Learning depth-based semantic segmentation of street scenes

Kangkan Jyoti Bharadwaj
Examiner1: Prof. Dr. Thomas Brox
Examiner2: Prof. Dr. Alexander Reiterer
Advisers: Christian Koch, Dominik Störk

Chair of Image Processing and Pattern Recognition
Department of Computer Science
Faculty of Engineering
Albert-Ludwigs-University Freiburg

05-Dec-2018

Writing Period

07.06.2018 - 07.12.2018

Examiner

Prof. Dr. Thomas Brox

Second Examiner

Prof. Dr. Alexander Reiterer

Advisers

Christian Koch, Dominik Störk

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare, that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

This work addresses multi-class semantic segmentation of street scenes by exploring depth information with RGB data. Our dataset comprises of street images from Berlin taken from four different camera angles and scanned using a laser scanner and later processed to create the depth images from 3D point clouds by projection. Our work also proposes an architecture model comprising of a Residual Network as an encoder and a UNet decoder for the Berlin set that learns good quality feature representation. We achieve a mean accuracy of 58.35%, mean pixel accuracy of 94.36% and mean IOU (Intersection over Union) of 51.91% on the test set. We further analyze the benefits that the model exhibits on certain classes when trained including depth to the RGB data with that of the model based only on RGB information. An alternative approach of feeding the depth information using a separate encoder was carried out to study the performance variation in segmentation and if it can bring any significant hike to it's quality. And finally we draw a performance contrast of our network to one of the state-of-the-art models on our dataset.

Contents

List of Figures

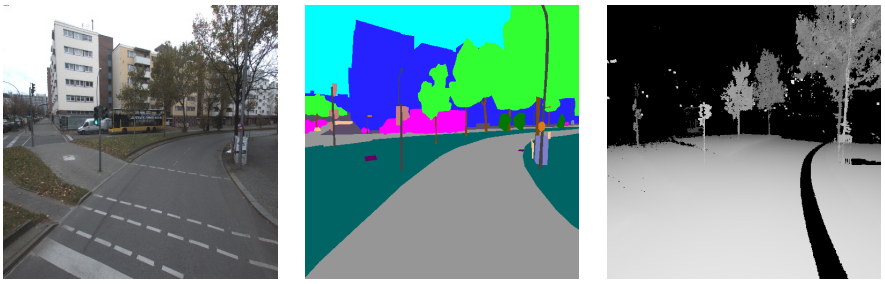
List of Tables

1 Introduction

Semantic segmentation refers to subdividing an image into multiple segments or classes each containing sets of pixels. Each of these pixel sets is assigned a label or a class in a way that pixels with the same class exhibit common visual properties like brightness, color, texture etc. This results in a compact representation of the image which is easier to analyze and interpret. In contrast, *segmentation* aims at partitioning an image into coherent regions without explicitly attempting to draw any relevant representations of these regions [60]. The success of image analysis depends on the quality of segmentation and accurate segmentation is a very challenging process. Semantic segmentation plays vital roles in autonomous driving, medical imaging, geo-sensing, face recognition etc.

In this paper, the approach used to address the problem makes use of *depth images* along with standard RGB images to improve the segmentation quality. The depth is another channel here that provides information about the distance of the surface of an object from a viewpoint (camera).

An example of sample data with depth is shown below:



(a) Image (b) False colored mask (c) Depth

Figure 1: A sample data where an image (left) is segmented (center) with it's corresponding depth map (right). The segmented image is colored for ease of visualization. Every color represents an independent region (class). For example, The color blue represent the class *building*, grey represents the class *asphalt/road*, light-green represents *trees*, violet is for the class *hole cover* etc.

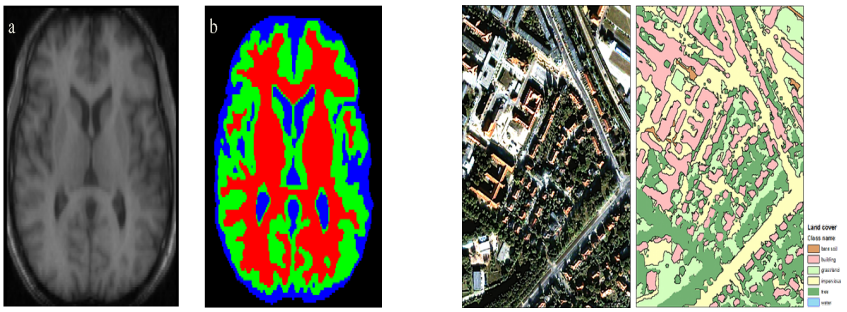
Our segmentation is learned using a residual network. Residual Networks have been very successful in the recent past in the fields of classification, detection, segmentation and recognition. The capability of preserving information across layers make the architecture very robust to handle deep hierarchies and maintain good training performance compared to conventional ConvNets that can wind up with the *Vanishing gradient problem.* The vanishing gradient refers to the extremely small gradient values that reflects in a marginal change in the weight updates or no update at all. Eventually the network stops learning. Hence higher expressibility can be achieved through deep layers creating higher

abstractions. Moreover, several layers in a ResNet are confined to abstract similar features which results in preserving information for future analysis if needed. These information can be discarded or used by the network for making the final decision.

The conventional belief of hierarchical dependency in a stack of network layers were restricted to adjacency which was overridden by ResNets later. Possibility of information requirement by a layer may arise not only from its previous neighbour but also from ones way higher in the hierarchy. To overcome this shortcoming, pass through connections were introduced preventing information loss. This way layers can receive more detailed information than just abstract information.

1.1 Motivation

Image segmentation has become a key approach in solving complex problems in many diverse industrial fields like automotive (automobile industry involved in the design, manufacture, selling motor vehicles [76]), construction (industries involved in construction of building or infrastructure [78]), medical, electronics, media, remote sensing (acquiring information about an object without making a physical contact [86]), space imagery (processing of astronomical images beyond the solar system [87]), telecommunication (exchange of information using electronic medium [88]) etc. For a better perception of these application, we illustrate two examples of segmentation in the figure 2 below.



(a) Medical image segmentation [89] (b) Satellite image segmentation [74]

Figure 2: Medical image segmentation on the left that comprises of two images (image and corresponding segmentation of the brain). The red color in the segmented image corresponds to the white matter in the brain, green color represents the gray matter and blue represents the cerebrospinal fluid in the brain[89]. To the right are two aerial images (image and corresponding segmentation of the city Rostock) showcasing aerial image segmentation. The orange color in the segmented image represents the class *Bare soil*, pink represents *Building*, light green represents *Grassland*, blue represents *Water* and green represents *Trees* [74]

Our work of street image segmentation applies to the field of telecommunication. A fully segmented map of a city can help service providers to automize the process of route planning for deploying underground cables. As a service provider, most of the the signal routing is carried out using optical fibres. These are glass cored optics for high speed communication with very little loss of signal strength. Deploying these in itself is an expensive business and taking into account the materials to deracinate on particular regions, the additional costs can only soar up the capital investment. The complexity of the task depends on the characteristics of that region. This is because, depending on areas to deploy the cables, there can be trees, light posts, traffic lights, different kinds of pavement that can add additional costs to remove them. On top of that, to investigate these regions with correct measurements, we need skilled human resource. In order to optimize this whole process from the point of capital, time and manpower, images are collected from all over the city and are semantically segmented. The point clouds generated from the laser scanner are further mapped to these segmented images resulting in segmented 3D maps (point clouds). These individually processed regions are clubbed together from the whole city resulting in an efficient 3D map of the city. An illustration of this map is provided below.

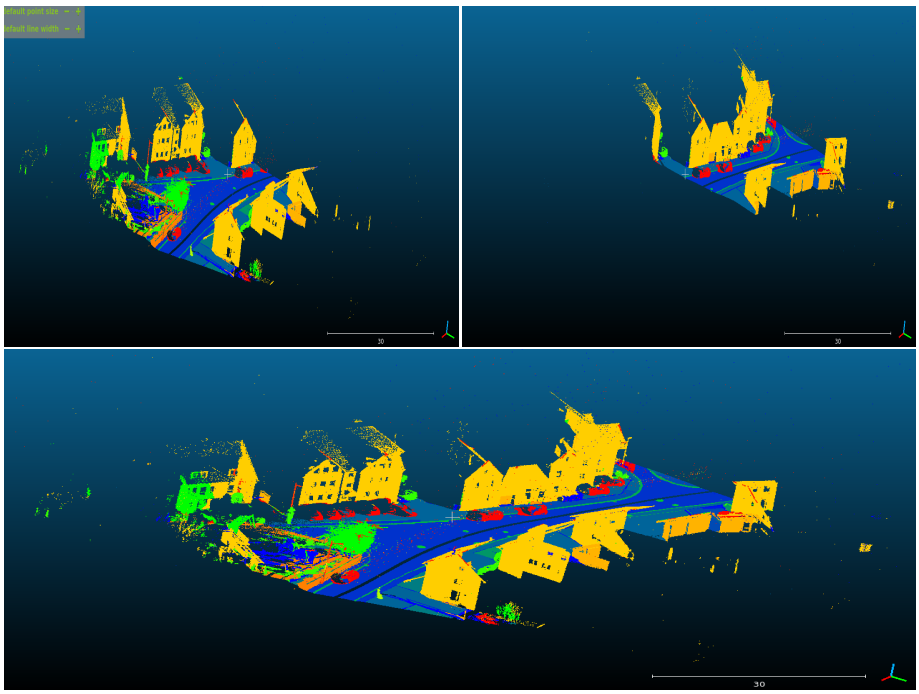


Figure 3: The top row contains independently processed cloud components of parts of a city and the second row illustrates a joined map of these components.

This map can be used for efficient route planning that can help service providers to plan in advance to avoid areas that can be expensive or impossible to deploy cables underground resulting in an optimized utilization of time and minimum involvement of human resource.

1.2 Goal & Contributions

Our work focuses on:

1. Achieving quality segmentation using RGB-D data
2. Making a comparison study of RGB-D segmentation to RGB
3. Exploring an alternative approach of feeding depth and analyze it's viability
4. A final comparison of our model to a state-of-the-art

The project is aimed to achieve high quality segmentation with optimized use of resources. The baseline architecture FCN8, initially used for segmentation is replaced by state-of-the-art residual network which is further customized in an attempt to attain performance gain. The depth information is included along with RGB data as input source to analyze the benefits the model can draw on the individual classes over a RGB model. This provides an insight to how helpful the depth can be in street scenes segmentation and it's significance in achieving considerable rise in quality. We explore an alternative approach to feed the depth images using a separate encoder and fuse them after activation before every downsampling. Fusion was studied in two different ways using element-wise summation and concatenation. The former fusion strategy was adopted from [28]. The major pros and cons of both the approaches were studied to conclude if any of them can be applicable in segmentation for real world problems. A current state-of-the-art model is tested on our dataset to examine the qualitative variation of both the models and examine the effect of varied width and depth of the models on the data. We also checked if involvement of dense connections along with residual connections can have an advantage over our model as our model is built only on residual blocks. The performance of the state-of-the-art was studied in detail by introducing topological modifications in it's architecture.

2 Related Work

Image segmentation has been studied since 40 years when it was first used in detecting edges in two different parts of an image [94] using an operator known

as *Roberts Edge detector* [90] in 1965. Since then, the area has evolved rapidly and most of it's applications today are carried out using "Convolutional Neural Network".

2.1 Fully Convolution Networks for Semantic Segmentation

The CNN (Convolutional Neural Networks) based approach for semantic segmentation was designed in 2015 by [45] that was independent of fully connected layers. Arbitrary size images were used to produce corresponding sizes of segmentation masks which surpassed the technological constraints of its historic counterparts. Introduction of *Pooling layers* improved computational efficiency and reduced parameter usage. The pooling layer down scales the image into smaller resolution. A Fully Convolution Network was used to train and infer on whole image at a time. Training was performed per pixel wise through forward and backward propagation. Further learnable upsampling was used to restore the image to its original resolution.

Transforming Classification networks into Fully Convolution Network by connecting back the coarse outputs from Classification model to corresponding pixels paved way for *Deconvolution*. Deconvolution is popularly known as *Transposed Convolution* which performs convolution with extra padding. This upsampling is performed using learnable parameters end-to-end in a network through back propragation, though bilinear upsampling was another choice for deconvolution.

The figure below shows the architecture of the model.

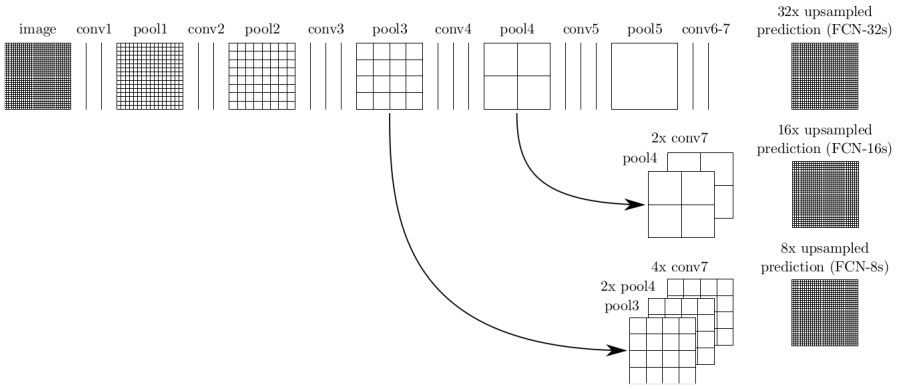


Figure 4: FCN architecture from [45]

The architecture has an encoder and a decoder. It holds 18 Convolution layers and 3 Deconvolution layers. The encoder holds 16 and the rest two are

held by the decoder. Once the image is fed, the first two stages in the encoder are followed by 2 convolution layers and subsequent pooling and *ReLu* activation. The next three stages hold 3 convolution layers each followed by pooling and Relu activation. Each pooling down scales the image to half of its resolution. The sixth stage in the encoder convolves twice each followed by a *Dropout* layer and upsampled to the original (coarse) map using 32 pixel stride before moving to the prediction layer.

To work this around, links were introduced from mid of the encoder where the shallow network that worked on finer details were merged to the prediction layer producing coarse details that helps the network to make local predictions in context to the global region. Moving from bottom to top, the first link was created from the pooling in the fourth stage $Pool4$ where a convolution of $1X1$ was introduced on top of that and output was fused to the final prediction from $Conv7$ at stride 32 by adding a $2X$ *Deconvolution* layer. The second link was created for pooling from the third stage $Pool3$ in the encoder where another $1X1$ convolution was introduced fusing the output to the the former prediction at stride 16 in the similar way.

1. **FCN-32** : Segmentation map from conv7 [1], transposed convolution with stride 32.
2. **FCN-16** : Sums the $2X$ upsampled prediction from conv7 [1] with pool4, use transposed convolution with stride 16 on top of that to produce segmentation map.
3. **FCN-8** : Upsample map from FCN-16 [1] with stride 2 transposed convolution and sums them with pool3, another transposed convolution with stride 8 to produce final segmentation map.

2.2 Segmentation of indoor scenes using depth

The work of [14] is the first attempt in exploiting depth in a learning based approach for semantic segmentation. The use of multi-scale convolution neural networks was a break through that surpassed the results of the then state-of-the-art achieving almost a 15% gain in segmentation accuracy for certain classes. The segmentation was performed in NYU-v2 depth dataset from [65]. The RGB input was normalized to have 0 mean and unit variance and processed using a Laplacian Pyramid from [11] to produce the three scales of input for the ConvNet. The convolution network had 3 stages of convolution, activation and pooling followed by a classification layer. The three scales are fed to three copies of the same network. The feature maps from all the stages are concatenated and necessary upsampling is performed for coarse maps produced from lower scales. The number of output channels per stage are $16, 64, 256$ and the kernels are $7X7$ for convolution layer and $2X2$ for pooling. The size of the input images are $240X320$. The classifier is a negative log likelihood loss function. In parallel to this, superpixel computation [49], [63], [31] is performed to

produce an independent segmentation map and these are further combined to the classifier prediction to improve the accuracy as a post-processing step. Superpixels are basically a process of combining pixels that are similar depicting a single feature rather than working with single pixels. For example features like ear,nose,wheel,tree,road etc are different classes of super-pixels. An illustration of the approach is shown below.

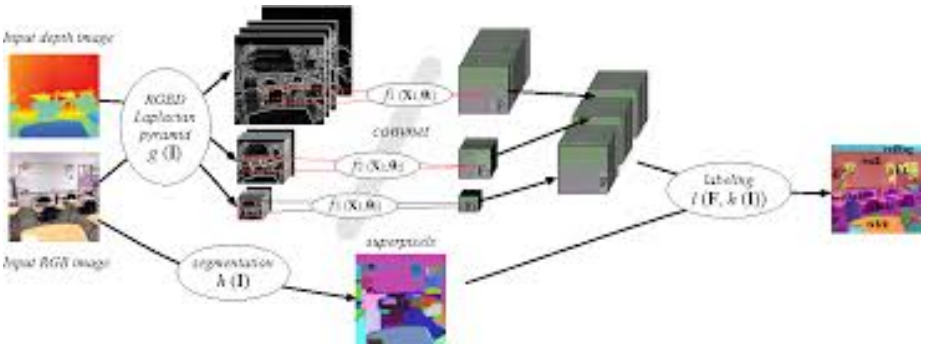


Figure 5: Multi-scale feature extraction using ConvNet from [14]

2.3 UNet

UNet is a convolutional neural network that is used extensively in bio-medical image segmentation from [51]. It consists of a contracting part known as an encoder to manipulate the image and a expanding part called decoder to produce the final full resolution image. Each analysis and synthesis part contains 4 levels. Each level in the analysis part contains $3X3$ convolutions each followed by rectified linear unit (ReLu), a $2X2$ Max Pooling with strides of two for downsampling. At each downsampling stage, the number of feature channels are doubled. In the synthesis part, each level consists of an up-convolution of $2X2$ where the feature outputs are halved of $2X2$ followed by strides of two and concatenating the feature from the corresponding contracting part, again followed by two $3X3$ convolutions, each followed by a ReLu and a weight initialization scheme "msra". A final $1X1$ convolution is used to get the complete image. It has overall 18 convolution layers.

The main advantage that U-Net provides is the feasibility of training small amount of data as the network comes with its own data augmentation capabilities to artificially increment data for the network that most neural network misses out.

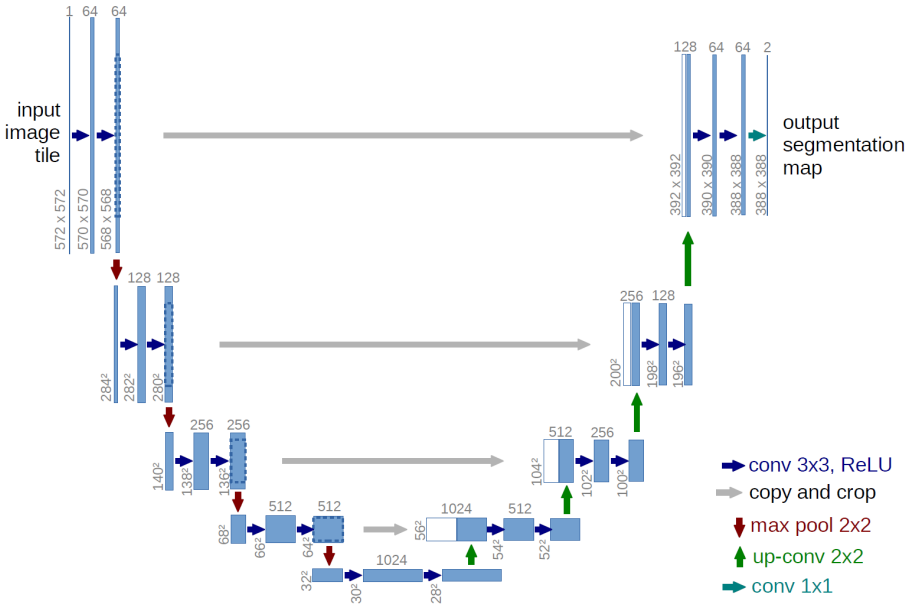


Figure 6: U-Net from [51]

2.4 Deep Residual Learning for Image Recognition

Microsoft Research took the world of deep learning by storm when they came up with the concept of Deep Residual Networks in [29] in 2014. The underlying working principle of ResNets made it possible to train neural networks that were even more than 1000 layers deep without any degradation of performance in training, contrary to conventional ConvNets that wears out with increasing depth.

The key idea to enhance network robustness to increasing depth was introduction of skip connections from a standard layer that bypasses couple of next layers giving rise to popularly known as a *Residual Block*. These connections can be inserted in any standard ConvNet turning it into a ResNet. An example diagram is shown below.

