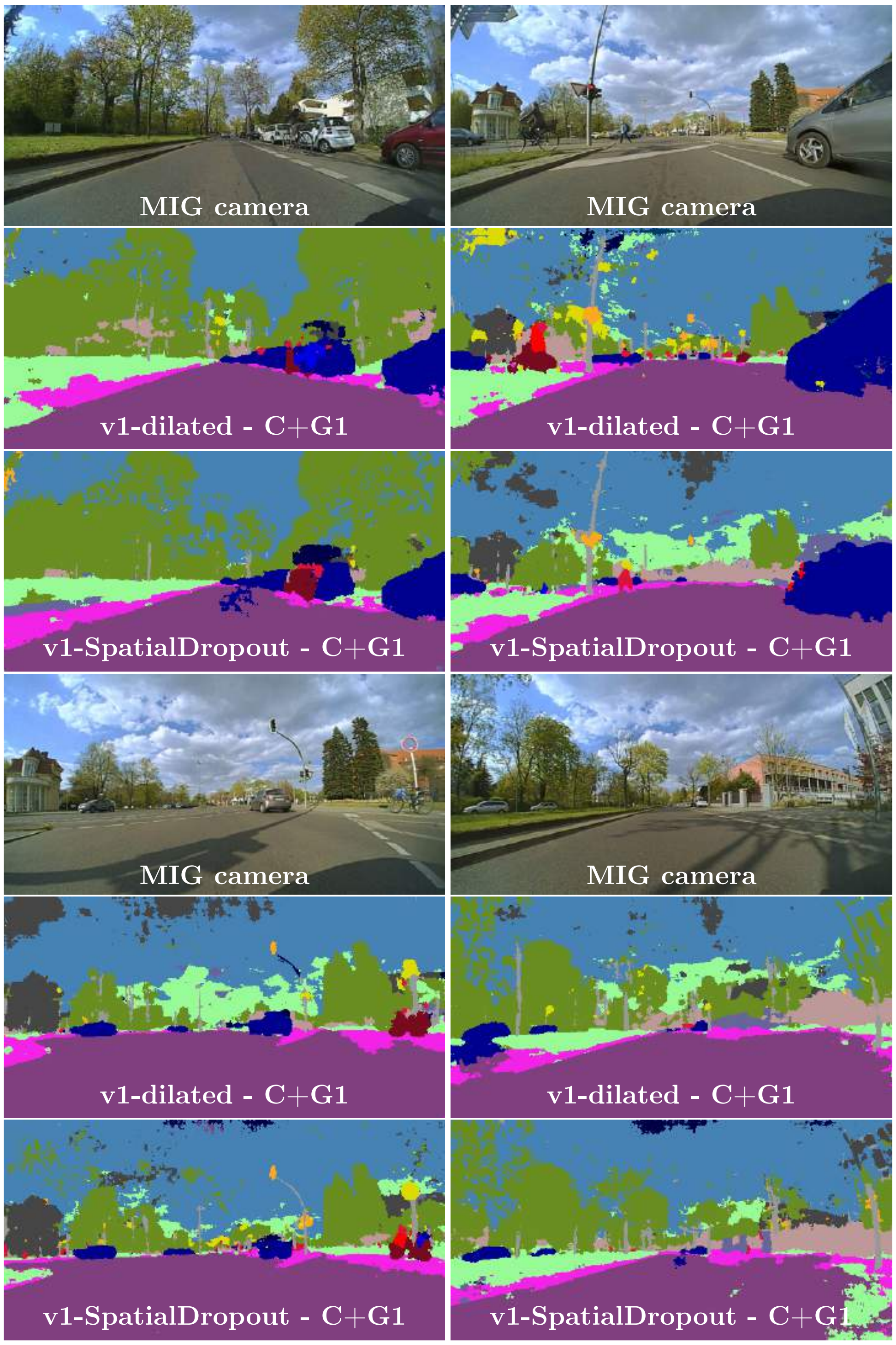
Qualitative results of SegNet-reduced versions trained on different datasets. Input size of images was 512 × 256 both for training and evaluation. The letter C stands for Cityscapes and G1 for GTA part 1. G2 includes GTA part 1 and part 2. G9 includes all parts from part 1 to part 9.