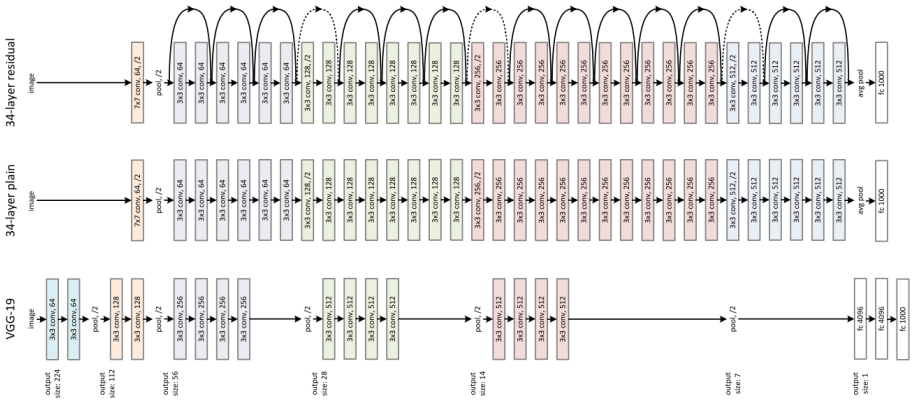


Figure 7: ConvNet and it's corresponding Residual counterpart from [29]

With the increasing depth of a ConvNet the performance starts degrading, its accuracy gets saturated and this is not caused due to overfitting but instead optimization gets difficult. So the added layers were made identity mapping. Hence, instead of learning a direct mapping from input to output, a residual function was introduced that was easier to optimize. If the identity as was optimal, the the residual function was pushed to zero leaving input = output. Hence the final model cannot produce training error greater than its original shallow counterpart. In simple terms if X is input and Y is output after a mapping of $F(X)$ which is basically a series of convolution layers then the equation for this mapping is:

$$Y = F(X) \tag{1}$$

Now introducing a skip connection from input X to straight Y and adding up the incoming inputs to Y we get:

$$Y = F(X) + X \tag{2}$$

These identity mapping does not add any extra computational overhead to the network learn. Optimization of the network becomes very cheap and the problem of *Vanishing Gradient* is dealt with in a smarter way where the gradient has a shortcut to in the backward path that can skip the intermediate steps to reach the top. A visual representation is illustrated below.

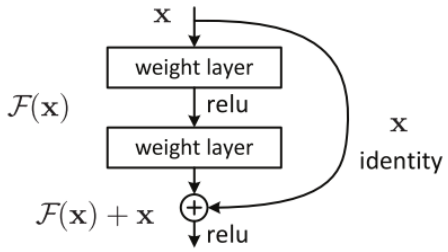


Figure 8: Identity mapping in a Residual Network from [29]

A standard residual block is shown below with the shortcut connections.

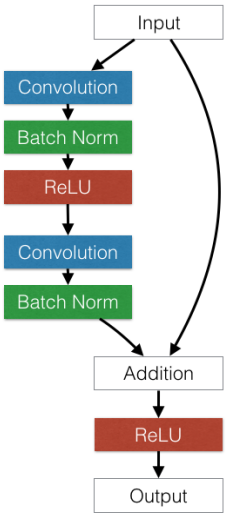


Figure 9: A standard residual block from [57]

The two types of residual blocks are:

1. Basic residual block
2. Bottleneck residual block

The bottleneck block is used in deeper networks for economical computation but performance of both the blocks are similar.

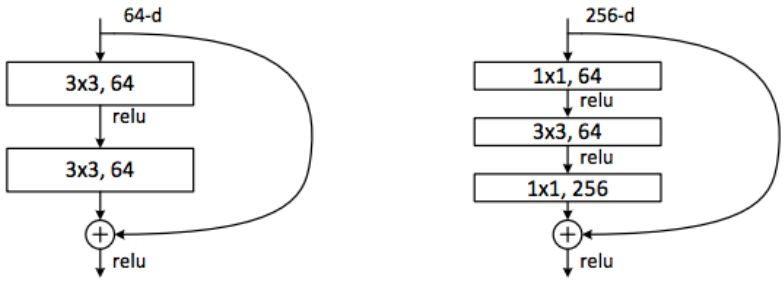


Figure 10: Basic block (left) Bottleneck block (right) from [20]

The common residual network architectures has 18, 34, 50, 101 or 152 convolution layers. The residual blocks used in the deep ResNets are bottleneck blocks that are computationally efficient but the shallow counterparts contain the basic blocks. The architectures start with a common $7X7$ *Convolution* with a stride of 2 followed by $3X3$ *Max Pool* with a stride of 2. The basic block contains two $3X3$ *Convolution* layers followed by *ReLu* activation and *Batch Normalization* wher as the bottleneck contains 3 *Convolution* layers of $1X1$, $3X3$ and $1X1$ each followed by *ReLu* and *Batch Norm*. Finally an *Average pooling* to wrap up the architecture. A compact visual representation is shown below.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 11: ResNet architectures from [29]

2.5 FuseNet

Another variation of feeding depth information using a fusion based approach was implemented by [28] where two separate encoders were used to work on the input data. More precisely, one encoder was used for RGB and the second one for depth. The depth was independently processed and fused using element wise

operation. Two kinds of fusion strategies were carried out, *sparse fusion* where the depth was fused before every downsampling or pooling operation and *dense fusion* where depth was fused after every Convolution, Batch Normalization and ReLu. An illustration of the architecture used is given below:

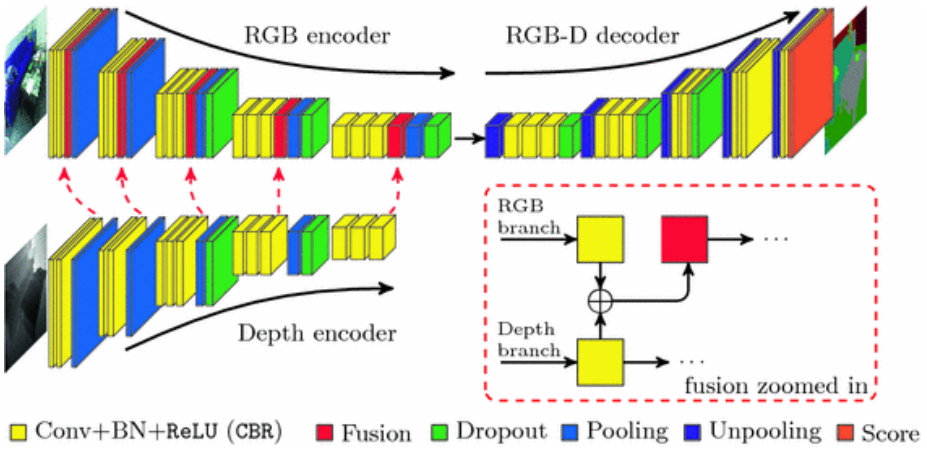
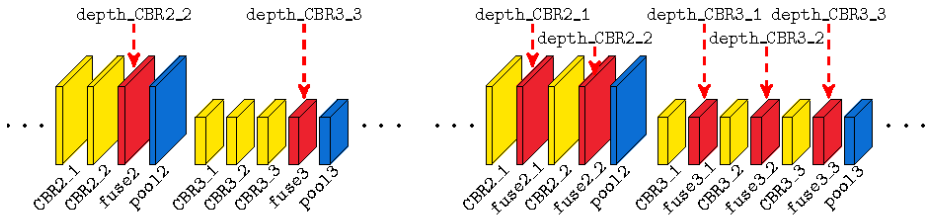


Figure 12: FuseNet architecture from [28]

Referring to the figure above, the encoder resembles VGG-16 without the fully connected layers. They used batch normalization after each convolution and also used dropout in both encoder and decoder. Activation was ReLu. As for the decoder, unpooling operation was used for upsampling with the continuation of convolution, batch norm and ReLu activation. The fusion strategy as depicted in the architecture above is sparse fusion. The proposed network architecture was evaluated on SUN RGB-D dataset from [68] where they claim to achieve state-of-the-art results to that of the early fusion (Stacked RGB-D) approach of RGB-D segmentation. An illustration of the fusion strategies is shown below.



(a) Sparse fusion (SF) (b) Dense fusion (DF)

Figure 13: Fusion strategies from [28]

2.6 DDNet

DDNet is an extension of so called DPDB Net from [48] which is an abbreviation of *Dual-path Dense-Block Network* with certain modifications in the decoder. DDNet stands of *Deep Decoder Network*. To start with, DPDB mainly focuses on exploiting the advantages of both the *DenseNet* and *ResNet*. DenseNets and ResNets have been very successful in multiple classification tasks with state-of-the-art performance. Combining these two working principles can generate a very powerful encoder that can not only reuse features but also explore new ones. The deocoder however is composed of only dense blocks as residual blocks' feature reusablity leads to feature map explosion.

2.6.1 DPDB Block

For a ResNet we have,

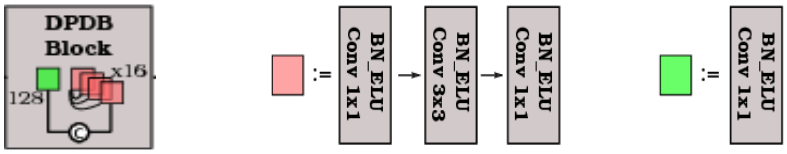
$$Y1 = F(X) + X \tag{3}$$

For a DenseNet we have,

$$Y2 = F(X, X-1, X-2...X-n) \tag{4}$$

The fusion of the output from Residual path and Dense path through concatenation produces:

$$Z = H(Y1 + Y2) \tag{5}$$



(a) Dual-Path Dense-Block (b) Underlying layer configuration

Figure 14: DPDB block from [48]

2.6.2 Deep Decoder Block

The deep decoder is composed of smaller networks and skip connections [48] that runs between encoder-decoder, decoder-encoder and decoder-decoder. These skip connections allow high level information to flow in the network that enables recovery of high-resolution information.

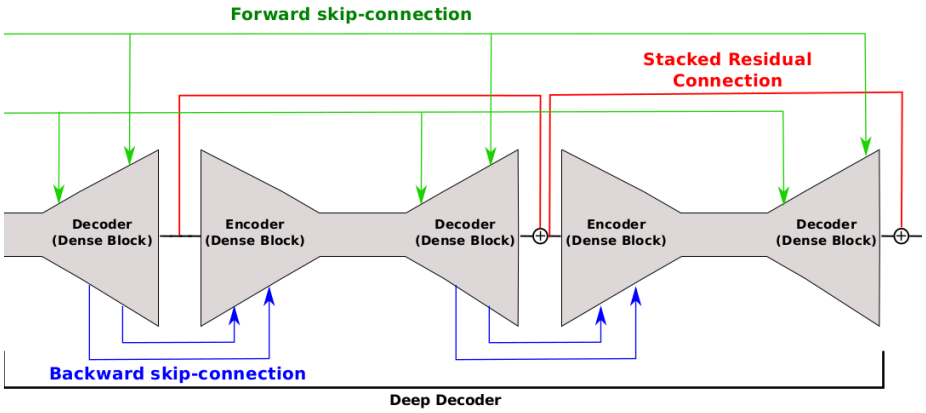


Figure 15: Deep Decoder [48]

The architecture of DPDB Net contains an encoder and a decoder. The encoder is composed of *DPDB* blocks and *Transition Down* operation. The residual part of DPDB block contains $1X1$ Convolution, Batch Norm and ELU activation and the dense part contains $1X1$ Convolution, Batch Norm, ELU, $3X3$ Convolution, Batch Norm, ELU and $1X1$ Convolution, Batch Norm, ELU. The *Transition Down* includes Max Pool, Dropout, Convolution, ELU activation and Batch Norm. The decoder in DDNet is a combination of shallow networks with encoders and decoders. This is to emphasize more on the effective upscaling of low resolution features back to high resolution from the scaled down versions as an key aspect in image segmentation.

The blocks here are of bottleneck style. The network has 4 stages of downsampling and subsequent upsampling. The decoder in the DDNet has only 3 stages of the upsampling and downsampling. The skip connections are of three types:

1. Standard forward connection from encoders to respective resolution decoders
2. Backward skip connection for adjacent encoder and decoder
2. Stacked residual connection for to enhance information flow

The DPDB Net is shown below:

Network	Output	DPDB Full
Input	360×480	-
Conv1	360×480	7×7, 96
DPDB_1	360×480	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 240 \end{bmatrix} \times 4$
DPDB_2	180×240	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 384 \end{bmatrix} \times 5$
DPDB_3	90×120	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 832 \end{bmatrix} \times 7$
DPDB_4	45×60	$\begin{bmatrix} 1 \times 1, 1024 \\ 3 \times 3, 1024 \\ 1 \times 1, 1336 \end{bmatrix} \times 10$
DPDB_5	22×30	$\begin{bmatrix} 1 \times 1, 1024 \\ 3 \times 3, 1024 \\ 1 \times 1, 1984 \end{bmatrix} \times 12$
DB_1	22×30	$\begin{bmatrix} 1 \times 1, 912 \\ 3 \times 3, 16 \end{bmatrix} \times 12$
DB_2	45×60	$\begin{bmatrix} 1 \times 1, 480 \\ 3 \times 3, 16 \end{bmatrix} \times 10$
DB_3	90×120	$\begin{bmatrix} 1 \times 1, 320 \\ 3 \times 3, 16 \end{bmatrix} \times 7$
DB_4	180×240	$\begin{bmatrix} 1 \times 1, 208 \\ 3 \times 3, 16 \end{bmatrix} \times 5$
DB_5	360×480	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 16 \end{bmatrix} \times 4$
Conv Final	360×480	1×1, N_{cl}

Figure 16: DPDB Net [48]

3 Background

This section focuses on the basic machine learning concepts.

3.1 Machine Learning

Machine Learning [69] is an application of artificial intelligence that allows systems to learn and grow based on experience with the environment. The environment can be composed of data that the system interacts with to learn decision making or observations where the system learns to draw behavioural patterns among the data without being explicitly programmed. Here, [24] the machine feeds on data to widen it's information space from which it can predict possible outcomes through statistical analysis. Learning improvement is made from growing input data source. There is no explicit programming of the machine to work on achieving the desired goal. As define by [36], *"Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions."*

Any machine learning algorithms consists of three components [69]:

1. **Model** that takes decisions
2. **Parameters or weights** that are learnable based on which actions are carried out
3. **Learner** that adjust these weights by based on contrast between true data and predictions from the model.

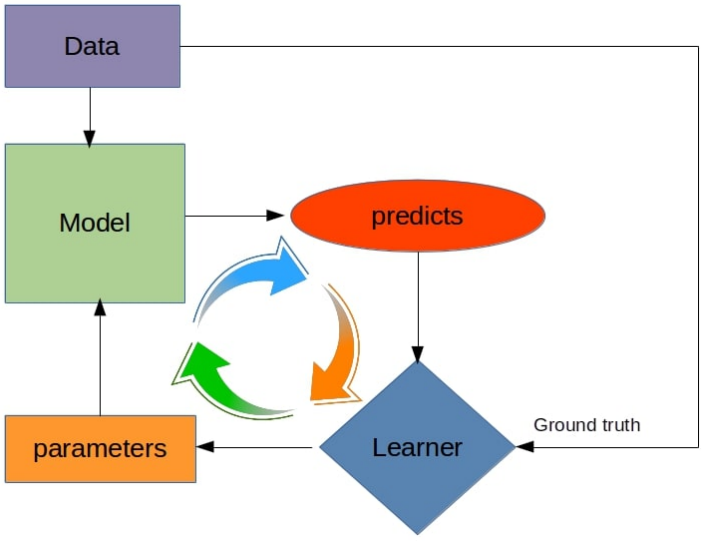


Figure 17: Basic Machine Learning process

The fundamental working principle of a machine learning process depends on the *learning system*. The learning system operates by minimizing a loss function in an attempt to arrive at a global minima using *gradient descent*. In this process, a forward pass is first carried out where the input into the model produces some output. Based on the ground truth, the output is analyzed to compute the variability between the expected data and the predicted one using the loss function. The learner as a part of the backward pass then starts adjusting the weights that contributes to the whole learning and final output in an attempt to minimize the loss in the next forward pass. This is a repetitive process as the model iterates until convergence when no more learning is possible.

According to [82] *Machine Learning* was a product of Artificial Intelligence back in 1959 when an expert in the field of computer gaming named Arthur Samuel working at IBM came up with the term. The endeavour during that period was to make machines learn from fed data and this was attempted using some symbolic methods what was termed then *Neural Networks*. These so called networks were composed of perceptrons. Probability theory was also integrated as a part of the machine learning process.

Machine Learning hit it off in the 90s as probabilistic reasoning became an important part of the whole learning process, and with the growing advent of the Internet it made use of surplus availability of information as a fuel to boost its growth in different applications. ML and Data Mining are two aspects of knowledge exploration that can go hand in hand. Data Mining is more focused on retrieving new information, exploring new properties of data whereas ML can only make use of available knowledge to learn and predict the same in new data. It always relies on available knowledge and can make use of data mining to achieve its target in an unsupervised manner. On the other hand data mining do make use of machine leaning methods with varied purposes but the working principle and their corresponding goals does not coincide. The capacity of a machine in learning a task is evaluated in terms of available knowledge whereas evaluating the performance of data mining approach relies on discovering new information and its relative importance in the practical applications.

ML also employs *optimization* techniques that makes us of *loss functions*. A loss function represents the differences between the original feature and the one predicted by the learning process. It tells the system how far is it from learning and representing the actual knowledge on a data accurately. Optimization on the other hand is a process of minimizing the loss in a way that the discrepancy between the original information and predicted one minimizes over time. The idea of ML is to built a generalized model on a set of data that can work with unseen data having the same characteristics.

Machine Learning approaches are broadly categorized in three different types:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforced Learning

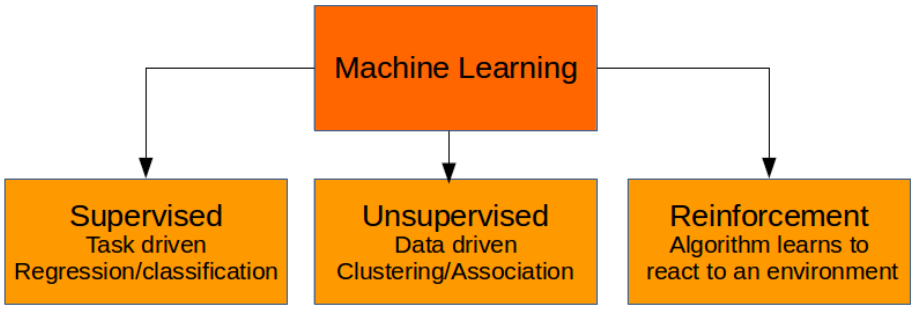


Figure 18: Machine Learning approaches [58]

3.1.1 Supervised Learning

Majority of the machine learning applications are driven in a supervised manner where the whole learning of addressing a problem is guided by data analyst or a machine learning expert. The data to be learned is pre-processed and any corruption or discrepancy is removed. The desired output is also fed and the whole learning is guided by optimization and evaluation. What to learn is decided by the teacher here. In a simple way, this can be explained as consider an input X and desired output Y [10].

$$Y = F(X) \tag{6}$$

Make use of an algorithm that maps X to Y as accurately as possible so that it can generalize to new X.

Supervised Learning are of two types:

1. Regression
2. Classification

3.1.1.1 Regression

Regression [8] refers to approximation of a mapping function from input x to a continuous output y. The output is a real value like an integer or floating point. For example height, weight, currency, size etc.

3.1.1.2 Classification

Classification is a process of associating a class with each sample in a given set of data. Here the output are categories, labels, regions like different colors, vehicles, species etc. Segmentation is a type of Image Classification. Classifiers are of different type [64] [4]. They are *Linear Classifiers* which are Bayes classifiers [54] that functions by computing posterior probability assuming conditional independence among features. We have *Decision Trees* where the task is modelled as a tree with decision nodes and leaf nodes, *Random Forest* that operates by taking the mean prediction from an ensemble of decision trees, *k-Nearest Neighbour* which classifies the input based on it's k nearest neighboring samples. Finally we have *Artificial Neural Networks* that we make use of in our project and is explained in the subsequent section.

Artificial Neural Networks Neural networks are a simulation of the functioning of the brain with interconnected neurons or perceptrons with associated weights. The neurons are arranged in layers and the network consists of input, output and hidden layers. The model learns by adjusting the weights to fit to a given problem definition. Neural networks [22] can be Feedforward network, Convolution neural network, Recurrent neural network etc depending on the problem definition.

In a Feedforward network as explained by [93], [59], the connections between the neurons do not form a cycle. This is the simplest form of neural network as illustrated below:

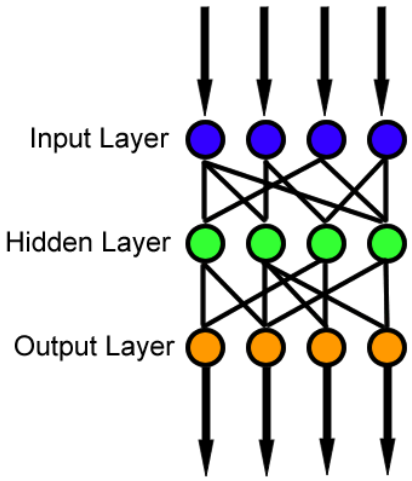


Figure 19: Working principle of a Feedforward network [79]

Feedforward neural networks can be *Single-layer perceptron* [79] where the network contains an input layer and output layer and the input is fed directly to output. Neurons are activated or fired if the sum of the products of neurons and weights go above a threshold. The threshold is normally 0 and activated and deactivated values are -1&1. It can also be *Multilayer perceptron* [79] where the model contains many hidden layers and neurons in one layer forward information to the neurons in the next layer. Generally the activation function used here is Sigmoid activation.

Recurrent neural networks are models with memory, holding information about past learning on specific tasks. These models come very handy when applied to solving problems with sequential information like speech recognition like in [55] and [44], handwriting recognition like in [7] etc. The nodes in the network form directed graphs. The output from one node depends not only on it's immediate input but also on the history of previous computations. An illustration of an unfolded Recurrent network is showcased below:

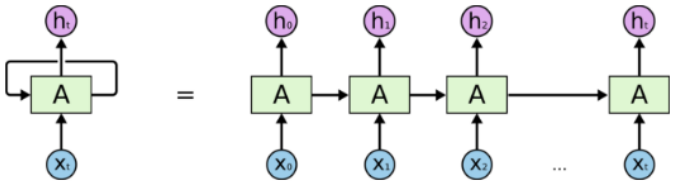


Figure 20: Unfolded Recurrent neural network from [6]

Finally, deep neural networks consisting of numerous hidden layers showed most promising results in handling real world problems with high tolerance to noise. But they take longer time to train and works well with continuous data. A more comprehensive study of these networks are outlined in the section 3.2.

3.1.2 Unsupervised Learning

In this form of learning [2] [42] there is no teacher to evaluate right and wrong. The model thus is not trained on mapping approximation from $X \rightarrow Y$. The model is provided only with input data but no desired output. The idea is to study and understand the underlying data distribution in the input and come up with own conclusions for the output. Unsupervised can be categorized in two different ways namely *Clustering* where the data distribution is modelled as groups of data that share similar properties and behaviour and *Association* where the algorithm studies the relation between X and Y to associate one data to another.

3.1.3 Reinforcement Learning

This is an iterative process of attempting to solve a problem where numerous attempts are made based on the condition that any favourable outcome will be rewarded and the undesired ones are penalized [52]. These are goal-oriented algorithms, they learn to perform complex operations over repetitive trials. This whole approach simulates a child's process of learning about the world outside. Making a right decision will be rewarded and a wrong one is punished. Starting from a blank state, over a period of time the model starts performing well learned from trial and error method [53].

3.2 Deep Neural Networks

Neural networks are a simulation of the brain along with the network of neurons utilized in the learning process. In case of a neural network the term neurons are coined as perceptrons. The network is a cascade of multiple layers of these perceptrons that operate on the output from the previous layer. These layers learns multi-level representations of data through multi-level abstractions using nonlinear processing, feature extraction and feature transformation. They participate in clustering, association, classification etc. A visual inference of the learning process of a neural network on multiple levels is illustrated below.

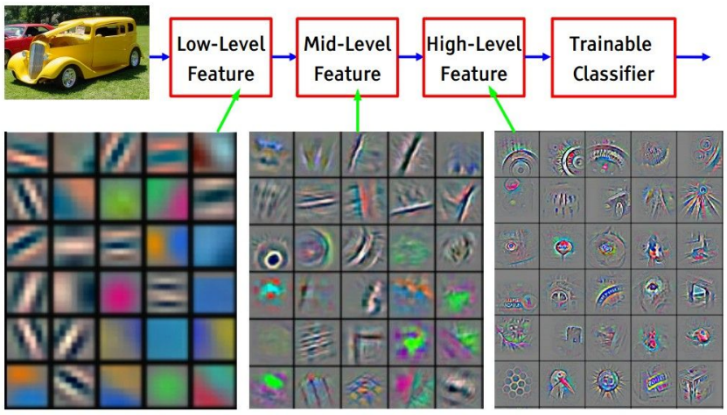


Figure 21: Feature visualization of a deep neural network [37]

3.2.1 Usage

Some common applications of Deep Neural Networks are in the fields of Bioinformatics [77] (a field that focuses on developing software tools to analyze biological

data by combining interdisciplinary fields of study like biology, computer science, mathematics etc.) to predict gene ontology annotation predictions [13] that focuses on locating the gene regions for protein synthesis, Speech recognition [46] (ability of a system to identify and respond to different sounds produced by human), Computer games like *Chess* [66] combining machine learning with human intuition where the game is played by computer aided humans that benefits from the intuitive power of human to plan powerful strategies and the raw computational power of machines. Another well known game taken over by deep learning is *Go* popularly known as AlphaGo developed by Google DeepMind [66]. Deep neural networks are also used in autonomous driving or self driven car that is capable of sensing it's environment and navigate with no or little human input for online camera-laser calibration in arbitrary environments, [43], for object recognition using segmentation, tracking and track classification to classify objects like car, pedestrians etc [70]. Deep neural networks are also used for forecast prediction commonly known as weather prediction, E-Commerce for buying and selling of goods online, Search engines like Google or Yahoo for searching information using the World Wide Web etc.

3.2.2 Neural Network Architecture

The fundamental building block of a neural network are perceptrons (nodes). These are artificial neurons whose functions are analogous to the neurons in a human brain. A set of these perceptrons compose a *layer*. The layers are of three types namely *input layer, hidden layer and output layer*. The neurons in these layers are responsible for passing information from input layer to the output layer through a series of hidden layers. Each neuron combines the input data with the weights resulting in either amplifying or damping the significance of the information to the relative importance of learning the assigned goal. The products of input data and the weights are finally added and passed to an *activation function* which decides if the output should contribute to produce the final goal (Activation functions are responsible for firing the neurons). Considering an equation below[73],

$$Y = \sum (weight * input) + bias \tag{7}$$

the neuron does not know the bound of Y, it can range from -infinity to +infinity. Its the activation function that decides the Y limits and the extent to which a signal progresses through the network to affect the final goal. If value of Y is within the bound set by the activation function, the signal lives to pass to the next layer.

The layers in a neural network are responsible for:

1. Input Layer: Connects to input data
2. Hidden layer: Can be more than 1 layer placed between input and output layer, responsible for computation

3. output Layer: Responsible for producing the desired outcome

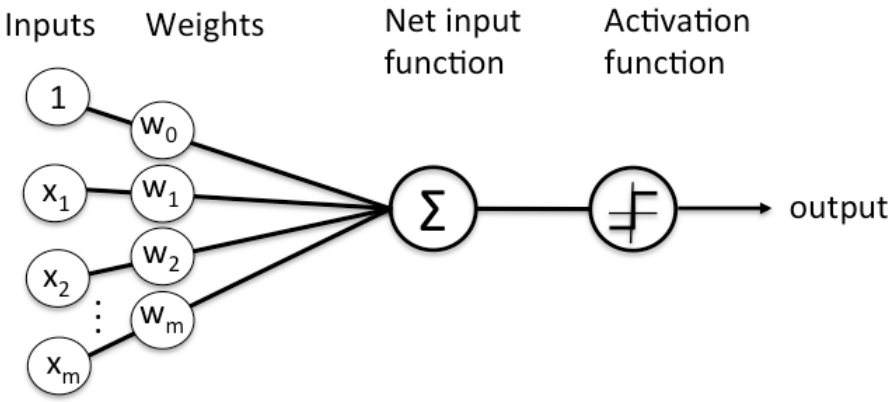


Figure 22: Neural Network Elements [67]

3.2.3 Feed-Forward and Back-Propagation

Neural Networks are forward propagating networks that take input and process it through it's stack of layers to come up with the desired output but they are also integrated with a concept of *Back propagation*. In this whole process, the network learns to recognize correlation between relevant features and optimize learning. The feed forward step can be jotted down in three steps. Consider X as input, W as weight, Y as ground truth, Z as initial prediction, P as weight's contribution to error, A as adjustment and L as error made during learning [67] :

$$X * W = Z \tag{8}$$

$$Y - Z = L \tag{9}$$

$$L * P = A \tag{10}$$

This adjustment to optimize the predicted output required the network to traverse back and learn the weights' contribution from each node upto the top of the network. This is commonly known as back propagation. Once the first stage of initial prediction is available, the network computes the differences to the ground truth and back propagates tweaking the weight parameter to try match the input to the desired output. This process is performed using *gradient computation*. Gradients in most simple form can be termed as change of a variable with respect to another variable.

3.2.4 Gradient descent

The literal meaning of gradient is slope or can also be considered as slope of a function. It is an optimization algorithm that operates iteratively to find the (local minima) minimum of a function. The algorithm aims to find the parameter values of the function to minimize a cost function. It is based on computation of partial derivatives with respect to it's inputs . To summarize the concept, consider the figure 23 below where one can imagine a ball [47] starting at any random point in the curve. With course of time, the algorithm moves the ball along the curve to the minimum position possible down the slope. Consider the initial position of the ball to be W and the new position of the ball that we want down the slope to be Y. To compute the gradient we have [21],

$$Y = W - \gamma * \nabla J(W) \tag{11}$$

Here the "minus" sign represents minimization that is moving against the gradient towards the minimum. γ is the *step size* or popularly known as *learning rate* taken at every iteration to go towards the local minima and $\nabla J(W)$ is the gradient (direction towards $J_{min}(W)$).

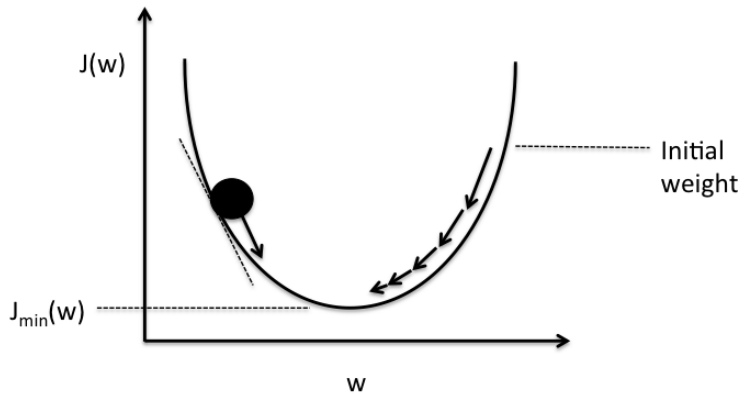


Figure 23: Working principle of Gradient Descent from [47]

The commonly used optimization schemes are *AdaDelta, AdaGrad, SGD, Adam, RMSProp* etc. Our work makes use of Adam optimizer from [40]. Adam (Adaptive Moment estimation) is a replacement to stochastic gradient descent (SGD) [50] [39] that is computationally efficient and demands less memory. But unlike SGD that maintains only one learning rate for all it's weights, Adam computes adaptive learning rates for each weight update [9] [72] so that parameters with smaller gradients receives bigger learning rates and vice-versa. Another

trick Adam brings to the table is the *momentum*. Momentum implies accumulated weight updates (previous weight update is added to the current one) in a particular direction (towards the minimum in case of the curve above in figure 23) diminishing the oscillations in the rest of the directions, hence resulting in a faster convergence.

3.2.5 Learning Rate

The learning rate plays a vital role in the learning process starting from learning the optimum solution in solving a problem to the time taken in the whole process. A good learning rate can help the gradient descent to reach the local minima in an optimized time frame. If the learning rate is too big, its likely that the learning algorithm can miss the local minima and if it is too small, it can take excessively long amount of time to reach the minimum point. One can infer these scenarios from the visualization below [21]. When the learning rate is too big, we can see that the gradient descent bounces back and forth in within the convex function in an attempt to reach the lowest point where on the other hand, a small learning rate takes a long time for the algorithm to converge to the minimum.

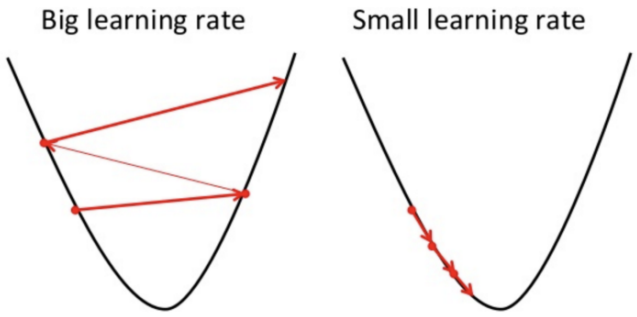


Figure 24: Gradient descent functioning with big learning rate(left) and small learning rate(right) [21]

3.2.6 Activation Functions

Activation functions are responsible for converting an input signal of a neuron to an output signal which serves as an input to the next layer in an artificial neural network. They play an important role in introducing non linearity in the network for solving complex problems. Complex problems are generally modelled as non linear functions (polynomial with degree more than 1) [75]. These non linear mappings of input to output is important to deal with data that comprises

of speech, images, audio, video etc. In the absence of an activation function the output signal is a linear function with limited expressive power (Linear Regression Model) that limits the performance. The commonly used activation functions are *ReLu, Elu, Identity, TanH, Sigmoid, LeakyRelu, Softmax* etc. For our work we used ReLU activation because it is computationally efficient than TanH and Sigmoid and works very well with vanishing gradient problem. This is better explained in the section 3.2.6.3. Couple of the rest are described below.

3.2.6.1 ELU

ELU also known as *Exponential Linear Unit* can also work with negative inputs which facilitates driving mean activation to zero and variance closer to one. This can speed up the learning process [85]. For positive inputs, it acts as an identity function but smoothens out slowly as output turns negative. ELU is expressed as:

$$R(z) = \begin{cases} z & z > 0 \\ \alpha.(e^z - 1) & z <= 0 \end{cases} \tag{12}$$

α here is a positive constant whose value is 0.01

3.2.6.2 LeakyReLU

LeakyReLU [75] [27] is an effort to fix the problem of dying ReLU as explained in the section 3.2.6.3 by introducing a negative slope to handle negative inputs. α takes generally the value 0.01. It is expressed as:

$$R(z) = \begin{cases} z & z > 0 \\ \alpha.z & z <= 0 \end{cases} \tag{13}$$

α here is a positive constant with value 0.01

3.2.6.3 ReLu

Very commonly known as *Rectified Linear*, it maps the output from a layer in neural network between 0 and 1. It is a non linear activation function. The equation can be expressed as:

$$F(x) = \max(0, x) \tag{14}$$

This is the most commonly used activation function. The function maps to 0 if input goes below 0 else it is the same as input, hence called half rectified [61]. The range is from 0 inclusive to infinity exclusive. This property has a disadvantage popularly known as the *dying ReLu problem*. If the input to the function is negative, it outputs only zero resulting in no weight updation and hence no learning as the weight derivatives are all zero. Once a model ends up in this state, it is unlikely to recover as the gradient descent algorithm no more update the weights (gradient at 0 is 0).

3.2.6.4 Sigmoid

Sigmoid [35] [75] [27] works with real values and produces output in the range of $0-1$. It is a non linear function that is continuously differentiable but suffers from the vanishing gradient problem. It is expressed as:

$$S(z) = \frac{1}{1 + e^{-}z} \tag{15}$$

3.2.6.5 A compact visualization of activation functions

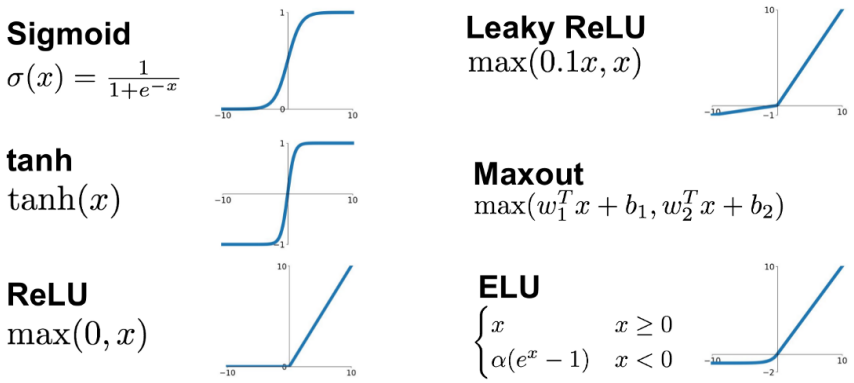


Figure 25: Activation Functions visualization [35]

3.2.7 Loss Functions

A loss function is responsible for evaluating the performance of a model dedicated to learn a given problem. In simple mathematical terms, it is the difference between the expected output to the predicted outcome. The commonly used loss functions are *Mean Squared Error, Cross Entropy, Mean Absolute Error, Hinge, Kullback-Leibler, Huber* etc. For our purpose we use the Cross Entropy Loss which is one of the commonly used evaluation approach for a classification task. This is explained in the section 3.2.7.4. Some of these loss functions are explained below.

3.2.7.1 MSE

Mean Squared Error (MSE) or L2 loss is the average of the squared difference between the ground truth and prediction. Citing from [84] it is expressed as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i^{'})^2 \tag{16}$$

Y_i is the ground truth and $Y_i^{'}$ is the prediction, n is the number of samples

3.2.7.2 MAE

Mean Absolute Error (MAE) or L1 loss is the average of the absolute difference between ground truth and prediction. Citing from [83], it is expressed as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - Y_i^{'}| \tag{17}$$

Y_i is the ground truth and $Y_i^{'}$ is the prediction, n is the number of samples

3.2.7.3 Hinge Loss

This loss function is used for classifiers like *Support Vector Machines* introduced by [62] for binary classification. Citing from [80] hinge prediction loss is expressed as:

$$l(y) = \max(0, 1 - t.y) \qquad t = \pm 1 \tag{18}$$

y is the raw output from the classifier, t and y carry the same sign if classifier predicts the right class making the loss 0.

3.2.7.4 Cross Entropy Softmax

In our work of semantic segmentation the loss function we used to evaluate our model is *Cross Entropy Softmax* function. The softmax outputs a probability distribution. It acts as a soft version of *MAX* function [15]. For an input vector "v" of real numbers it outputs a vector of same dimension having values within $0-1$ where each value in the range represents the probability of distribution of the corresponding input elements. For a class 'c' where $c = 1...N$ of the vector 'v' The probability of class 'c' is:

$$p_c = \frac{e^{vc}}{\sum_{n=1}^{N} e^{v_n}} \tag{19}$$

The cross entropy between a true distribution and a predicted distribution is given as:

$$H(y, p) = -\sum_{i=1}^{n} y_i \log(p_i) \tag{20}$$

y_i represents element in the vector y and p_i element in the vector p. Cross entropy can be used when the previous layer outputs a probability distribution. It is an alternative to squared error.

3.3 Convolutional Neural Networks

Convolution Neural Networks are deep neural networks that share common architectural properties with slight modifications. These are also feed forward networks with learnable weights and biases and activation function. The input layer feeds on information that are passed on to the hidden layers for computation and finally producing the output from output layer. ConvNets treat input

data as images. This facilitates specific architectural modifications that can reduce computational resources [37].

The neurons in a regular Neural Net does not share any connection with others in the same layer. They do are connected to other neurons from previous hidden layer but within a single layer they act independently. For ConvNets however the neurons in the hidden layers are fully connected to each other. Neural Nets also does not scale well to bigger resolution images in contrast to ConvNets. The input data in a ConvNet are arranged in three dimensional order as *Height X Width X Depth*.

An example visualization of a ConvNet architecture is shown below:

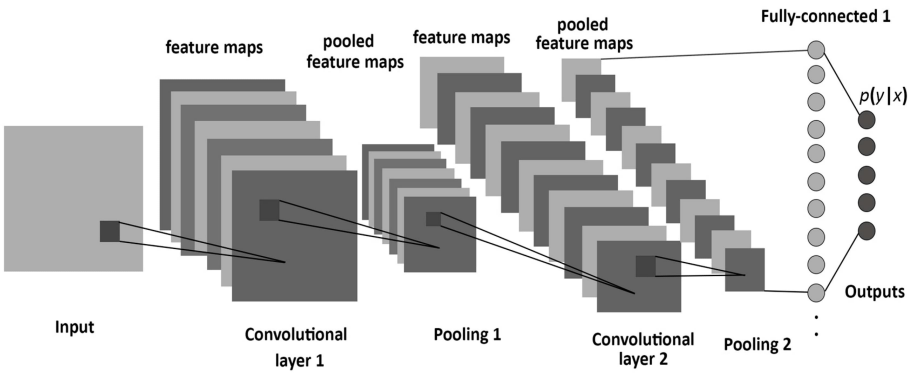


Figure 26: Architectural design of a CNN [56]

3.3.1 ConvNet Building Blocks

A standard ConvNet uses three important layers as part of the transformation process, *Convolution Layer*, *Pooling Layer* and *Activation layer*. Here we describe some more layers as these were a part of our network architecture. These are *Batch Normalization Layer*, *Scaling Layer*, *Eltwise Layer* , *Concatenation Layer*, *De-convolution Layer* and *Softmax Layer*. In short the whole architecture can be explained as follows:

INPUT: The network accepts an input of raw pixels of shape H*W*D
CONV: Matrix multiplication operation
POOL: Down scaling operation
RELU: Activation operation
BATCH NORM: Normalize a mini batch
SCALE: Scale output from previous layer within a range
ELTWISE: Element wise summation

CONCAT: Concatenation operation
DE-CONV: Upscaling operation though matrix multiplication
SOFTMAX: Cross entropy softmax as loss function for evaluation

3.3.1.1 Convolution

The convolution layer is the first layer in the network computation part of the architecture. The convolution layer consists of *Filters*. They are of certain size of shape $h * w * d$ and d the depth of the filter should be same as that of image depth. The filter is composed of numbers known as weights. Convolution as matrix multiplication slides the filter through the entire image performing a dot product. Every region that the filter is placed is known as the *receptive field* and the multiplications are summed up to get one number from each region. Finally convolution produces an *activation map* that holds information from every region in the image. Depending on the filter the activation map contains the kind of features which are represented by numbers in the map. Higher numbers represent presence of certain traits and lower numbers indicate absence [18].

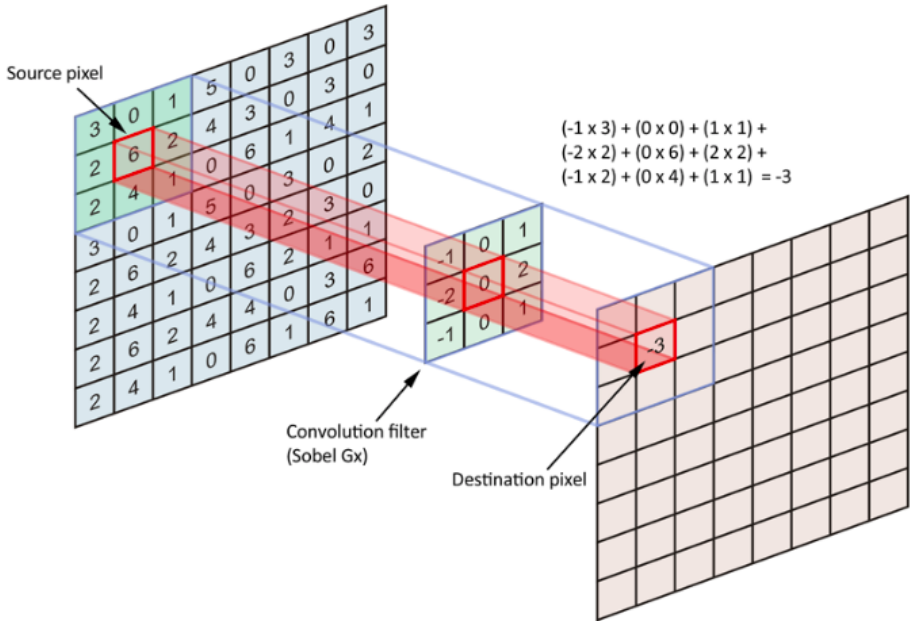


Figure 27: Convolution underlying process [18]

The convolution layer is responsible for detecting features, edges, colors, curves etc. The kernels are feature identifiers [19] that responds to the presence

of certain features. An example of detecting a curve, the filter will have pixel values shaped like a curve with higher numbers and rest of the map will have lower pixel values.