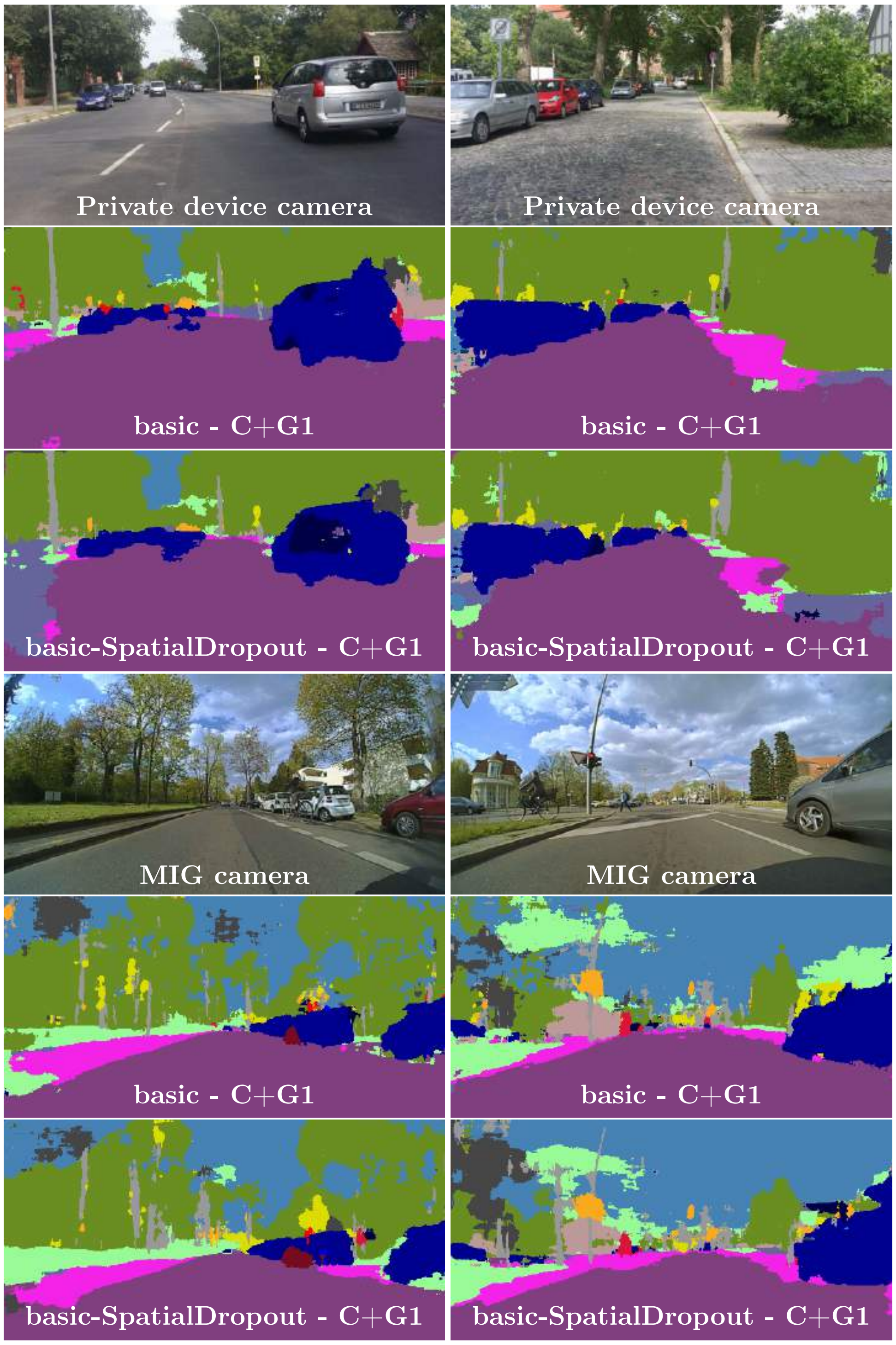

GoPro HD Hero2
GoPro HD Hero2
v1 - C
v1 - C
v1 - C+G1
v1 - C+G1
v1 - C+G2
v1 - C+G2
v1 - C+G9
v1 - C+G9


MIG camera
MIG camera
v1 - C
v1 - C
v1 - C+G1
v1 - C+G1
v1 - C+G2
v1 - C+G2
v1 - C+G9
v1 - C+G9

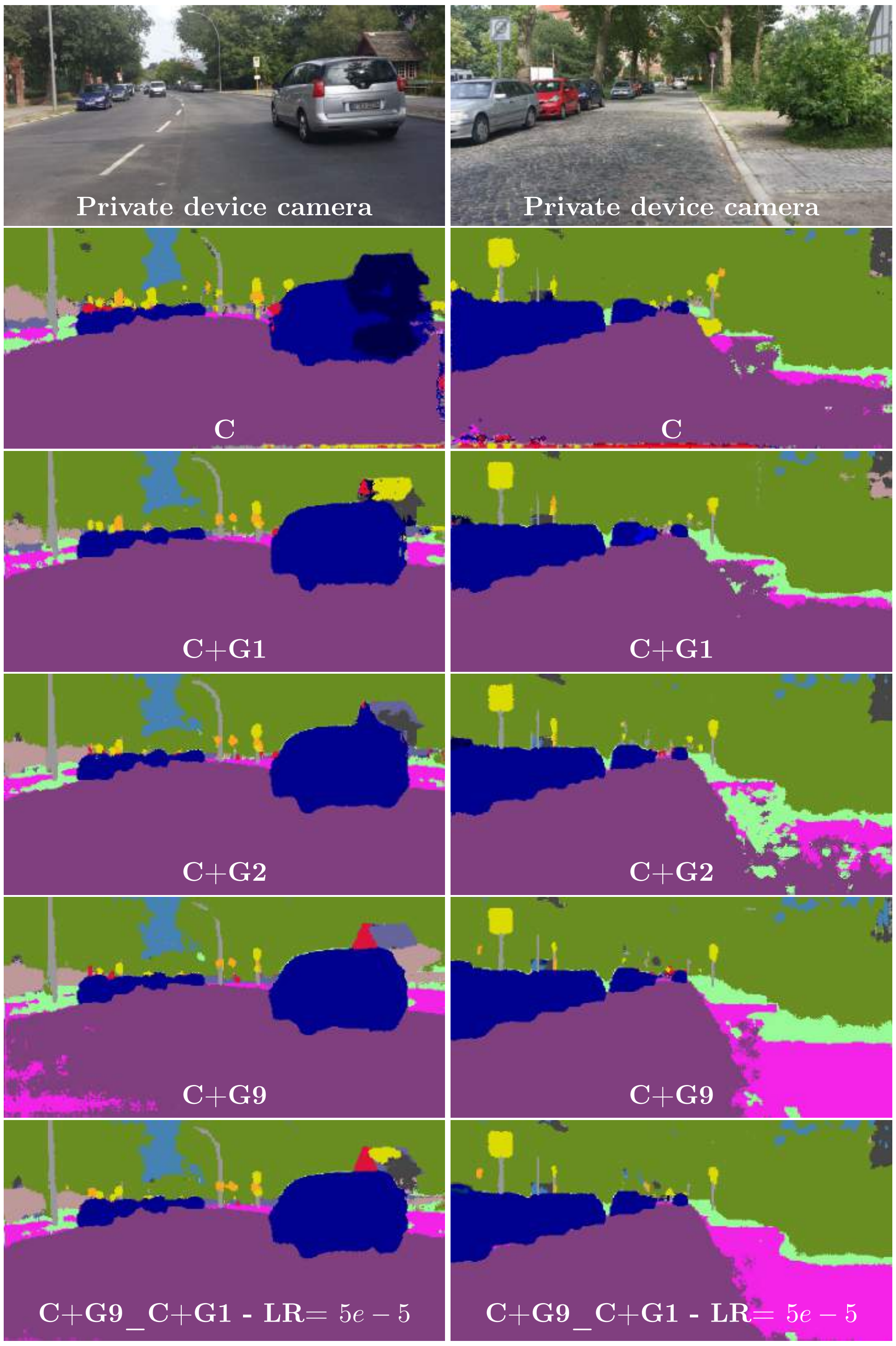

Private device camera
Private device camera
v1-dilated - C+G1
v1-dilated - C+G1
v1-SpatialDropout - C+G1
v1-SpatialDropout - C+G1
GoPro HD Hero2
GoPro HD Hero2
v1-dilated - C+G1
v1-dilated - C+G1
v1-SpatialDropout - C+G1
v1-SpatialDropout - C+G1