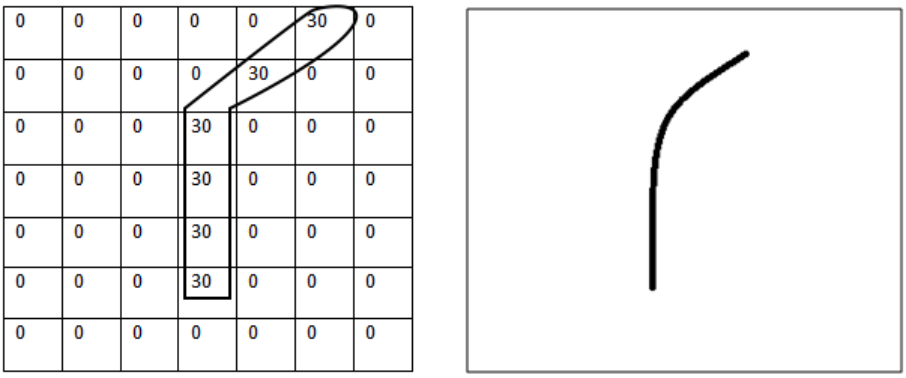


Figure 28: Pixel values(left) and Visual (right) representation of a filter [19]

Summarizing it mathematically [37]:

1. Accepts input of *height* X *width* X *channel/depth*
2. Hyper-parameters required are:

   (a) Number of filters (K)
   (b) Filter dimension (F)
   (c) Stride (S)
   (d) Padding (P)

3. The output volume has the following size:

$$W_{out} = \frac{W_{in} - F + 2P}{S} + 1 \tag{21}$$

$$H_{out} = \frac{H_{in} - F + 2P}{S} + 1 \tag{22}$$

$$D_{out} = K \tag{23}$$

4. Number of parameters:

$$parameters = F_{height} * F_{width} * D_{in} * K \tag{24}$$

3.3.1.2 Grouped Convolutions

Group convolution [38] [41] was first mentioned in AlexNet to split a network in two GPUs. A group convolution operates by dividing the filters into multiple groups where each group focuses on learning different feature representations. This also reduces computational complexity.

The simplest way to understand group convolution [23] is consider a model that has 4 input channels and 32 output channels. And we have two groups, each with 2 input channels and 16 output channels. The computation cost with a single group with filter size $height * width$ would be $height * width * 4 * 32$ but with two groups, we have $(height * width * 2 * 16) * 2$, which is half the computational cost.

In the figure below we illustrate a standard convolution with a single group (left), a group convolution (GConv) with efficient correlation between the input and output channels as GConv2 takes input from different groups in GConv1 (center). And finally, a group convolution when the channels are shuffled (right). Channel shuffling is randomly mixing the output channels of the group convolution.

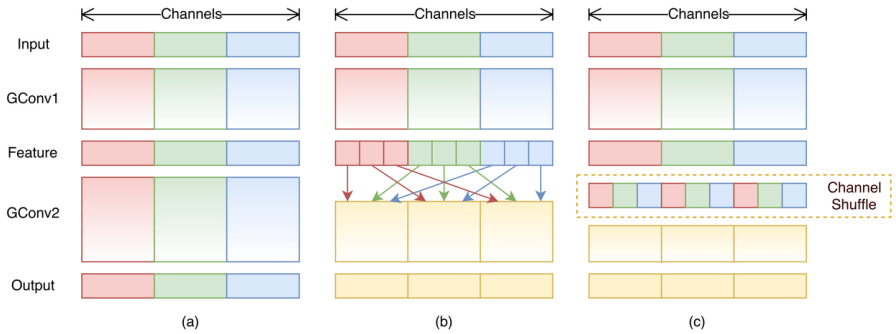


Figure 29: GConv stands for Group convolution. (a) Two stacked convolution layers with same number of groups (b) Input and output channels are fully related as GConv2 takes data from different groups (c) (b) implemented by shuffling the channels [38]

3.3.1.3 Pooling

It is common to insert pooling layers in between convolution layers to reduce spatial dimension of data and enhance computation efficiency [37]. This in turn reduces training time and fights overfitting. *Average Pooling*, *Max Pooling* are some of the common pooling techniques used in current architectures. The Pooling used in our network is Max Pooling.

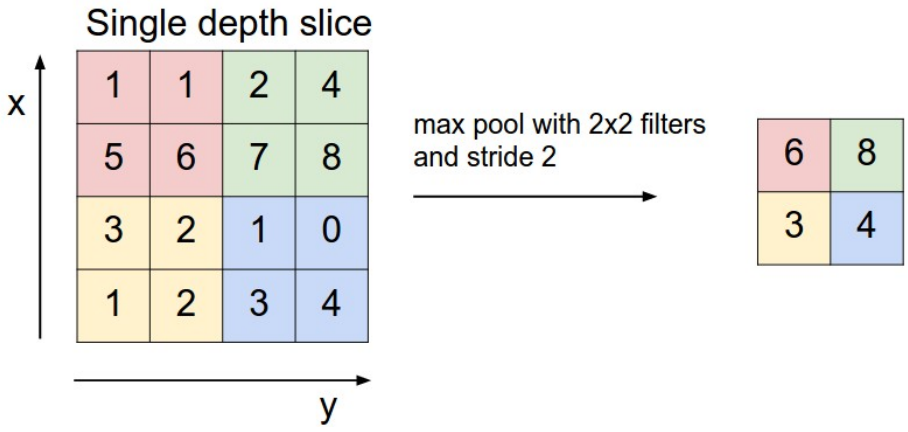


Figure 30: Max Pool downsampling operation [37]

The *max Pool* functions by taking the max in each region cutting the activation by 75%. The pooling above reduces the spatial dimension by half. It does not introduce any parameters. Mathematically it can be expressed as [37]:

1. Accepts input of *height* X *width* X *channel/depth*
2. Hyper-parameters required are:

   (a) Filter dimension (F)
   (b) Stride (S)

3. The output volume has the following size:

$$W_{out} = \frac{W_{in} - F}{S} + 1 \tag{25}$$

$$H_{out} = \frac{H_{in} - F}{S} + 1 \tag{26}$$

$$D_{out} = D_{in} \tag{27}$$

3.3.1.4 Batch Normalization

This layer is used to normalize a layer with zero mean and unit variance. Consider a mini batch $B = x_1...x_n$. Learnable parameters γ and β. Looking at the mathematical representation of Batch norm [17] [33]:

$$\mu_B = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \rightarrow Mean \tag{28}$$

$$\sigma_B^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu_B)^2 \qquad \rightarrow Variance \tag{29}$$

$$x_i^{'} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad \rightarrow Normalization \tag{30}$$

$$BN = y_i = \gamma x_i^{'} + \beta \qquad \rightarrow Scale \ \ and \ \ shift \tag{31}$$

The γ and β are learnable parameters which makes this algorithm very strong. This gives network the flexibility to keep the input as it is or transform it through the series of equations as desired if any other distribution suits better to achieve the desired goal.

3.3.1.5 Transpose Convolution

Correctly knows as *Transposed Convolution*, the deconvolution works by padding each pixel and apply convolution on top of it. A simplistic illustration is shown below:

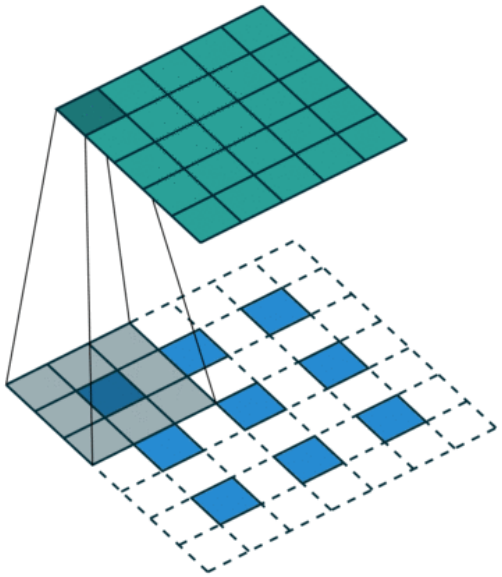


Figure 31: Working principle of Transposed Convolution Layer [16]

4 Approach

The segmentation was performed on street images provided by *Lehmann & Partner*, a medium sized engineering company founded in 1990. We clubbed a ResNet with a UNet to work this out. A ResNet is popularly known as a classification network. To serve our purpose it needs a decoder that can upsample the images back to it's original resolution. This is where a UNet comes into picture. We use the decoder of the UNet to make it a whole segmentation architecture.

4.1 Data Acquisition

The data was provided by *Lehmann & Partner* in form of RGB images and point clouds. Hence RGB acquisition and depth acquisition were handled separately. A laser scanner was mounted on the roof of a vehicle to scan the surrounding where the vehicle was located to gather point cloud data and four different cameras pointing to four different directions were used to gather images. These data were acquired from *Berlin*. Two of the cameras *CAM1* and *CAM2* were positioned for front view and rear view respectively and other two *CAM3* and *CAM4* for left and right view respectively. The images were taken at $2452X1840$ resolution. *AVT Stingray* cameras were used for the RGB images and *Fraunhofer IPM CPS* [34] scanner known as CLEARANCE PROFILE SCANNER was used for the point clouds. To illustrate the camera angles with their respective images, a visualization is showcased below.

(a) Camera1 (Front view)

(b) Camera2 (Rear view)

(c) Camera3 (Left view)

(d) Camera4 (Right view)

Figure 32: A visualization of images from four different angles

4.1.1 Depth map generation

The collection of depth information was done using a *Mobile Mapping Vehicle* with a laser scanner mounted on the top. The sensors were calibrated to a reference point in the vehicle. This includes extrinsic calibration that maps the camera coordinate system to the world coordinate system and intrinsic calibration that transforms these camera coordinates to 2D image coordinates by computing the distortion coefficients namely *Radial distortion* and *Tangential distortion* to correct the lens distortion. As the vehicle moves, the trajectory is computed using GPS as well as IMU (Inertial Measurement Unit [81]). These sensors are used in combination of GPS information to determine accurate localization of the vehicle with the vehicle's pose (position and orientation). These are in time sync with the scanner and camera information to enable projecting coordinates to world coordinate system. The depth images were generated by processing the point clouds collected by the scanner.

The point clouds are represented using *voxel grids*. A voxel represents value in a regular 3D grid as pixels in bitmap. In the figure 33 below we visualize the vehicle and scanner used for data extraction.

Lehmann & Partner encodes the unstructured point cloud as a collection of voxels. Each of these voxels are of 10 cm in length, breadth and height. The grids can hold minimum of zero to any number of values. Generally the grids at the top part of the point cloud were empty representing infinite distance (sky). The maximum distance allowed for depth processing is 1 km. To create a depth image out of this, an empty artificial image of the same resolution as that of the RGB images was placed in front of the camera.

(a) Data acquisition vehicle (b) CPS Laser scanner

Figure 33: Data acquisition using CPS scanner mounted on the top back of the vehicle with cameras for four different angles [34]

A ray is shot from the camera view point passing through the empty frame to the point cloud encoded as voxel grids. When it hits a voxel, it checks for values inside. Considering, if there is a single point in the voxel, the distance is measured in a straight forward manner. For more than one point, the point closest to the camera in the particular voxel is considered for depth value estimation. A mathematical and schematic representation is illustrated below:

```
fun get_depth(dist):

    if(dist<threshold):
        return (threshold-dist)/threshold
    else:
        return 0.0
```

A threshold of 1 km is initialized which is used to check the measured distance. If the objects are closer to the view point, they get a value nearly equal to 1. Objects that are half way get values nearly equal to 0.5. For objects which are beyond 1 km are initialized 0. The empty floating point image is initialized with the corresponding distance as pixel value in the (X,Y) location where the ray intersects. This is illustrated in the figure 34 below.

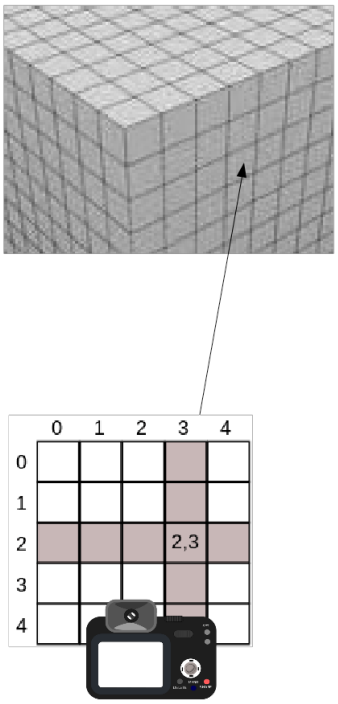


Figure 34: Depth map creation using ray projection and tracing

4.1.2 Data Pre-processing

4.1.2.1 Remove empty depth maps

Some of the depth images did not contain any value and were literally black. These were the corrupted ones which were removed along with their corresponding RGB and labelled data for further network processing. A valid depth image is shown below:

Figure 35: Valid depth image

Apart from this we also remove the first and the last 5 data samples containing RGB,mask and depth from each set of images taken by individual cameras. This is done because there are areas in the beginning and in the end of the vehicle trajectory where the laser scanner does not operate but the camera still records information. For these regions, we will have images but no cloud data to process and acquire corresponding depth images.

4.1.2.2 Scale the depth images for visualization

The distance information represented by the depth maps does not exceed 100 metres. Hence the maps contain pixels whose values differ by few decimal points. This makes it hard to read and analyze the maps. Hence, we scaled the maps by a scaling factor. For a depth image, we determine the minimum pixel value that is greater than 0.01 and we subtract this from 1.0. We divide 255 by this subtracted value which is used as a scaling factor to multiply the depth map. Values less than 0 are assigned 0 value.

4.1.2.3 Remove data without depth images

A bunch of RGB image and label data comes with no corresponding depth maps. These images are sorted out from entire set and discarded.

4.1.2.4 Resize the data

An option to work with the original image resolution to the preferred network architecture is we can crop small patches from the data and send these patches as input to the network train. But training on full image resolution (entire image) and maintaining the image cohesion has proven better for our dataset. Hence we resize the images to $512X512$ as this was the maximum workable size with the preferred architecture. For resizing, inter-nearest interpolation was used.

4.1.2.5 List of labels in our dataset

The original labelled images contained 35 different labels. These are:

Class value & definition			
Class Values	**Class Definition**	**Class Values**	**Class Definition**
Class 00	Unclassified	*Class 18*	Encasement of equipment
Class 01	Concrete	*Class 19*	Mail box
Class 02	Asphalt	*Class 20*	Advertisement panel
Class 03	Concrete pavement	*Class 21*	Radar trap
Class 04	Mosaic pavement	*Class 22*	Rail tracks
Class 05	Natural stone pavement	*Class 23*	Building
Class 06	Loose natural surface	*Class 24*	Garage
Class 07	Loose stony surface	*Class 25*	Stairs
Class 08	Gutter, curb stone	*Class 26*	Enclosure
Class 09	Hole cover	*Class 27*	Tunnel
Class 10	Tree trunk	*Class 28*	Wooden pole
Class 11	Broadleaf crown	*Class 29*	Metal pole
Class 12	Conifer crown	*Class 30*	Sky
Class 13	Bush	*Class 31*	Vehicle
Class 14	Street sign	*Class 32*	People
Class 15	Streetlight	*Class 33*	Undistinguished
Class 16	Traffic light	*Class 34*	Background
Class 17	Guardrail	*Class 35*	Bollard

Table 1: Class values as pixel values in a mask with their corresponding description

For simplicity some of these labels were treated as a single label and some which did not provide any relative importance to the goal realization or were present in extremely negligent proportions were ignored. For example, the pavements namely *Concrete pavement, Mosaic pavement, Natural stone pavement, Loose natural surface and Gutter, curb stone* are treated as a single label as they refer to the common concept of pavement. Similarly, *Broadleaf crown and Conifer crown* were treated as a common label as they refer to the idea of trees.

Some of the labels that occurred at a very low frequency are *Concrete, Loose stony surface, Mail box, Radar trap, Rail tracks, Garage, Tunnel, Wooden pole* etc. Along with these label, the *background and undistinguished* were also ignored. Finally we were left with $20, (0 - 19)$ labels that we used as number of classes for segmentation. The final list of classes is illustrated in the table below.

Class value & definition			
Class Values	**Class Definition**	**Class Values**	**Class Definition**
Class 00	Asphalt	*Class 10*	Encasement of equipment
Class 01	Pavement	*Class 11*	Advertising panel
Class 02	Holecover	*Class 12*	Building
Class 03	Tree trunk	*Class 13*	Enclosure
Class 04	Trees	*Class 14*	Metal pole
Class 05	Bush	*Class 15*	Sky
Class 06	Street sign	*Class 16*	Vehicle
Class 07	Street light	*Class 17*	People
Class 08	Traffic light	*Class 18*	Bollard
Class 09	Guardrail	*Class 19*	Undefined

Table 2: Final class list with class value and corresponding description

4.1.3 Dataset preparation

The original dataset from Berlin consists of 23715 data samples which can be categorized into 7905 of RGB, mask and depth images. These are street scenes or outdoor scenes. They are divided in four groups of camera angles namely Camera1 (front view), Camera2 (rear view), Camera3 (left view) and Camera4 (right view). They are further pre-processed and is divided into 3 splits of *Train, Validation and Test* set. The train set consists of 4894 data samples, validation consists of 669 data samples and test set has 1353. Each of these splits contain equal number of samples from all four cameras. The images from each camera group are picked at random for each split. The train, test and validation splits

are roughly in the ratio of $70:10:20$.

1. Train: 4894
2. Val: 669
3. Test: 1353

4.1.3.1 Data Augmentation

A train set of not more than 5000 images does not suffice for a network with around $50M$ parameters. Hence data augmentation was considered a part of pre-processing step to enlarge the input set. It includes translation, rotation, horizontal and vertical flips and color augmentation. The train set after augmentation was increased to 68516 data samples. We used 15, 30 and -15 degrees of rotation, for translation, X and Y scales (translation factors) were, $(x = 0, y = 120)$, $(x = -150, y = 0)$, $(x = -100, y = 150)$, $(x = 150, y = 150)$, $(x = 0, y = -120)$, $(x = -150, y = -150)$, $(x = 150, y = 0)$ and $(x = 150, y = -150)$. Color augmentation includes brightness change with a floating point value that ranges from $0.3 - 0.9$.

4.1.3.2 Inconsistent Labelling

There are certain images where the corresponding mask contains poor labelling of the classes. This in turn also affected the quantitative results from the network as the ground truth itself is incorrectly labelled. These inconsistencies occur approximately in 15% of train data and around 10% of validation and test data. For instance, the bollards in some of the images or the traffic lights or the road etc. A couple of examples are illustrated below:

(a) RGB image (b) Colored mask

Figure 36: In the mask (right), the bounding box in red shows incorrect labelling of the class bollards and the green shows the correct way of labelling. This can be cross verified using the original image (left) to see how the class originally features.

In the figure below, we show a couple of images where in the second row the class *street light* in the mask highlighted in red is labelled incorrectly as class *street sign*. The correct way of labelling a *street light* is illustrated in the first row in the mask highlighted in yellow.

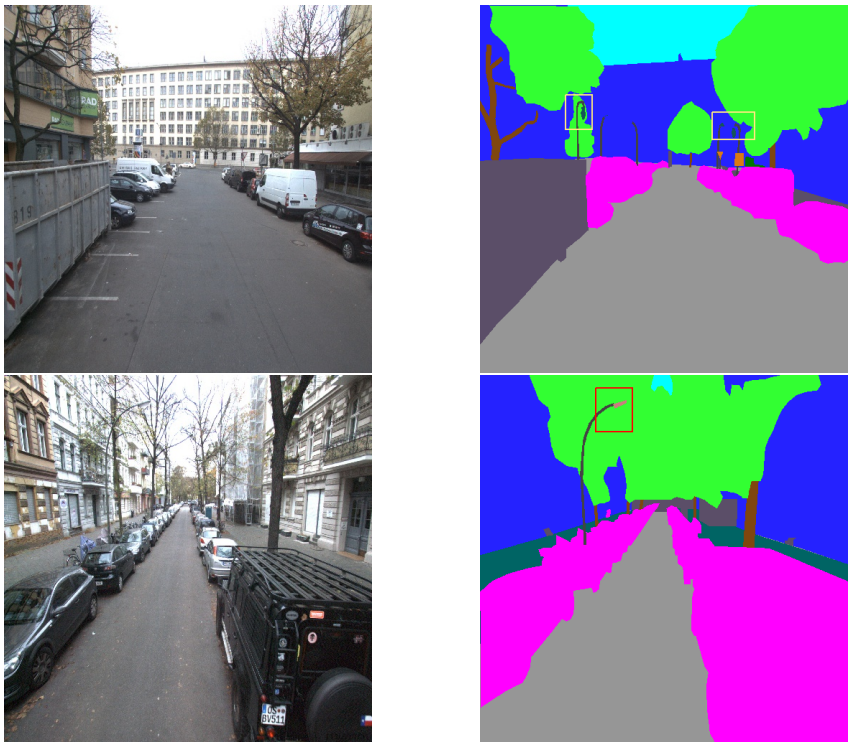


Figure 37: RGB Image (left) and corresponding false colored ground truth masked (right)

4.2 Architecture Model

4.2.1 ResNets vs DenseNets

Residual networks vary from Dense Networks in terms of the configuration of the skip connections. The inter layer connections in ResNet are structured in a manner that any 'kth' layer receives input only from the second or third previous layer. But in a Densenet, a 'kth' layer receives information from all the previous layers above that is any layer receives the feature maps from all the layers above and the it's output is send to all the subsequent layers below [26]. In a ResNet, the inputs are all added with the same depth, hence the depth does not undergo any feature changes whereas Densenets hosts concatenation of information along the depth dimension. ResNets encourage learning the iden-

tity (residual) function but densenets add identity function where output of the identity function is added to the next layer [25]. To review the equations again in short:

$$Y_k = F(X_k) + X_{k-2} \qquad \rightarrow ResNet \tag{32}$$

$$Y_k = F(X_k, X_{k-1}, X_{k-2}...Xk - n) \qquad \rightarrow densenet \tag{33}$$

Densenets are very efficient networks in terms of parameter usage. However they are memory hungry networks [71]. A wide ResNet with ten times more parameters than a DenseNet consumes almost the same memory. The training time is also severely affected when working with a Densenet. This might attribute to the fact that a densenet structure is built analogous to a ResNet structure; "thin and deep". According to [32], a thin and deep fashioned network proved parameter efficient compared to their shallow and wide counterparts. Another attribute reasoning to demand excessive memory are connections. More the connections more the memory requirement, and a Densenet hosts more of these than it's ResNet variant. Considering all these factors into account we decided to use a ResNet as our primary model (encoder). A visual structural difference is shown below:

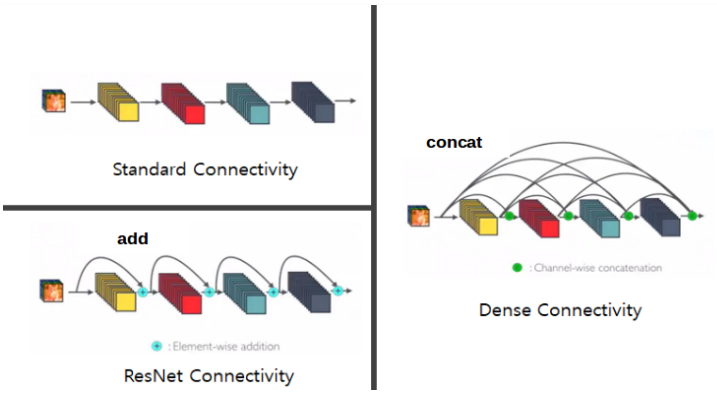


Figure 38: Structural difference between regular ConvNet (top left), ResNet (bottom left) and denseNet (right)

4.2.2 ResNet-34

Having said that the encoder is a ResNet, we decided to use a *ResNet-34*. The ResNet-34 comprises of 33 convolution layers from where the residual blocks consist of 32 convolution layers, considering the fact that these are basic blocks. All convolution layers are followed by Batch Norm and Relu activation. The model starts off with a $7X7$ convolution layer with a stride of 2, with 64 output channels immediately followed by $3X3$ Max Pool with a stride of 2. A bigger kernel size provides a larger field of view. Starting with the residual blocks, a

ResNet-34 comprises of 4 residual branches. Each residual block inside these branches is comprised of two convolution layers of kernel size $3X3$. The first branch encapsulates 3 residual blocks, the second has 4, the third has 6 and the last one has 3. The number of channels in the first branch through out the residual blocks of two convolution layer each is $(64, 64)$, in the second $(128, 128)$, the third has $(256, 256)$ and the fourth has $(512, 512)$. The network then performs an Average pooling before evaluating. An illustration of a residual block in terms of it's layers is shown below.

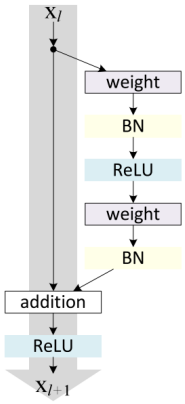


Figure 39: Layer structure in a residual block from [25]. Weight here can be interpreted as Convolution layer, BN is Batch Norm

To build our ResNet model we kept the original residual block architecture, though couple of more it's variants are available with minor modifications in it's layers. Some of these are illustrated below:

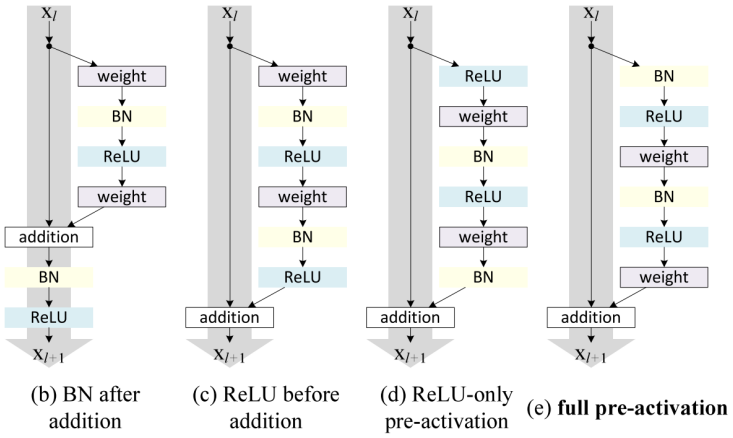


Figure 40: Variations of residual blocks from [25]

Here, an advantage of pre-activation according to [30] is gradients can flow through the skip connections directly without any hindrance to any other previous layers. These changes were proposed by [29] to refine the residual blocks as going more than 1000 layer deep with the original block resulted in performance degradation.

4.2.3 Architectural modifications

For our purpose we made some modifications in the encoder before fusing the decoder. The basic building blocks were kept intact. The first $7X7$ convolution was replaced by three $3X3$ convolution layers. Each of these $3X3$ convolution had 64 output channels as that of the original convolution layer, but these are non-strided convolutions as they do not downsample the images to half of it's original resolution, preventing information loss. An advantage of using smaller kernel size is computational efficiency as the original layer required 9408 parameters whereas in our case, only 5184 parameters were used.

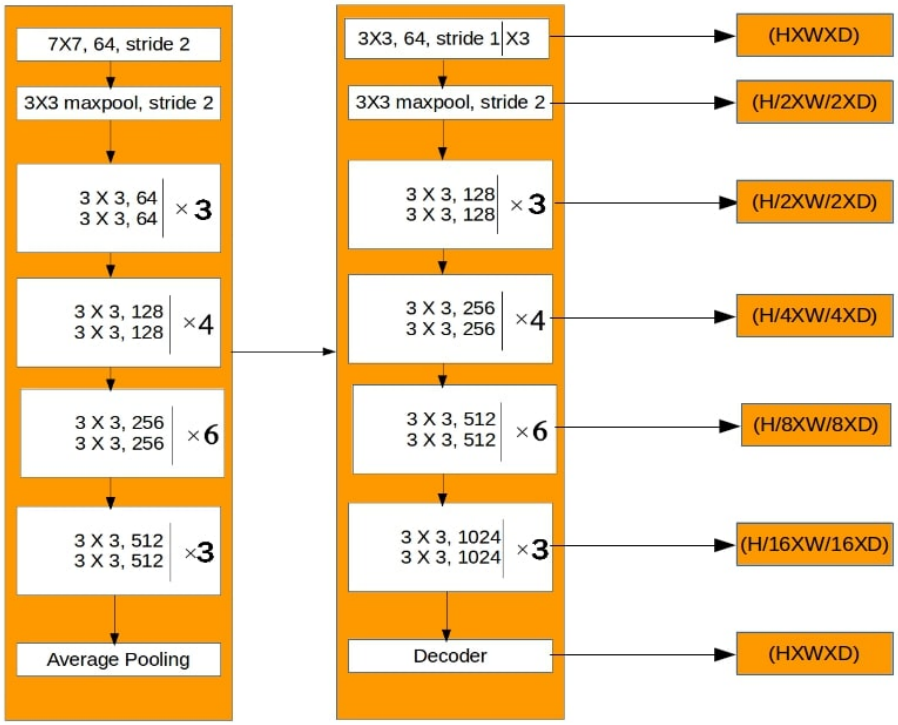


Figure 41: Original ResNet-34 architecture (left) to our modified version (right), the right most orange boxes represent the output image resolution from every branch of the network. $(3X3)$ are kernel sizes.

According to [92], a wide variant of a ResNet is more efficient than it's deeper counterpart, as training deep networks has a problem of diminishing feature reuse. A small percentage of improved performance demands almost doubling the number of layers. Hence we decided to stick to a shallow ResNet (we moved from ResNet-50 to ResNet-34) and make it more wider than it's original counterpart. For this, we doubled the number of channels in the residual branches from $(64, 64) \rightarrow (128, 128)$, $(128, 128) \rightarrow (256, 256)$, $(256, 256) \rightarrow (512, 512)$, $(512, 512) \rightarrow (1024, 1024)$. We also remove the Average pooling layer in the end as that would be replaced by a complete decoder. Hence our encoder downsamples the image only 4 times compared to the original version of ResNet-34. To illustrate this in a schematic manner, we present a comparative study of both the architectures in the figure 41 above.

4.2.4 Fusing UNet Decoder

The decoder is used here to upscale the images to it's original resolution. In some of the work by [12] and [91], The upsampling is performed using straight forward bi-linear interpolation. The problem with this approach is that the step is not learnable. Same can be witnessed with the SegNet [5] that uses *Unpooling layers* for upsampling. Hence, there is every possibility that the performance of the network could have been improved if we can replace this step with a much smarter approach. The downfall to this is that the overall resources for the network in terms of memory or parameters increases along with train time.

Nevertheless, we decided to go for a proper decoder as we were more focused on achieving better quality segmentation. A very simple and effective decoder that could have done the job is from the UNet [51]. Multiple learnable upsampling layers and concatenation of higher resolution feature maps rather than summation like in FCN8 [45] makes the UNet decoder robust to scale variations. The decoder consists of a *deconvolution layer* of kernel size $2X2$ and stride of 2 followed by two consecutive non strided *convolution layers* of $3X3$ kernel size. We have four branches of these trio with decreasing number of output channels as resolution increases ($512 \rightarrow 256 \rightarrow 128 \rightarrow 64$). Every deconvolution and convolution is followed by Batch Norm and Relu activation. A concatenation is performed after every deconvolution from the corresponding resolution branch in the encoder. This can help better localize and represent features. A schematic representation is illustrated below.

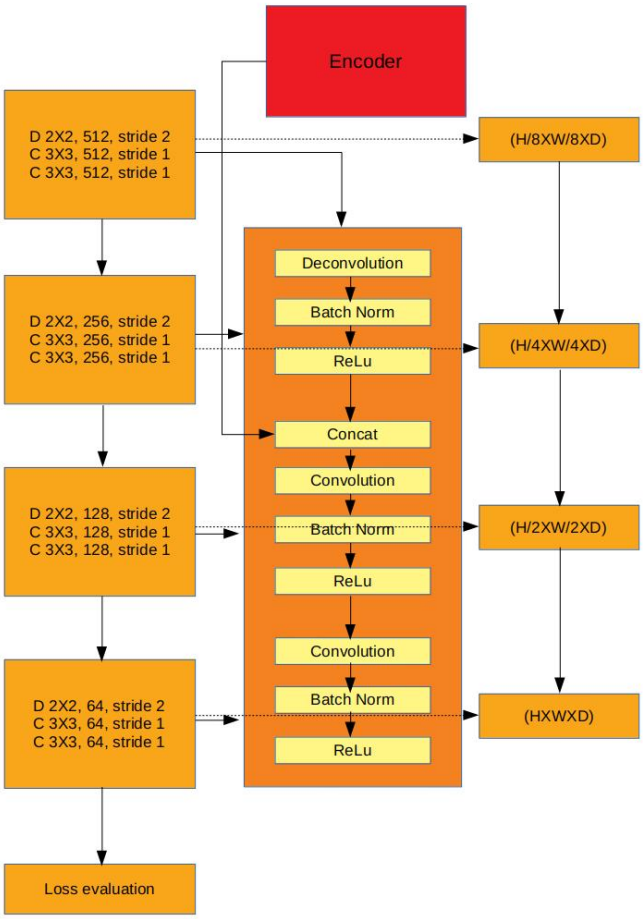


Figure 42: The orange boxes on the left shows the basic architecture flow of convolution (C) and decovolution (D) layers in the decoder branches, in the center the dark orange box shows the underlying flow of individual layers in a single decoder branch and the right orange boxes shows the corresponding resolution of every branch. $(2X2)$ or $(3X3)$ are kernel sizes.

4.2.5 Final architecture

Finally we club the modified ResNet-34 encoder to the UNet decoder to make a final network architecture that can be effectively used for segmentation of images. A visual representation of the architecture is shown below.

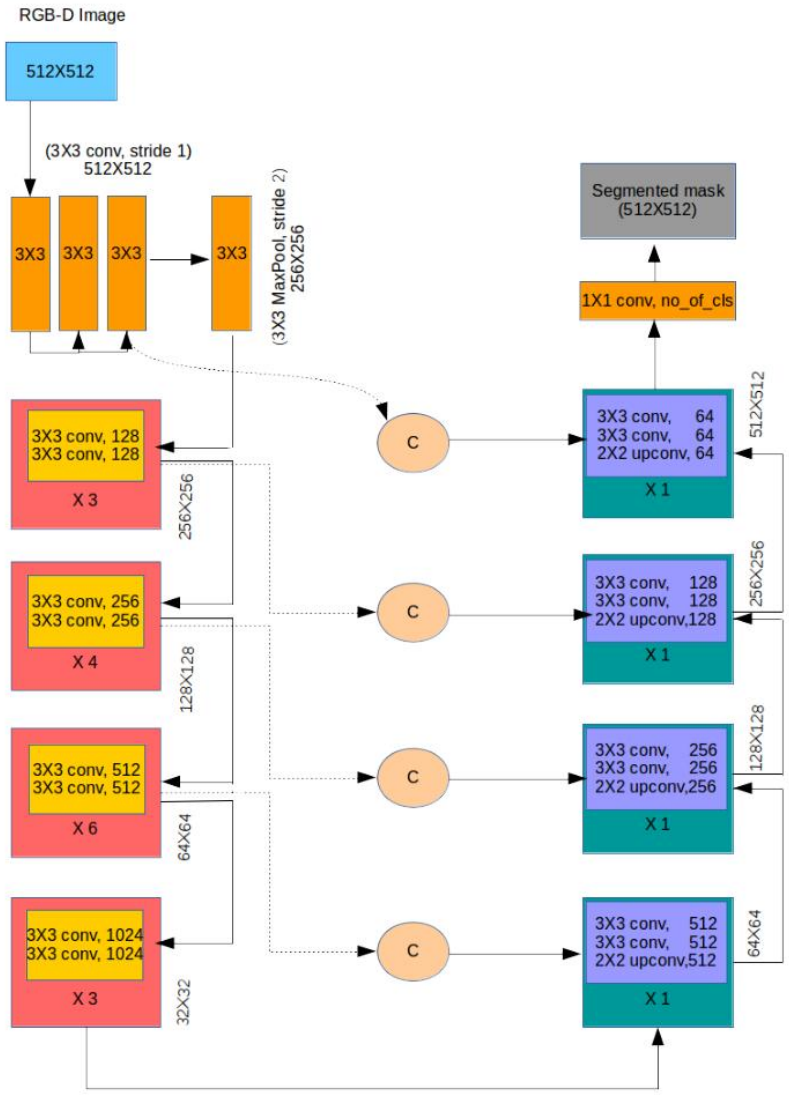


Figure 43: Each convolution and up convolution is followed by Batch Normalization and ReLu activation. 'C' stands for concatenation, feeding information from corresponding resolution level in the encoder. $(2X2)$ or $(3X3)$ are kernel sizes.

5 Experiments

5.1 Training

The experiments were performed on 4894 training images which after augmentation resulted in 68516 training samples. For validation and testing we have 669 validation images and 1353 test images. We started with three basic experiments to validate the ResNet model that can be used for the test set. They comprised of three models, ResNet-34, ResNet-50 and ResNet-101. The experiments were evaluated using *Softmax Cross Entropy loss*. We took the model with the highest performance and tried out parameter variations to come up with a better version of it. Finally we used the best model to evaluate on the test data and bring out the comparison between segmentation performance when using depth information and without. We then tried out another approach to feed the depth information as described in [28] in contrast to the conventional way of stacking the depth channel along with RGB and compare the possible improvements and deterioration in the segmentation performance. Finally we compared our best version of performance to a state-of-the-art approach from [48]. The experiments were conducted using the *Caffe* framework and later transferred to *Tensorflow*.

The initial training learning rate was 0.0001 with a weight decay of 0.0005 and batch size of 1. The optimizer used is *Adam* and momentum (1 and 2) were 0.9 and 0.999 respectively. The learning policy was *fixed* and the step size for saving a model was 5000. The train data was augmented and randomly shuffled for robustness. The models were normally trained to a max of around $10 - 12$ epochs until convergence.

5.2 Testing

For testing, we first validated the models using cross entropy loss function on the validation set of 669 data samples and the model with the lowest loss is used to evaluate on the test set of 1353 data samples. Quantitative evaluation was performed using two standard metrics, accuracy and IoU.

5.2.1 Quantitative Metrics

We used *Global Accuracy*, *Mean Accuracy* and *Mean Intersection Over Union* as three standard metrics to evaluate the performance of our model quantitatively. These metrics are inspired from the paper [28].

5.2.1.1 Global Accuracy

The global accuracy (Global) for a mask having K number of classes is the percentage of correctly classified pixels (TP) to that of total number of annotated

pixels (N) in that mask. The global accuracy is computed per image and averaged over the entire test set.

$$Global = \frac{1}{N}\sum_{c} TP_c, \qquad c \in 1...K \tag{34}$$

5.2.1.2 Mean Accuracy

The mean accuracy (Mean) for a test set is the average of per class accuracy. We compute the independent class accuracies on the test set and take an average to come up with the mean.

$$Mean = \frac{1}{K}\sum_{c} \frac{TP_c}{TP_c + FP_c}, \qquad c \in 1...K \tag{35}$$

5.2.1.3 IoU

The mean IoU for a mask is the mean of intersection over union of the predicted mask to the ground truth. We compute the IoU for independent classes on the test and take the mean to produce the final IoU.

$$IoU = \frac{1}{K}\sum_{c} \frac{TP_c}{(TP_c + FP_c + FN_c)}, \qquad c \in 1...K \tag{36}$$

- **TP** : True positive, when predicted mask is 1 (positive) when ground truth mask is 1
- **FP** : False positive, when predicted mask is 1 (positive) but ground truth is 0 (negative)
- **FN** : False negative, when predicted mask is 0 (negative) but ground truth is 1 (positive)

The quantitative evaluation for all the experiments are done using the above metrics in a similar fashion. The metrics are used to evaluate the overall performance on a dataset and per class performance. To evaluate the performance on a dataset, say for example *accuracy*, we compute accuracy using the *Global Accuracy* metric per image and take the mean to find the global accuracy of the dataset. To compute per class accuracy on the dataset, we maintain a dictionary containing keys as class and values as corresponding accuracies. We take the mean of the values per key to find mean accuracy of individual classes on the dataset and mean of the mean of these individual classes to compute *Mean Accuracy* on the dataset.

5.3 Results

5.3.1 ResNet-34 vs ResNet-50 vs ResNet-101

The comparison among the various ResNet architectures on validation set are plotted in a tabular form with their mean accuracy and mean IOU on the entire set. These experiments were conducted using RGB-D inputs. The ResNet-34 was made wider with $(64, 128, 256, 512, 1024)$ number of feature channels in the encoder and $(512, 256, 128, 64)$ in the decoder compared to ResNet-50 which was relatively thinner and deeper had $(48, (48, 192), (96, 384), (192, 768), (384, 1536))$ channels in the encoder and $(768, 384, 192, 48)$ in the decoder. ResNet-101 was the thinnest and the deepest of all having $(32, (32, 128), (64, 256), (128, 512), (256, 1024))$ channels in the encoder and $(256, 128, 64, 32)$ in the decoder. ResNet-34 has basic residual blocks and ResNet-50 and 101 contains bottleneck residual blocks. Based on these experiments, we select the model that we can use for the test set and also use to compare the model when trained without depth.