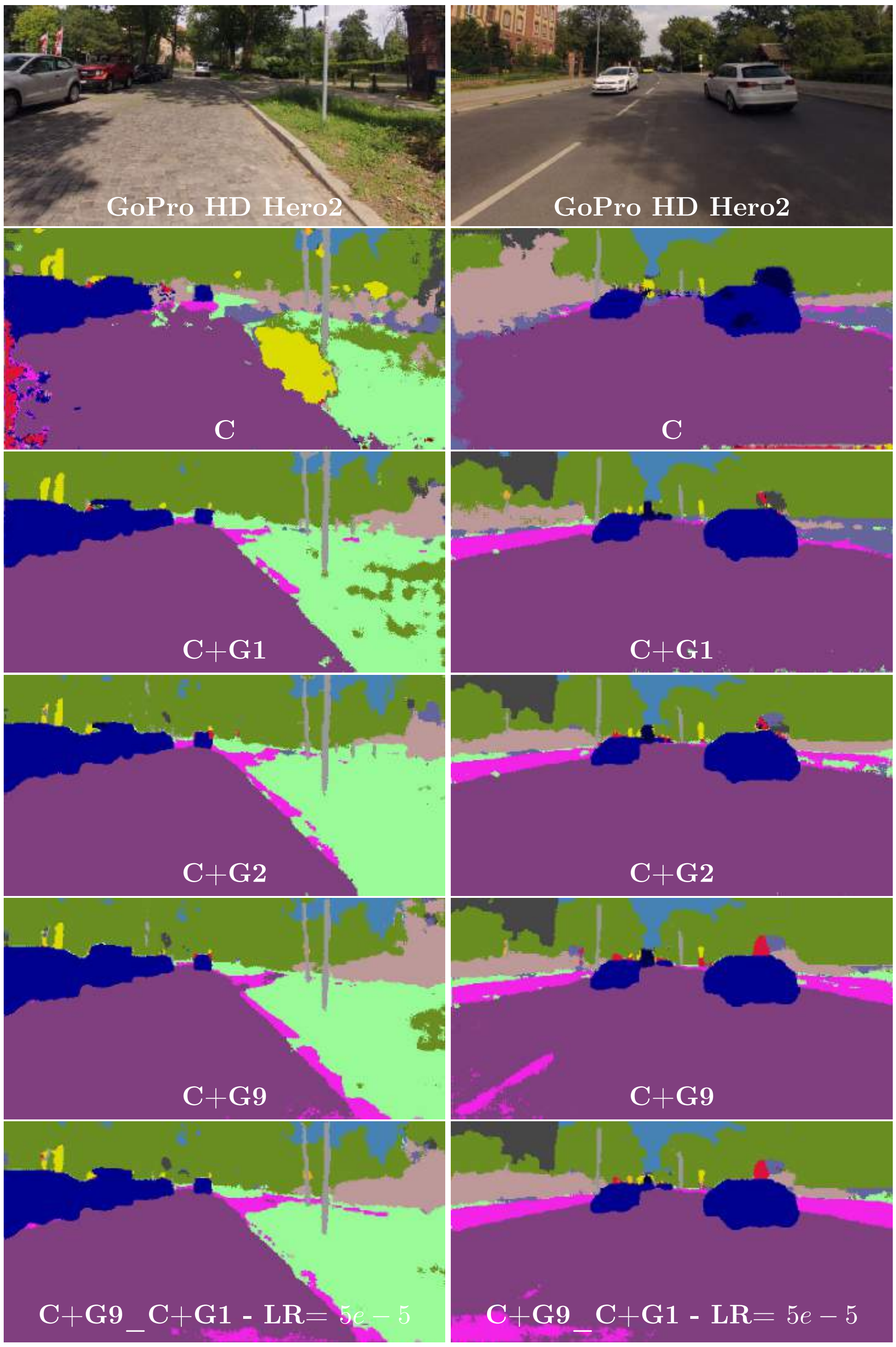

MIG camera
MIG camera
v1-dilated - C+G1
v1-dilated - C+G1
v1-SpatialDropout - C+G1
v1-SpatialDropout - C+G1
MIG camera
MIG camera
v1-dilated - C+G1
v1-dilated - C+G1
v1-SpatialDropout - C+G1
v1-SpatialDropout - C+G1