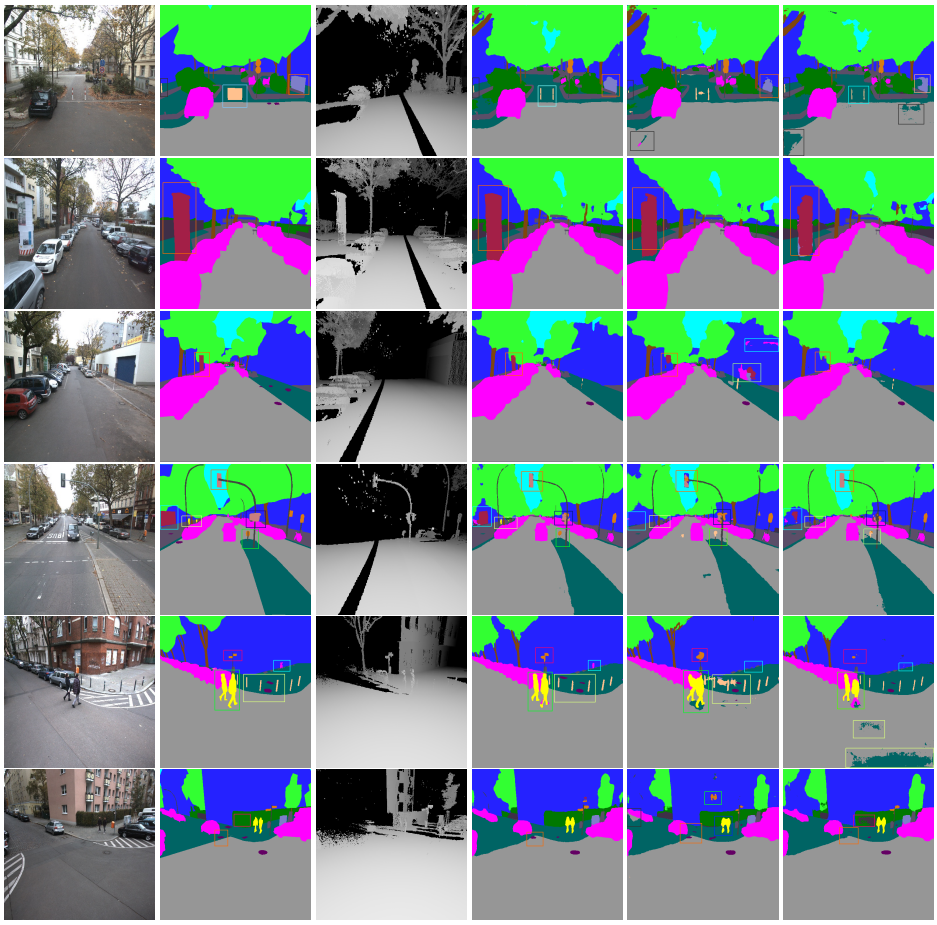
Performance comparison of ResNet architectures			
Input type	**Global Acc**	**Mean Acc**	**Mean IoU**
ResNet-34	**94.83**	**60.59**	**53.17**
ResNet-50	94.50	58.85	51.16
ResNet-101	94.30	54.03	48.13

Table 3: Accuracy and IoU comparison of three basic ResNet architectures on validation set

From the table 3 above we can see that the *ResNet-34* outperformed the rest. Making the network wider could have incorporated enough number channels to represent the important features from both RGB and depth. Making a network wider is more computationally efficient than to have thousands of small kernels as GPU is much more efficient in parallel computations on large tensors [3]. Moreover, according to [92], increasing the depth of the network has a reverse effect in the regularization process than making a network two or three times wider as this might require doubling or tripling the depth of it's deeper counterparts to learn the same representations, making them computationally expensive. The qualitative differences between ResNet-34 and ResNet-50 are a bit hard to distinguish, which is also showcased in their quantitative values as well, yet we decided to stick to ResNet-34 as we achieve almost the same performance using a much smaller network (maximum performance gain consuming less resources).

5.3.2 Visual comparison of the ResNet architectures

The qualitative differences between the different architectures are illustrated in the figure 44 below. The figure can be divided into four categories of perception. The first two rows are dedicated to camera1 (front view), third and fourth rows are for camera2 (rear view), fifth and sixth rows are reserved for camera3 (left view) and last two rows for camera4 (right view). These differences shows that ResNet-34 performs better compared to ResNet 50 & 101 on all the four different camera angles. ResNet-101 as shown produces way more artifacts than rest of the networks. This can be attributed to the fact as stated in [92] that wider and shallow networks performs relatively better than their thin and deep counterparts. Moreover, it is safe to say that for the current dataset, may be a smaller version of ResNet would perform well rather than using a bigger version of it.



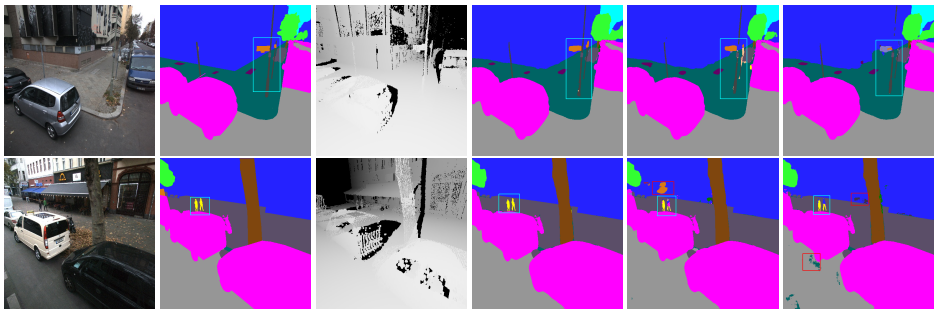


Figure 44: Starting from the left, are the original images and next to it are their corresponding masks, third columns holds the corresponding depth images, fourth column holds masks from ResNet-34 and next to it are from ResNet-50 and extreme right are masks from ResNet-101. These comparisons are on validation set. The first two rows are from camera1, 3rd and 4th from camera2, 5th and 6th are from camera3 and last two are from camera4.

Analyzing closely on the images from the figure above, we can see that some of the challenging classes like *encasement of equipment, traffic lights, people, street signs etc* are better segmented using ResNet-34 and it produces way lesser artifacts than it's deeper counterparts. For easy analysis these are further highlighted using the colored boxes.

5.3.3 Variations of ResNet-34

Before proceeding to evaluate the model on the test set, we decided to try out some variations of ResNet-34 in order to find the best version of parameter settings that can justify our segmentation better. For this, we implemented the model by tweaking the number of channels in the branches resulting in making the network thinner. We also used dilated convolution layers in one of the models to see if wider field of view can provide additional advantage over network depth and width. We evaluated the models on the validation set. In the table 4 below, are the quantitative scores of the variations. We can see that a wider model performed better than the rest on our dataset, with which we proceeded for further image processing. For simplicity let's name the original ResNet-34 to be Res1, doubling the model size to be Res2, reducing the model size to three-fourth of the original size to be Res3 and reducing it to half with dilated convolution layer to be Res4. Res1 has $(64, 128, 256, 512)$ feature channels in the residual branches of the encoder and $(256, 128, 64, 32)$ in the decoder. Res2 has $(128, 256, 512, 1024)$ feature channels in the encoder and $(512, 256, 128, 64)$ in the decoder, Res3 contains $(48, 96, 192, 384)$ feature channels in the encoder and $(192, 96, 48, 24)$ in the decoder and finally Res4 has $(32, 64, 128, 256)$ channels in the encoder and $(128, 64, 32, 16)$ feature channels in the decoder. We used a dilation factor of 4 and 8 in the third and fourth residual branches of the encoder in Res4 respectively.

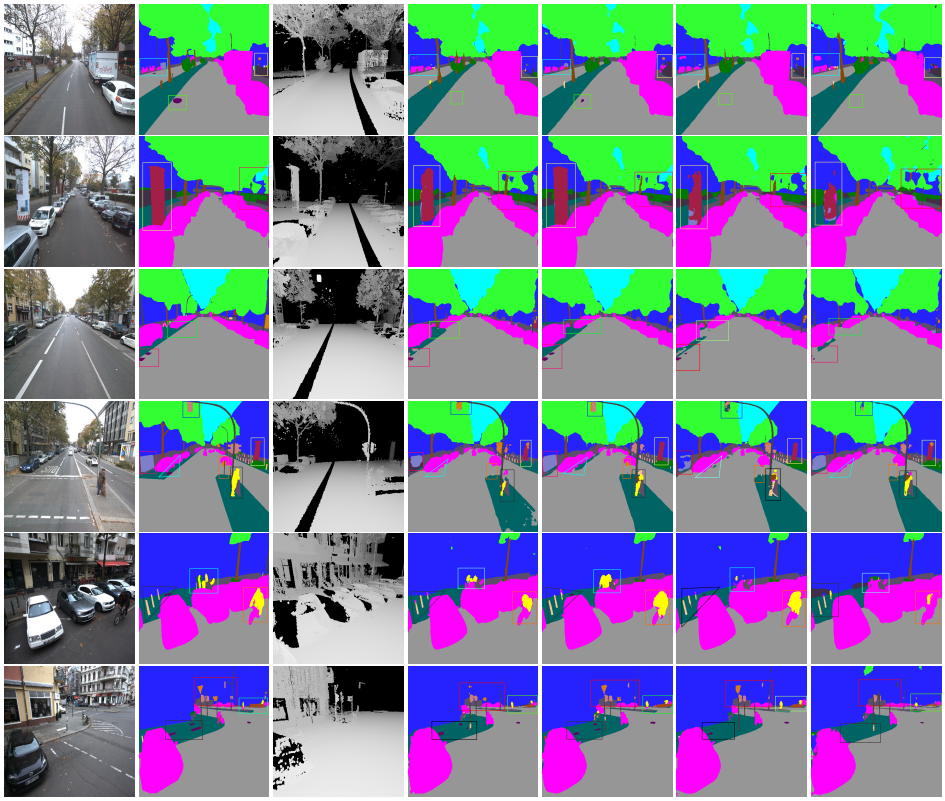
Performance comparison of ResNet-34 variations			
Input type	**Global Acc**	**Mean Acc**	**Mean IoU**
Res1	94.68	58.35	51.08
Res2	**94.83**	**60.59**	**53.17**
Res3	94.54	57.27	50.13
Res4	94.42	55.49	48.31

Table 4: Evaluation of ResNet-34 variations on validation set

From the table above we can see clearly that reducing the size of the network resulted in poorer performance. Usage of dilated convolutions also did not bring any advantage to the segmentation quality. Instead when we doubled the size of the original network, there was a marginal increase in the model performance.

5.3.4 Visual comparison of the variations of ResNet-34 architecture

The figure below illustrates the qualitative differences between the different architectures of the model.



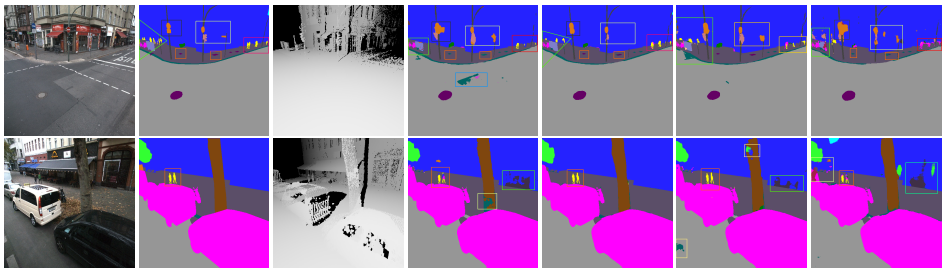


Figure 45: Qualitative visualization of the variations of ResNet-34 on validation set. The first columns contain the images, the second columns are the original masks, the third holds the depth images, fourth column contain prediction masks from Res1, the fifth column holds prediction masks from Res2, the sixth column refers to Res3 and the last column has prediction masks from Res4.

From the figure above, we can see that ResNet-34 model by doubling the size has a slight qualitative advantage over the original one. This can also be justified by their respective scores as mentioned in table 4. But as we decrease the model size, there is a significant drop in performance quality. The qualitative differences are highlighted using colored boxes. The last two variations of the ResNet-34 produces quite a lot artifacts. Keeping this in mind, we decided to proceed with Res2 variant.

5.3.5 ResNet-34 on Testset

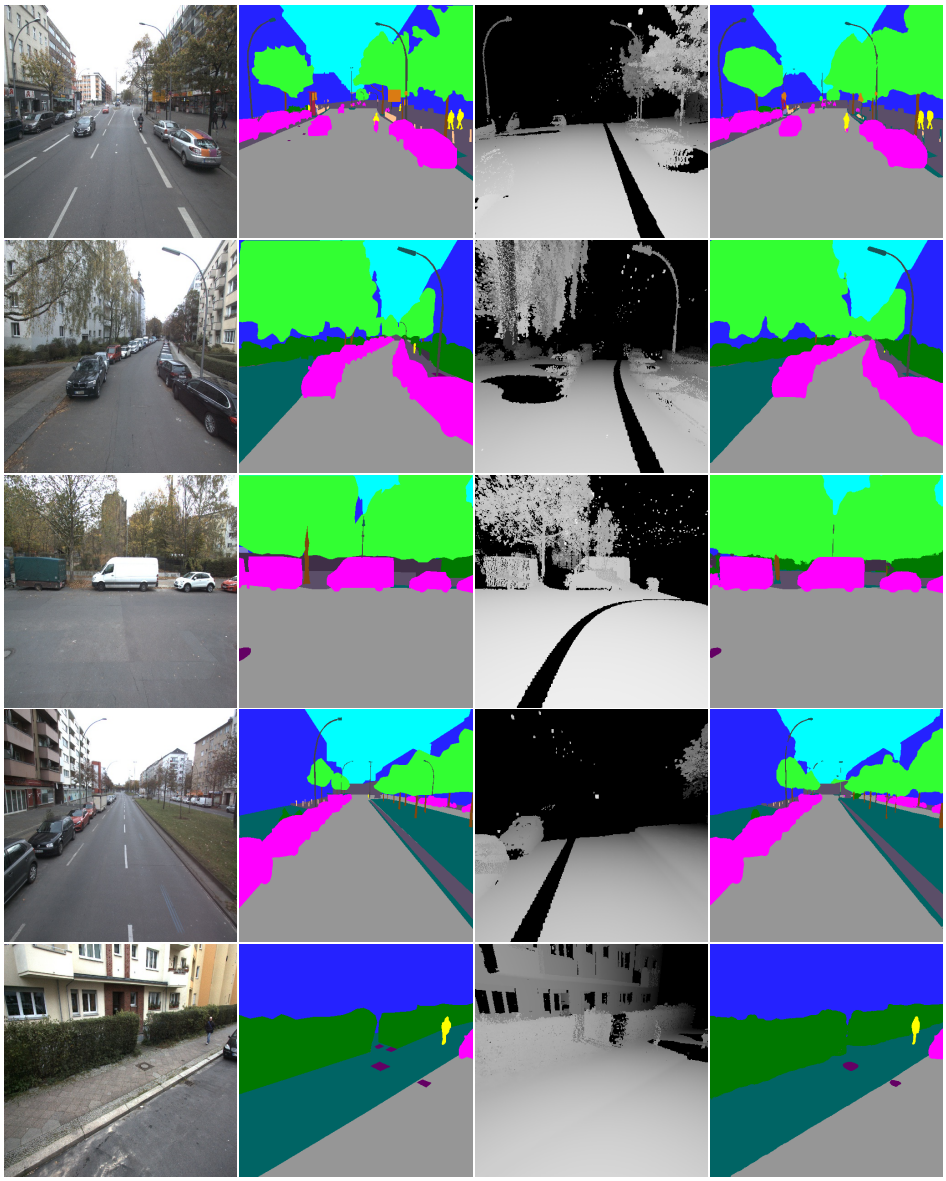
The test set comprises of 1353 data samples and this section showcases the quantitative and qualitative results of ResNet-34 on the test set.

ResNet-34 on val and test set			
Input type	**Global Acc**	**Mean Acc**	**Mean IoU**
Validation set (669 data samples)	94.83	60.59	53.17
Test set (1353 data samples)	94.36	58.35	51.91

Table 5: Performance of ResNet-34 on validation set (top row) and test set (bottom row)

The table above illustrates the validation and the test performance of our model. As we have tuned our architecture on the validation set, the validation score is slightly higher than the test score. The validation set is a part of the test set.

5.3.6 Visual analysis of ResNet-34 on test set



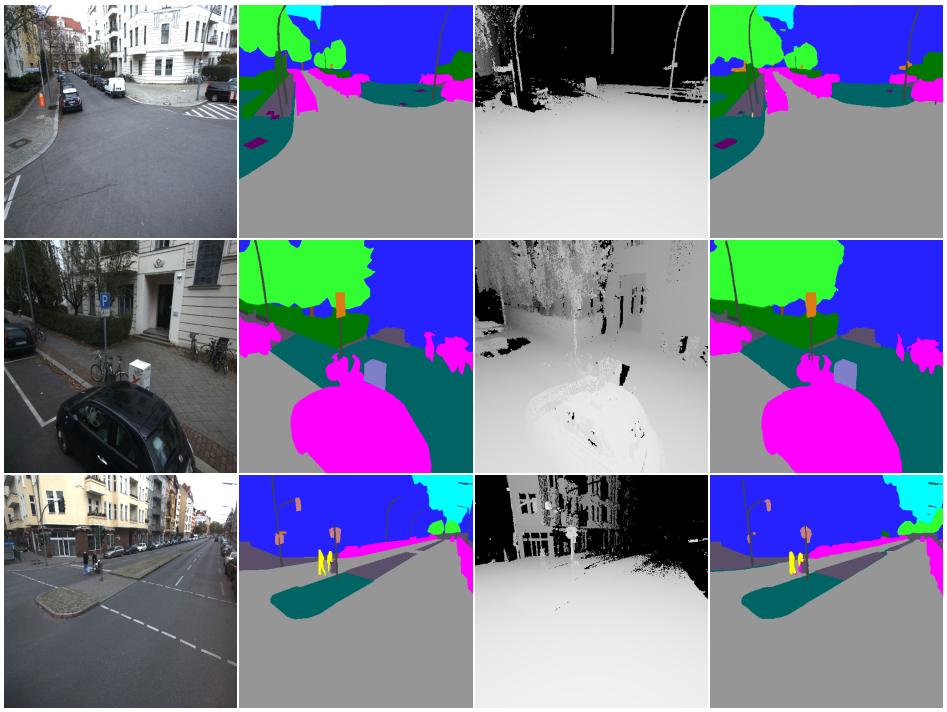


Figure 46: Qualitative visualization of ResNet-34 on test set. On the extreme left we have the RGB images, followed by the ground truth masks in the second column, next to it are the corresponding depth images and on the extreme right are the masks produced by ResNet-34.

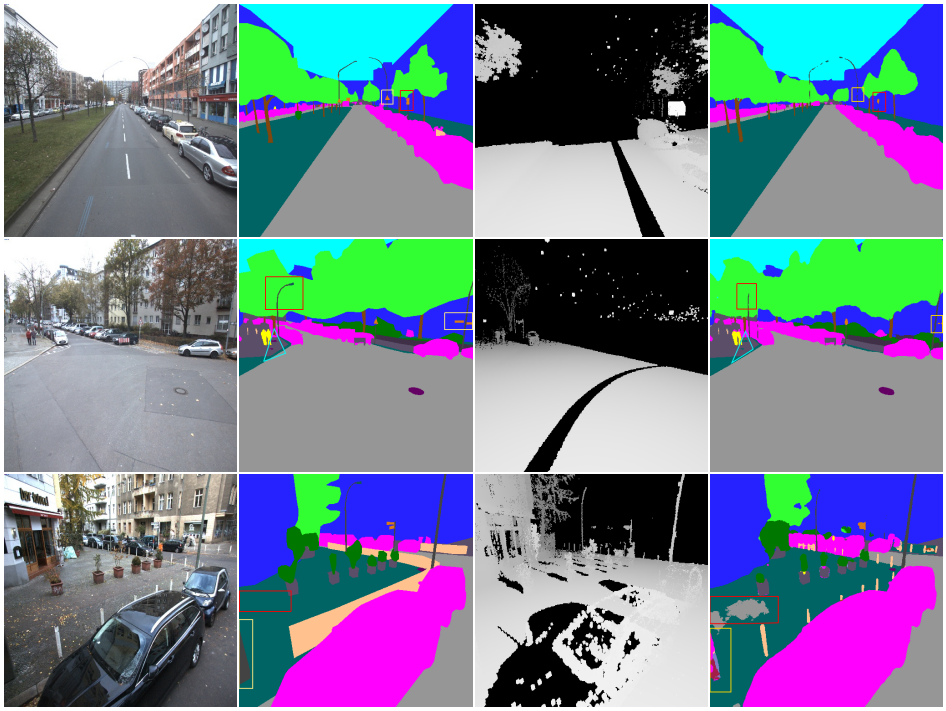
The figure 46 above illustrates the performance quality of ResNet-34. The experiment was conducted using RGB-D inputs. The first two rows are from camera1, third and fourth from camera2, fifth and sixth from camera3 and last two rows are from camera4. The network performs extremely well on all the four angles, but the side cameras has relatively better scores which is illustrated in table 6 below. This might be because the side cameras capture class details that occupy larger regions in the image compared to the front and rear which can capture the same classes relatively at larger distances therefore occupying smaller regions in the image. As for the metrics that validates the quality, the scoring is based on region occupancy, therefore introducing bias. Never the less, the *Global Acc* depicts a completely different story. However, quantitative evaluation in the figure 47 below shows that the differences among the angles are hard to detect.

Asphalt (road), pavement, trees, building, sky and vehicle are some of the dominant classes in the dataset that the model was able to segment with high precision. These classes appear in 90% of the train set. Apart from these, we had some of the challenging classes like *traffic lights, street lights, street signs,*

metal poles, people, hole covers, bollards etc that our model handled it really well, exhibiting sharp and well defined segmentation.

Test set scores on individual camera angles			
Camera angles	**Global Acc**	**Mean Acc**	**Mean IoU**
Camera1 (Front view)	95.28	54.24	48.59
Camera2 (Rear view)	94.98	57.02	50.54
Camera3 (Left view)	94.09	59.09	52.29
Camera4 (Right view)	93.08	63.07	55.72

Table 6: Comparison of the segmentation performance of the cameras separately



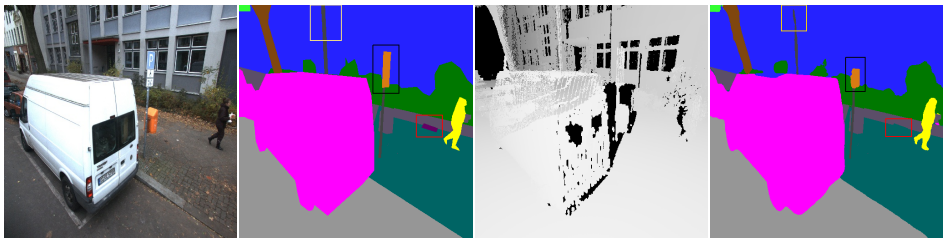


Figure 47: Qualitative visualization of ResNet-34 on test set on four different camera angles. First column contain original images, second column contain original masks, third column contain the depth images and fourth column contain the prediction masks from ResNet-34. The four rows represent four different views from cam1,cam2,cam3 and cam4 respectively.

The figure above contain images from all the camera angles starting from camera1 until camera4 from row1 to row4 respectively. The quality of segmentation on the four different angles are almost similar. Classes like *hole cover, street sign, metal pole* etc are equally challenging for the model to classify irrespective of the view. The higher *Global Acc* for the front and the rear views can be attributed to the fact that most of the dominant classes like *sky, pavement, building, vehicles etc* are present in the images from the former camera angles occupying major proportions in a single image for over 80% of the data in the train set. This could have pull up the global score as these classes are better classified than the rest.

5.3.7 RGB-D vs RGB segmentation

This section highlights a comparison study of the model with and without depth for segmenting images. The table below represents the quantitative advantage of using RGB-D inputs over RGB. This is further followed by illustrating the comparison in terms of visual representation. Using depth as an additional channel do provide a slight advantage over few classes but the dominant classes are predicted more or less with the same accuracy. Moreover, we retrained our model for both RGB and RGBD inputs three times each and took the average of the performance to assert our claims on the comparison. We also report the standard deviations of the individual metrics to assure less fluctuations in the model certainty in predicting the classes. We report the individual class scores in the form *Mean Acc*/$\pm$*stddev* and *Mean IoU*/$\pm$*stddev*.

RGB-D vs RGB (averaged on test set)			
Input type	**Global Acc**	**Mean Acc**	**Mean IoU**
RGB-D	94.33	**58.51**	**51.97**
RGB	**94.38**	57.17	51.01

Table 7: Performance evaluation of RGB-D and RGB segmentation

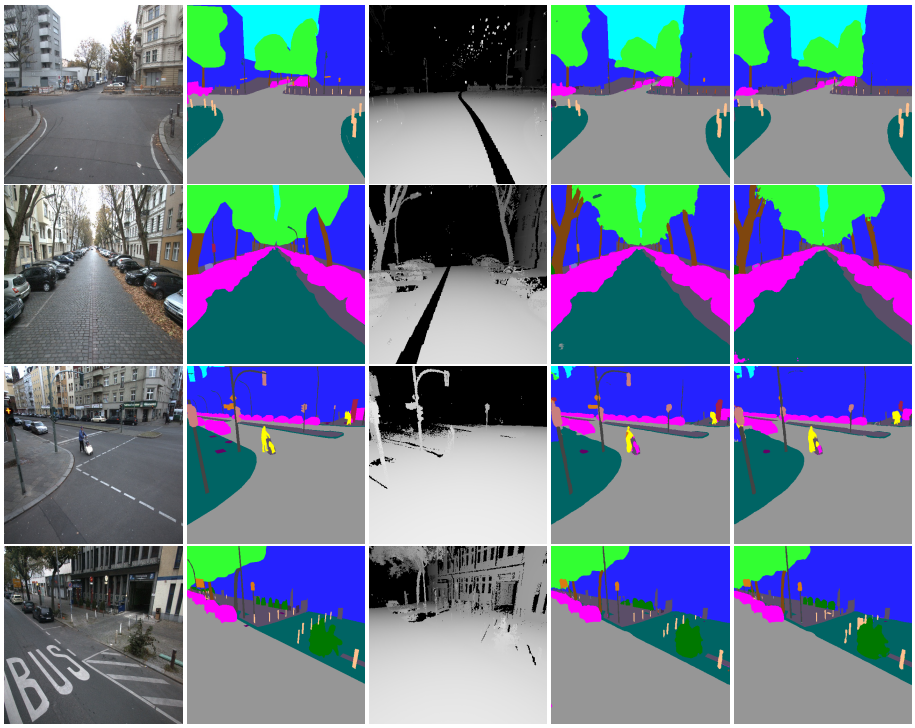


Figure 48: Illustrating performance comparison of using RGB-D inputs to RGB. Extreme left are the RGB images, next to it are the original masks, to it's right are the depth images, fourth column holds masks produced using RGB-D inputs and extreme right are masks from RGB inputs. The four rows represent four different views from cam1,cam2,cam3 and cam4 respectively.

The figure above shows the qualitative analysis of the performance of the model with and without the assistance of depth. Major classes are segmented almost with the same accuracy with a little room for differences. The images from both RGB-D and RGB models look kind of similar with minor differences in classes like *traffic lights, steet lights etc.* The later section highlights these advantages of using depth in detail.

5.3.8 RGB-D vs RGB per-class segmentation

The table below summarizes the benefit drawn from using depth on a per class basis. It also shows how the depth can negatively affect the quality on certain classes as well. Certain classes like *hole covers, street signs, street lights, traffic lights, encasement of equipment, advertising panel etc* benefit excessively (more than 2% rise in IoU) from using depth information. These are highlighted by coloring the cell in blue. However, using depth had a drop in performance (more than 2% drop in IoU) in the *enclosure* class (highlighted in blue as well). We further discuss this in the section 5.3.9.

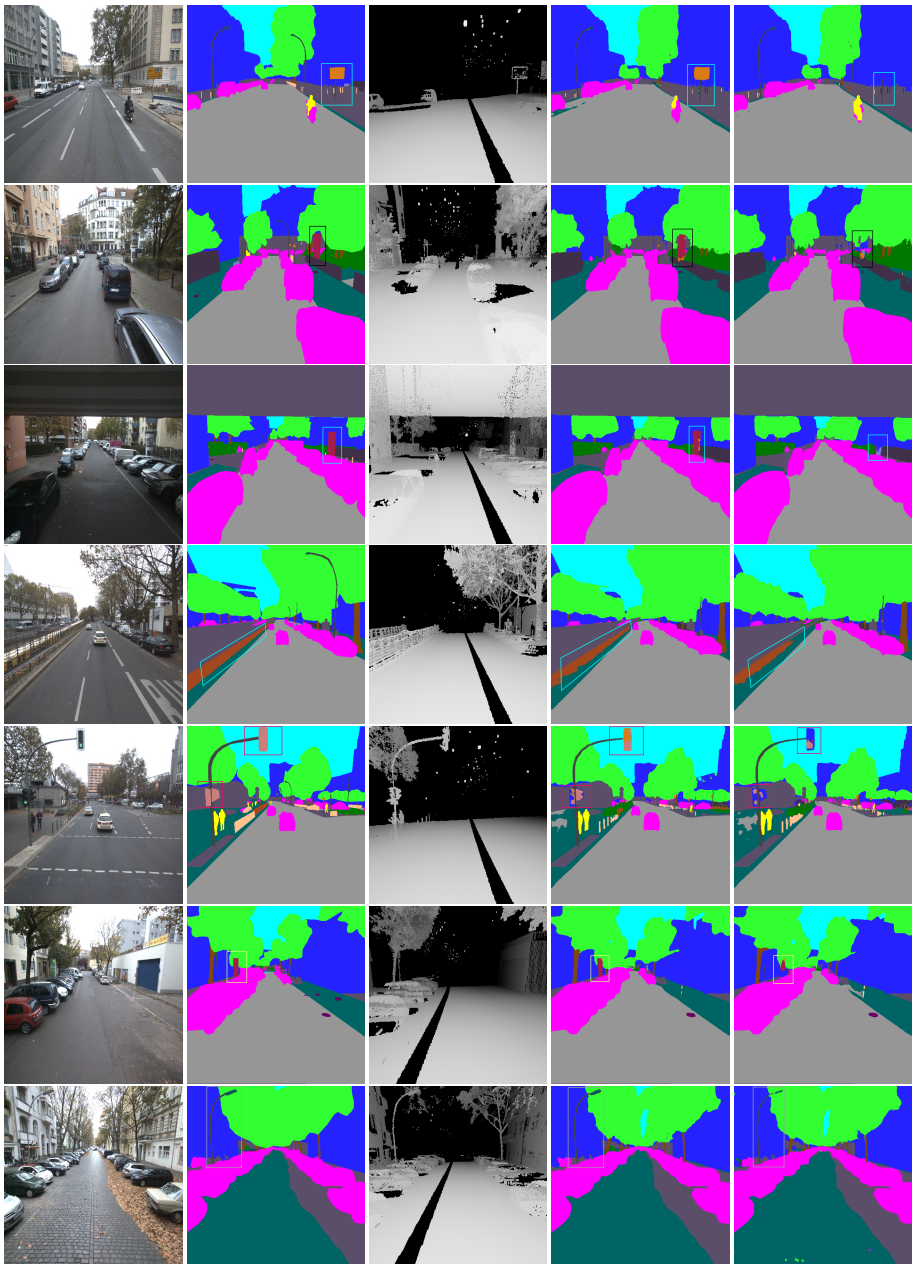
5.3.8.1 Quantitative analysis of individual classes from RGB-D and RGB segmentation

RGB-D vs RGB segmentation (per class)				
	RGB-D		RGB	
Class Names	**Mean Acc**	**Mean IoU**	**Mean Acc**	**Mean IoU**
Asphalt	96.80/±0.25	**95.32/±0.33**	96.32/±0.31	95.01/±0.21
Pavement	91.92/±0.20	80.96/±0.39	90.96/±0.26	**81.72/±0.03**
Hole cover	34.23/±3.68	**29.89/±2.03**	28.49/±0.94	26.36/±0.69
Tree trunk	57.78/±2.83	49.22/±1.02	60.10/±5.23	**49.55/±1.74**
Trees/crown	90.52/±0.52	**81.55/±0.08**	88.72/±1.18	81.01/±0.38
Bush	71.03/±3.40	**54.36/±0.81**	67.35/±1.13	53.64/±0.47
Street sign	43.43/±0.53	**35.78/±0.01**	34.57/±3.18	31.21/±2.42
Street light	16.85/±2.22	**14.47/±1.6**	9.85/±2.49	9.17/±2.06
Traffic light	23.74/±2.69	**21.31/±2.30**	18.00/±1.23	16.42/±0.77
Guardrail	44.68/±0.90	**41.24/±0.27**	42.36/±5.29	39.23/±4.56
Encasement of equipment	45.56/±2.20	**38.68/±0.81**	38.64/±1.40	34.67/±0.64
Advertisement panel	38.36/±0.94	**34.52/±0.74**	33.86/±0.32	31.30/±0.23
Building	90.75/±0.94	84.61/±0.15	92.42/±0.51	**84.92/±0.02**
Enclosure	45.77/±2.17	40.19/±1.54	54.21/±3.12	**44.92/±0.96**
Metal pole	50.57/±0.26	43.58/±0.83	53.15/±2.45	**45.29/±1.60**
Sky	85.11/±1.72	77.01/±0.05	84.22/±0.94	**77.28/±0.40**
Vehicle	94.67/±0.70	**89.36/±0.22**	94.37/±0.05	89.17/±0.08
People	48.69/±0.63	41.71/±0.25	55.20/±1.97	**43.88/±2.61**
Bollard	41.30/±4.52	33.79/±2.33	43.41/±2.10	**34.41/±0.33**
Mean	**58.51**	**51.97**	57.17	51.01

Table 8: Class specific comparison scores between RGB-D and RGB data

From the above table, we can see that the dominating classes like sky, asphalt(road), pavements, trees, vehicle etc exhibit similar performance irrespective of using depth. The class *bollard* is colored pink as this is a case of inconsistent labelling as mentioned in the section 4.1.3.2 rather than incompetency of the RGB-D model to segment better. This is also illustrated in the qualitative section below.

5.3.8.2 Qualitative analysis of individual classes from RGB-D and RGB segmentation



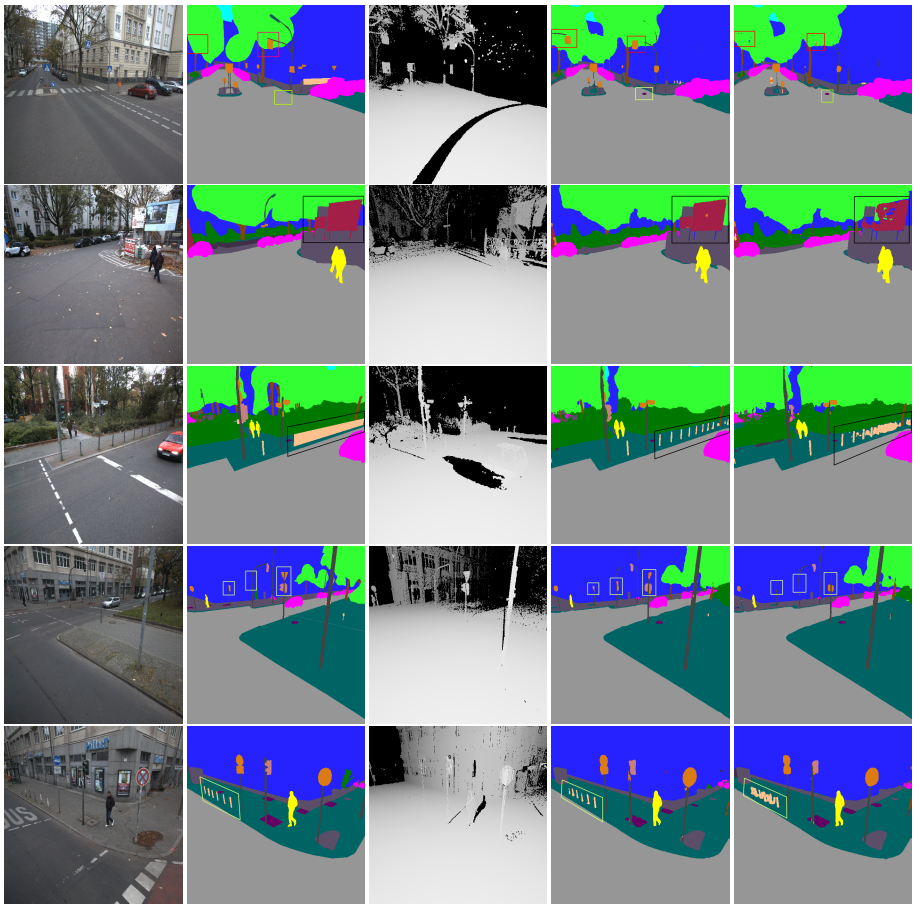


Figure 49: Showcasing qualitative differences per class for RGB-D and RGB inputs. First column holds images, second holds original mask, in the center we have the depth images, fourth and fifth column represents qualitative results using RGB-D and RGB inputs respectively.

The figure above illustrates the advantages of using depth information for segmentation. These are highlighted using colored boxes. As per the table 8, we can see that *advertisement panels, traffic lights, street signs, street lights, encasement of equipment etc* drew major benefits from using depth. The depth plays an important role for these challenging classes as they are some of the very important components for a road. It brings the advantage in segmenting different classes with similar appearance but different shapes as illustrated in the 2nd row of the figure. Nevertheless, the class *enclosure* was better segmented in the absence of depth. The *bollards* here are a special case. Their quantitative and qualitative performances contradict each other as mentioned in the section 4.1.3.2 earlier. The use of depth in fact helps in segmenting this class

more distinctly where each pole is classified independently. This is shown in the 10th and 12th row in the figure above. Though the class *people* and *metal pole* has better scores using the RGB model, we did not find any significant drop in quality with the RGB-D model.

5.3.9 Analyzing the quality drop of class Enclosure during RGB-D segmentation

The aforementioned class has nearly 6% drop in IoU when segmentation was performed using depth. Hence we decided to study the problem to what exactly caused this poor performance. We computed the confusion matrix of the test set to check for classes that got confused for the class *enclosure*. We can see that classes like *pavement, building, bush* etc has a higher confusion rate which are marked in green boxes. the class *people* has a higher confusion rate with the class *vehicle* which is also marked green. The matrix is illustrated below.

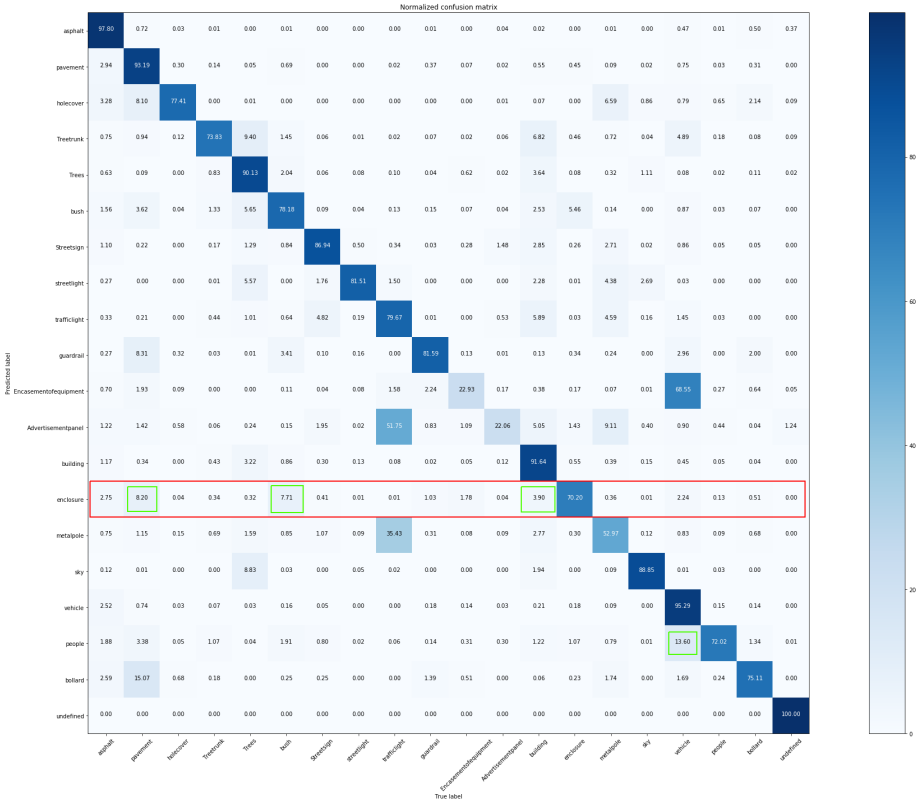
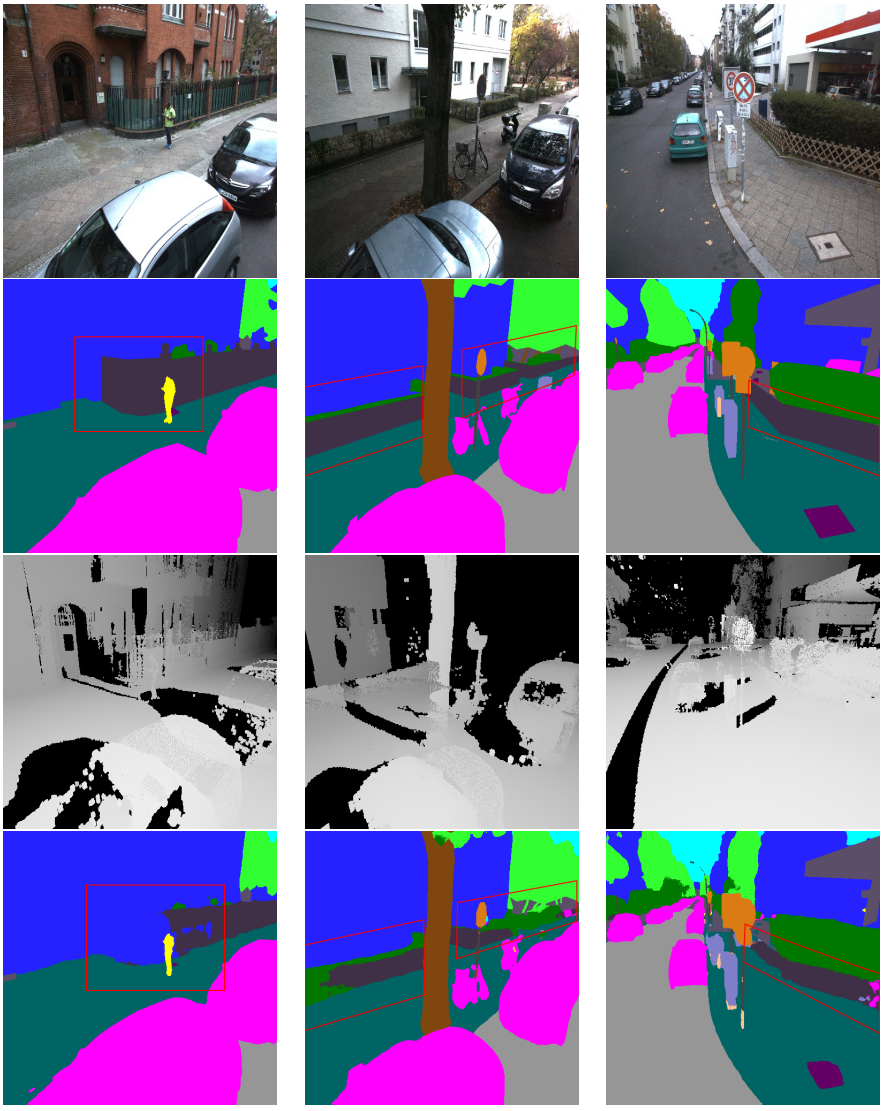


Figure 50: Confusion matrix of the test set for RGB-D model

To better understand the reason behind the uncertainty in predicting the class *enclosure*, we decided to make a visual comparison of couple of images with their corresponding masks and depth maps. From the figure below we can see that the classes like building, bush in the images exhibits structural similarity in their corresponding depth maps. We believe, there can be a possibility that due to similar visual patterns the model when using depth might confuse one class to be another resulting in a misclassification.