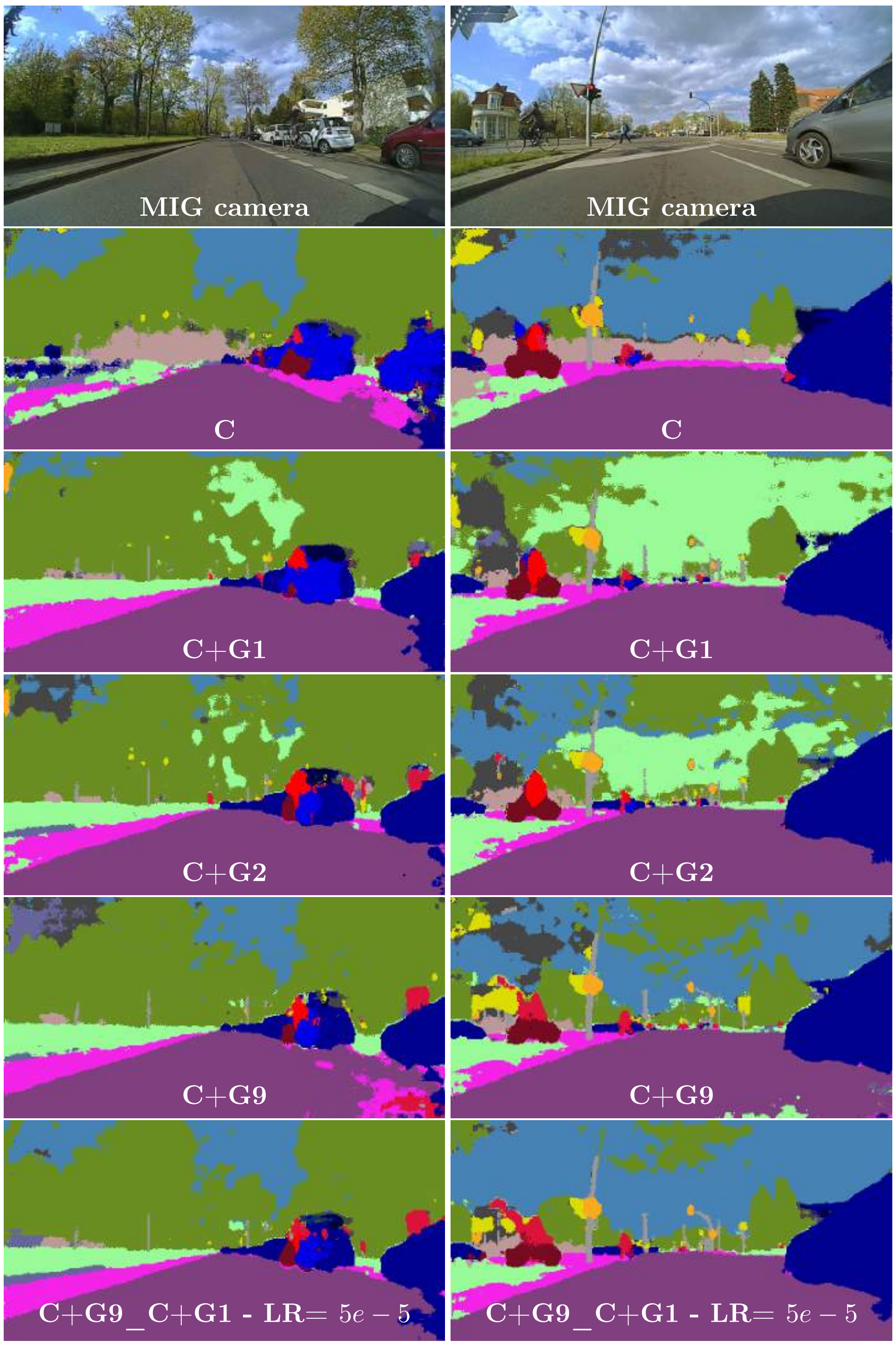
Qualitative results of SegNet-basic and SegNet-basic-SpatialDropout both trained on the combined dataset of Cityscapes and GTA part1. Input size of images was 512×256 both for training and evaluation.



Qualitative results of ENet fine-tuned on different datasets. Input size of images was 800×400 both for training and evaluation. LR stands for learning rate.




GoPro HD Hero2
GoPro HD Hero2
C
C
C+G1
C+G1
C+G2
C+G2
C+G9
C+G9
C+G9_C+G1 - LR= $5e-5$
C+G9_C+G1 - LR= $5e-5$


MIG camera
MIG camera
C
C
C+G1
C+G1
C+G2
C+G2
C+G9
C+G9
C+G9_C+G1 - LR= $5e-5$
C+G9_C+G1 - LR= $5e-5$

B iIoU Scores on Cityscapes Validation Dataset

Table 9: iIoU scores of *SegNet-reduced v1* trained on different datasets compared to iIoU scores of SegNet original architecture fine-tuned on Cityscapes. Both networks used *webdemo* weights for weight initialization. The letter C stands for Cityscapes and G1 for GTA part 1. G2 includes GTA part 1 and part 2. G9 includes all parts from part 1 to part 9.

	SegNet-Original	SegNet-reduced v1			
Dataset	C	C	C+G1	C+G2	C+G9
Infer memory (MB)	1461	1241			
Infer time (ms)	561	395			
Mean iIoU	23.2	19.9	19.6	21	26.4
iIoU on class-level					
Person	33.3	31.2	32.3	31.9	37.5
Rider	15.2	11.6	10.8	14.4	18.9
Car	69.8	64.2	66.5	64.6	68.5
Truck	7.8	6.6	5.7	7.7	14.8
Bus	16.4	10.7	5.8	10.2	23.8
Train	9.9	3.7	3.4	7	7.7
Motorcycle	7	6.2	8.1	6.8	11.8
Bicycle	26.5	24.6	24.6	25.3	28.5
iIoU on category-level					
Mean iIoU	51.8	49.4	50.3	49.6	54.1
Human	36.8	35.6	36.2	36.5	41.7
Vehicle	66.9	63.2	64.5	62.7	66.4

Table 10: iIoU scores of SegNet-reduced v2, SegNet-wide-dropout, SegNet-dilated, SegNet-SpatialDropout all fine-tuned (using *webdemo* weights) on combined dataset of Cityscapes and GTA part 1. C stands for Cityscapes and G1 for GTA part 1.

	SegNet-reduced-v2	SegNet-wide-dropout	SegNet-dilated	SegNet-SpatialDropout
Dataset	C+G1			
Infer memory (MB)	1077	869	1543	1023
Infer time (ms)	284	214	451	461
Mean iIoU	15.6	6.2	21.2	24.5
iIoU on class-level				
Person	25.5	11.8	31.3	33.7
Rider	6.7	0	12.2	20.3
Car	57.8	36.8	64.5	64.8
Truck	4.6	1	9.8	11.5
Bus	6.2	0	17	18.9
Train	0.4	0	6.1	7
Motorcycle	3.3	0	4.6	10.6
Bicycle	20.5	0	24.2	29
iIoU on category-level				
Mean iIoU	43.4	26	48.9	50.7
Human	29.8	13.5	35.1	38.1
Vehicle	57.1	38.6	62.7	63.3

Table 11: iIoU Scores of ENet fine-tuned on different datasets and input sizes. All values refer to the evaluation on the Cityscapes validation dataset, except the first column which shows IoU scores of original ENet model evaluated on the test dataset. C stands for Cityscapes and G1 for GTA part 1. G2 includes GTA part 1 and part 2. G9 includes all parts form part 1 to part 9.

Input size	1024 x512 (test set)	1024 x512	800x400			
Dataset	C		C	C+G1	C+G2	C+G9
Infer memory (MB)	-	2255	1733			
Infer time (ms)	-	260	173			
iIoU on class-level						
Mean iIoU	34.4	28.4	25	24.4	29.3	32.1
Person	47.6	50.2	43.3	43.1	43.6	42.6
Rider	20.8	31.2	28.9	28	27	25.6
Car	80	73.7	67.3	67.3	69.6	69.3
Truck	17.5	11.2	11	11.6	19.7	19.8
Bus	26.8	0.1	0	0	24	33.1
Train	21.8	0	0	0	0	16
Motorcycle	20.9	20.6	17.2	16.2	18	19
Bicycle	39.4	40	32.4	32.1	32.6	31.4
iIoU on category-level						
Mean iIoU	79.5	63.2	45.8	57.2	56.7	57.3
Human	70.4	54.2	31.6	47.8	47.2	47.3
Vehicle	88.6	72.2	60	66.6	66.2	67.2

Table 12: iIoU scores of two ENet models first fine-tuned on combined dataset of **Cityscapes and GTA part 1 to part 9**, then on **Cityscapes and GTA part 1**. *Learning rate* refers to the learning rate at the beginning of each training. The model was evaluated on Cityscapes evaluation dataset. C stands for Cityscapes and G1 for GTA part 1.

Input size	800x400	
Learning rate	5e-4	5e-5
Dataset	C+G1	C+G1
Infer memory (MB)	1733	
Infer time (ms)	173	
iIoU on class-level		
Mean iIoU	32.8	33.7
Person	44	45.4
Rider	28.3	28.5
Car	69.5	70.4
Truck	23.5	21.9
Bus	33.6	34.7
Train	12.5	14
Motorcycle	18.2	19.9
Bicycle	33.2	34.4
iIoU on category-level		
Mean iIoU	57.7	58.7
Human	48	49.1
Vehicle	67.5	68.4

References

[1] Doaa A Shoieb, Sherin Youssef, and Walid Aly. Computer-aided model for skin diagnosis using deep learning. 4:116–121, 12 2016.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[3] Léon Bottou. *Large-Scale Machine Learning with Stochastic Gradient Descent*, pages 177–186. Physica-Verlag HD, Heidelberg, 2010.

[4] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella, and Juergen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2843–2851. Curran Associates, Inc., 2012.

[5] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *CoRR*, abs/1003.0358, 2010.

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

[7] Li Deng and Dong Yu. Deep convex network: A scalable architecture for speech pattern classification. In *Interspeech*, August 2011.

[8] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. 03 2016.

[9] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *CoRR*, abs/1411.4734, 2014.

[10] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *CoRR*, abs/1411.4734, 2014.

[11] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, jan 2015. DOI: 10.1007/s11263-014-0733-5.

[12] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, Aug 2013.

[13] A G. Ivakhnenko and V G. Lapa. Cybernetic predicting devices. page 250, 04 1966.

[14] Richard Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. 405:947–51, 07 2000.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[17] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, March 1968.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[19] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. 30:88–97, 01 2009.

[20] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[21] A. Karpathy. Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io/neural-networks-1/`. Accessed: 2017-11-01.

[22] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *CoRR*, abs/1511.02680, 2015.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[26] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[27] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010.

[28] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

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

[30] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

[31] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[32] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[34] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[37] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[38] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. *CoRR*, abs/1411.4280, 2014.

[39] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[40] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.

[41] M. Zeiler. Clarifai, 2013.

[42] Matthew D. Zeiler and Rob Fergus. *CoRR*, abs/1311.2901, 2013.

[43] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr. Conditional random fields as recurrent neural networks. *CoRR*, abs/1502.03240, 2015.