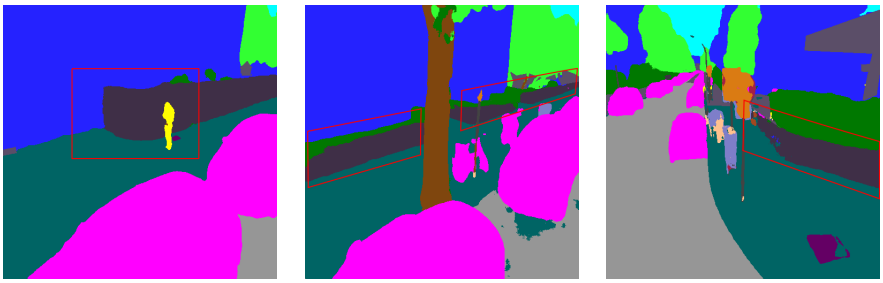


Figure 51: Visualizing the structural similarity of the class *enclosure* to that of *bush* and *building*. The first row contains the images, second row contains the ground truth masks, third row has the depths, fourth row contains masks from RGB-D segmentation and fifth row contains masks from RGB segmentation

5.4 Comparison of Fusenet approach to conventional RGB-D stacking

Fusenet [28], as discussed in section 2.5 came up with an alternative approach to use depth along with RGB images which supposedly can provide better segmentation quality than the conventional way of stacking depth along side RGB channels. They used a separate encoder for RGB and depth respectively as two independent processing steps and fused the depth at regular intervals after activation before downsampling using element-wise summation. Finally, the decoder integrates the information before evaluation. To test the integrity of their claims, we decided to reproduce their approach using our architecture model and dataset. The fusion approach they described in the paper are of two types namely *sparse fusion* and *dense fusion*. We decided to stick to the sparse fusion approach as it produced better results compared to the latter one. Based on the proposed approach, we built our model as illustrated in the figure 52.

The RGB and depth encoder are modelled in a similar fashion having the same structure and channels that runs parallelly with intermediate integration before merging into the decoder. To start with, the encoders consist of a stack of three layers of Convolution, Batch Normalization and ReLu activation each with 64 feature channels followed by element-wise summation for fusion. Then we max pool them to reduce it to half the original size followed by *residual branch 1* with 64 feature channels and fusion, *residual branch 2* with 128 feature channels followed by fusion, *residual branch 3* with 256 feature channels followed by fusion and *residual branch 4* with 512 feature channels and fusing them before decoding. The decoder has a similar structure to the original ResNet-34 we use in the project. Each branch has a deconvolution layer followed by concatenation from corresponding encoder level with fused features to make sure we take the advantage of both RGB and depth processing followed by two stacks of Convolution, BatchNorm and ReLu. The four branches in the decoder has $256, 128, 64, 32$ feature channels respectively in the top down fashion before

evaluating the score.

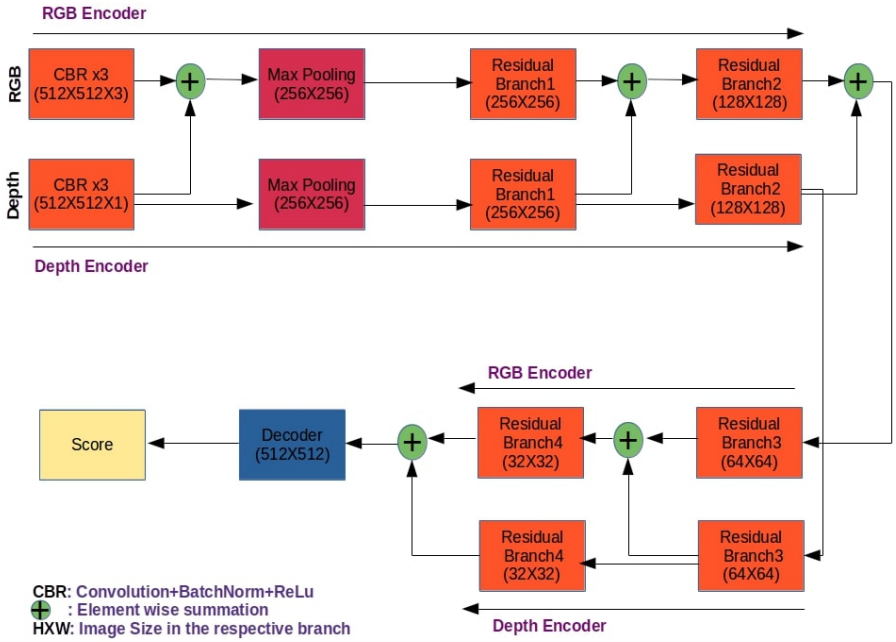


Figure 52: ResNet34 clubbed with UNet decoder based on fusion approach

5.4.1 Quantitative analysis of RGB-D stacking to RGB-D fusion

In table below we compare the quantitative results from fusion based ResNet-34 to that of ResNet-34 by stacking the depth (early fusion/Stacked RGB-D). To ensure the results and support our claims we retrain each network pertaining to an approach three times to make sure that the results are consistent and not random. We then report the mean evaluation from each set of three trials along with their standard deviations for both the approaches. Below are two tables showcasing the quantitative results, average IoU and accuracy on the entire set followed by average IoU and accuracy per class on the test set with their respective standard deviations.

Stacked RGB-D vs Fused RGB-D			
Input type	**Global Acc**	**Mean Acc**	**Mean IoU**
Stacked RGB-D	**94.36**	**58.07**	**51.60**
Fused RGB-D	93.61	54.24	47.97

Table 9: Performance evaluation of Stacked RGB-D and Fused RGB-D segmentation

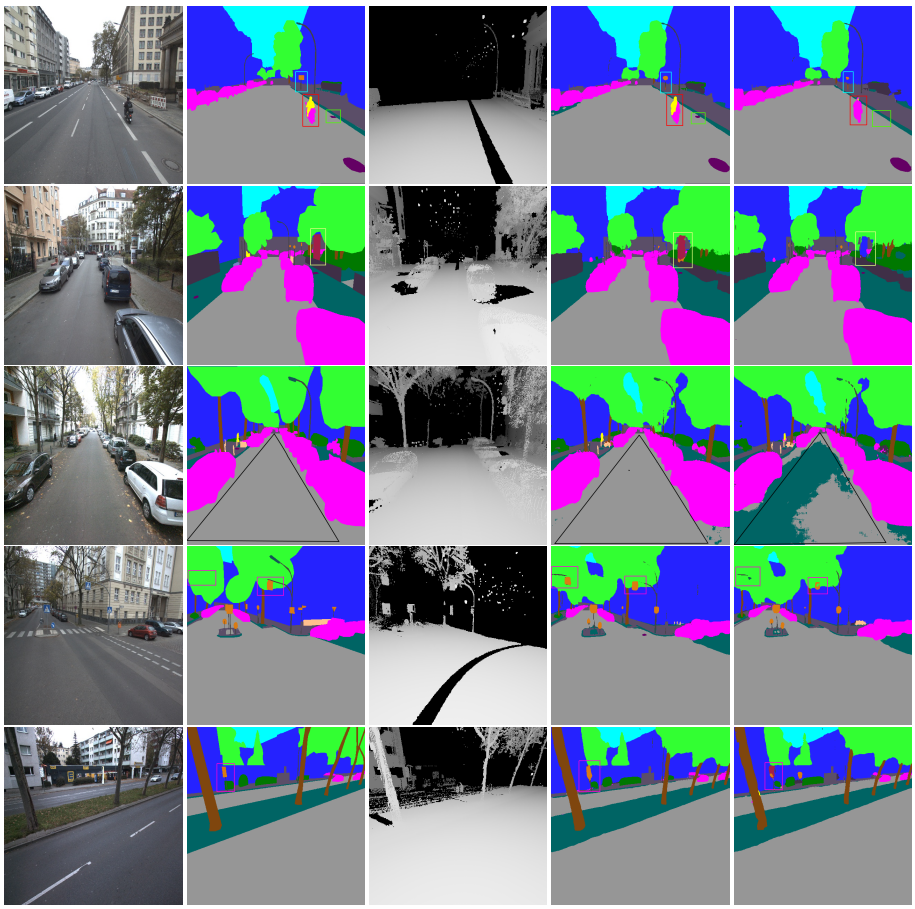
Stacked RGB-D vs Fused RGB-D (per class)				
	Stacked RGB-D		**Fused RGB-D**	
Class Names	**Mean Acc**	**Mean IoU**	**Mean Acc**	**Mean IoU**
Asphalt	96.90/±0.25	**95.41/±0.29**	95.82/±0.82	94.49/±0.89
Pavement	91.46/±0.67	**81.72/±0.03**	91.62/±1.91	78.87/±1.20
Hole cover	32.67/±3.73	**28.90/±2.17**	27.71/±6.41	24.28/±4.96
Tree trunk	62.56/±4.30	**50.69/±1.60**	54.92/±0.32	46.21/±0.49
Trees/crown	89.83/±1.06	**81.45/±0.18**	86.99/±3.06	79.39/±1.20
Bush	70.10/±3.08	**54.25/±0.68**	61.06/±4.64	49.66/±2.13
Street sign	38.80/±6.56	**33.01/±3.92**	34.27/±4.14	29.89/±3.21
Street light	15.16/±3.00	**13.33/±2.08**	11.91/±3.01	10.53/±2.34
Traffic light	20.88/±4.61	**18.99/±3.79**	18.84/±2.52	17.31/±1.99
Guardrail	43.46/±1.88	**40.11/±1.61**	25.43/±3.33	24.40/±3.07
Encasement of equipment	41.77/±5.66	**36.26/±3.44**	40.09/±3.04	33.18/±2.64
Advertisement panel	36.17/±3.20	**32.93/±2.33**	30.54/±4.20	27.97/±3.49
Building	91.53/±1.33	**84.62/±0.12**	91.96/±2.24	83.51/±0.70
Enclosure	46.36/±1.95	40.42/±1.30	53.85/±2.92	**42.62/±0.32**
Metal pole	52.74/±2.08	**45.32/±1.30**	52.29/±2.97	43.51/±1.69
Sky	85.38/±1.45	**77.20/±0.35**	86.30/±2.91	76.59/±0.44
Vehicle	94.56/±0.60	**89.46/±0.19**	93.93/±1.00	88.03/±0.87
People	53.18/±1.94	**43.49/±1.06**	34.37/±1.82	31.39/±1.60
Bollard	40.27/±3.97	**33.14/±2.11**	38.73/±1.28	29.55/±1.73
Mean	**58.07**	**51.60**	54.24	47.97

Table 10: Comparing individual class scores between Stacked RGB-D and Fused RGB-D approaches

From the table 10 above, we can see the per class scores of both the approaches. IoUs with more than 2% difference are highlighted by coloring the cell in blue. Fusing depth using element-wise summation has degraded the performance of 17 classes out of 19 which is depicted using the IoU scores. The overall accuracy and IoU on the test set is comparatively lower than the conventional approach of stacking depth along RGB channels. Couple of classes like *pavement, bush, traffic light, guardrail, encasement of equipment, people* have extremely low IoU scores when segmented using the fusion based approach. Having said that, class *enclosure* do show improved performance compared to the conventional approach of stacking RGB-D, which is also seen when compar-

ing RGB-D segmentation to RGB in table 8. This can be attributed to the same fact as mentioned in the section 5.3.9 where the class *enclosure* is misclassified with classes like *bush, building etc* during early fusion. Analyzing the accuracies of the individual classes for both the approaches, we can see that 13 classes out of 19 has a lower performance scores when segmented by fusing RGB and depth. Out of these, classes like *bush, street sign, traffic light, guardrail, encasement of equipment and people* have extremely low scores compared to the Stacked RGB-D model for segmentation. The figure below illustrates visual differences between both the approaches.

5.4.2 Visual comparison of depth stacking to depth fusion



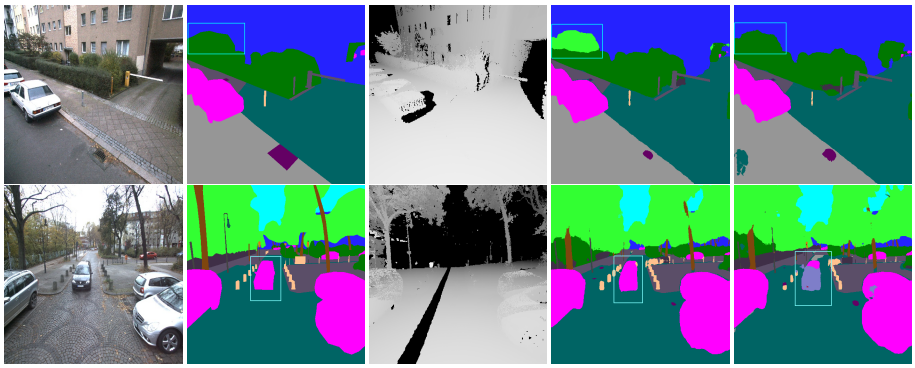


Figure 53: Qualitative comparison between Stacked RGB-D to Fused RGB-D approach. First column holds images, second holds original mask, in the center we have the depth images, fourth and fifth columns represents results from Stacked RGB-D and Fused RGB-D models respectively.

From the figure above, we can clearly see the drawbacks of the depth fusion approach. Processing the depth independently weakens the model's capacity to use the information with corresponding RGB data that can improve the quality of segmented classes. It is counter-productive and the depth can no longer be used to distinguish different objects with similar appearances but different sizes. The basic intuition behind this is depth and RGB are processed in two independent steps and depth alone is not informative enough to bring out the best representation of features pertaining to respective classes. Its very likely that a particular class in the image can be represented in two different ways using RGB and depth independently that are non-complementary. Fusing these two features in this case will end up weakening the model decision. This can be seen in the figure at the 3rd and 7th row where the model produces a lot of artifacts and confuses one class to be another. The fusion is based on element-wise summation which happen to nullify signals with different magnitude. Summing up two signals with different magnitudes can result in masking the signal with smaller magnitude by the higher magnitude signal. This can result in perturbation in the final signal. That is, if the model processes a region of the image corresponding to a certain class independently based on RGB and depth respectively, in ways that are not complementary to each other, then summation of their activations can have a reverse effect in segmenting the class quality.

Furthermore, we decided to compare another Stacked RGB-D model by reducing the number of parameters to that of the Fused RGB-D. This was done to analyze if the Fusenet having extra parameters can deliver any advantage over the early fusion approach.

5.4.3 Quantitative analysis of RGB-D (reduced) stacking to RGB-D fusion

Stacked RGB-D vs Fused RGB-D			
Input type	**Global Acc**	**Mean Acc**	**Mean IoU**
Stacked RGB-D	**94.18**	**56.24**	**49.58**
Fused RGB-D	93.63	55.00	48.77

Table 11: Stacked RGB-D vs Fused RGB-D segmentation

Stacked RGB-D vs Fused RGB-D (per class)				
	Stacked RGB-D		**Fused RGB-D**	
Class Names	**Mean Acc**	**Mean IoU**	**Mean Acc**	**Mean IoU**
Asphalt	95.75	**94.62**	94.93	93.39
Pavement	91.69	**80.73**	93.14	77.33
Hole cover	28.38	**24.68**	27.59	24.3
Tree trunk	58.15	**47.86**	54.61	46.91
Trees/crown	90.56	**81.18**	88.2	80.76
Bush	68.4	**52.87**	61.12	50.19
Street sign	29.77	26.82	38.88	**33.76**
Street light	7.14	6.75	15.81	**13.45**
Traffic light	20.33	**18.33**	15.96	15.2
Guardrail	39.2	**36.46**	28.43	27.08
Encasement of equipment	36.89	32.78	38.37	**33.42**
Advertisement panel	28.94	27.33	35.46	**32.01**
Building	91.22	**84.5**	92	84.42
Enclosure	52.8	42.75	57.02	**43.01**
Metal pole	49.72	42.23	51.34	**43.77**
Sky	83.74	**77.07**	87.31	77.06
Vehicle	93.22	**89**	93.86	88.97
People	52.65	**41.64**	33.2	30.71
Bollard	49.93	**34.37**	37.84	30.78
Mean	**56.24**	**49.58**	55.00	48.77

Table 12: Comparing individual class scores between Stacked RGB-D and Fused RGB-D approaches

Above are two tables illustrating the quantitative results of both the approaches. From the table 11 above, we can see that a smaller model can still perform better with a reduced set of parameters if depth is not fused for the learning process. Having said that, table 12 showcases the per class comparison of both the approaches. Having lesser parameters do decreases the scores of some classes but 13 out of 19 classes still have better results with the stacked (early fusion) approach. IoUs with more than 2% difference are highlighted in blue.

5.4.4 Visual comparison of stacked RGB-D (reduced) to Fused RGB-D to RGB

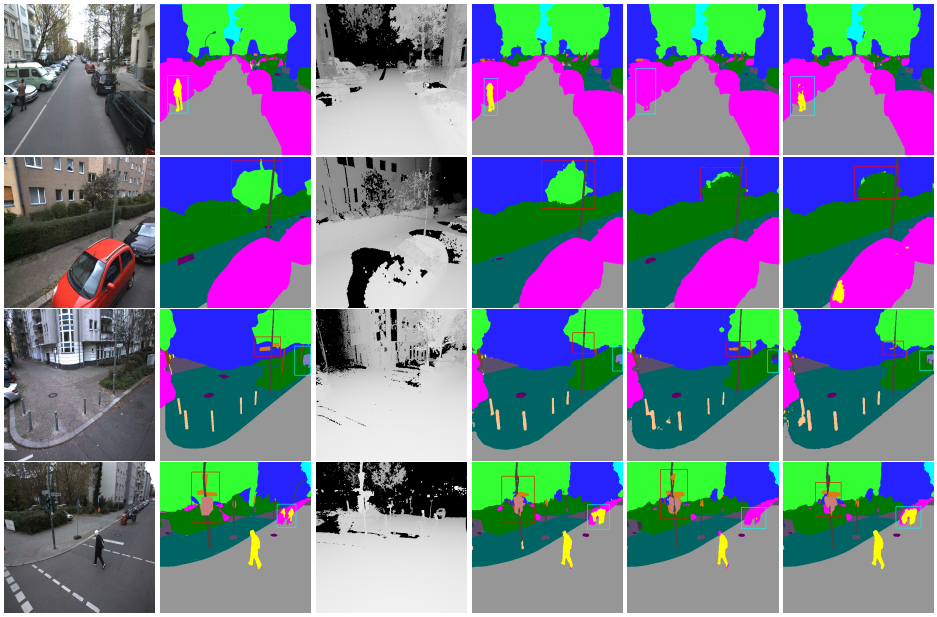


Figure 54: Qualitative comparison between Stacked RGB-D with reduced set of parameters to Fused RGB-D approach to RGB model. First column holds images, second holds original mask, third holds depth images, fourth and fifth column represents results from Stacked RGB-D and fused RGB-D model respectively and sixth represents results from the model based on RGB inputs.

In the figure above, we visualize the contrast drawn between the two models using depth stacking and fusion. With a reduced set of parameters, the model is still able to extract the benefits of depth when stacked channel wise to distinguish between objects that are blended together giving an illusion of belonging

to a single class which can be seen in the second row. The fusion based approach, however is able to segment the classes like *street signs, traffic lights, hole covers* better as mentioned in the table 12 above due to more parameters. But again, in many cases fusing the depth has proven unproductive which is illustrated by contrasting the fused model with the model that learns only using RGB data. In the 1st, 3rd and 4th row, we can see that the RGB model is still able to separate classes that are in close proximity better than the model based on depth fusion which are highlighted using the colored boxes. The figure as a whole also showcases the strength of the application of depth information by a model with a smaller set of parameters which can surpass a much bigger RGB model in segmenting challenging classes and their respective blends.

To explore this approach in more depth, we decided to try out a variation of the fusion strategy used above in an attempt to ensure we don't miss any existing methodologies that could have made the Fusenet [28] feasible enough for real time solutions, as our work and results attempt to falsify the claims about the approach. For this, we replaced the *element-wise* fusion method by *concatenation*. Rest of the model remains the same and we perform as usual *sparse fusion* before every downsampling after activation. Below, we illustrate the quantitative comparison of the two late fusing approaches based on element-wise summation and concatenation to that of the early fusion.

5.4.5 Quantitative analysis of RGB-D stacking to RGB-D fusion (element-wise sum) to RGB-D fusion (concatenation)

Stacked RGB-D vs Fused RGB-D (sum) vs Fused RGB-D (concat)			
Input type	**Global Acc**	**Mean Acc**	**Mean IoU**
Stacked RGB-D	**94.36**	**58.35**	**51.91**
Fused RGB-D (sum)	93.63	55.00	48.77
Fused RGB-D (concat)	94.21	54.78	49.38

Table 13: Stacked RGB-D vs Fused RGB-D (element-wise sum) vs Fused RGB-D (concat) segmentation

From the table above, we can see that the fusing depth by concatenation did not bring any notable changes to the quantitative evaluation. Both of the fusion approaches have less than 1% variation in their accuracy and IoU. As a whole, the fusion based model still stays behind the conventional RGB-D stacking model. For a more detail analysis let us have a look at the independent class scores of the concatenation approach.

The table 14 below illustrates the class wise comparison of the late fusion approaches to the early fusion. We clearly see that the fusion approach based on concatenation did not yield any promising results. The highest scores (IoU) among the three approaches are made bold to highlight the associated model.

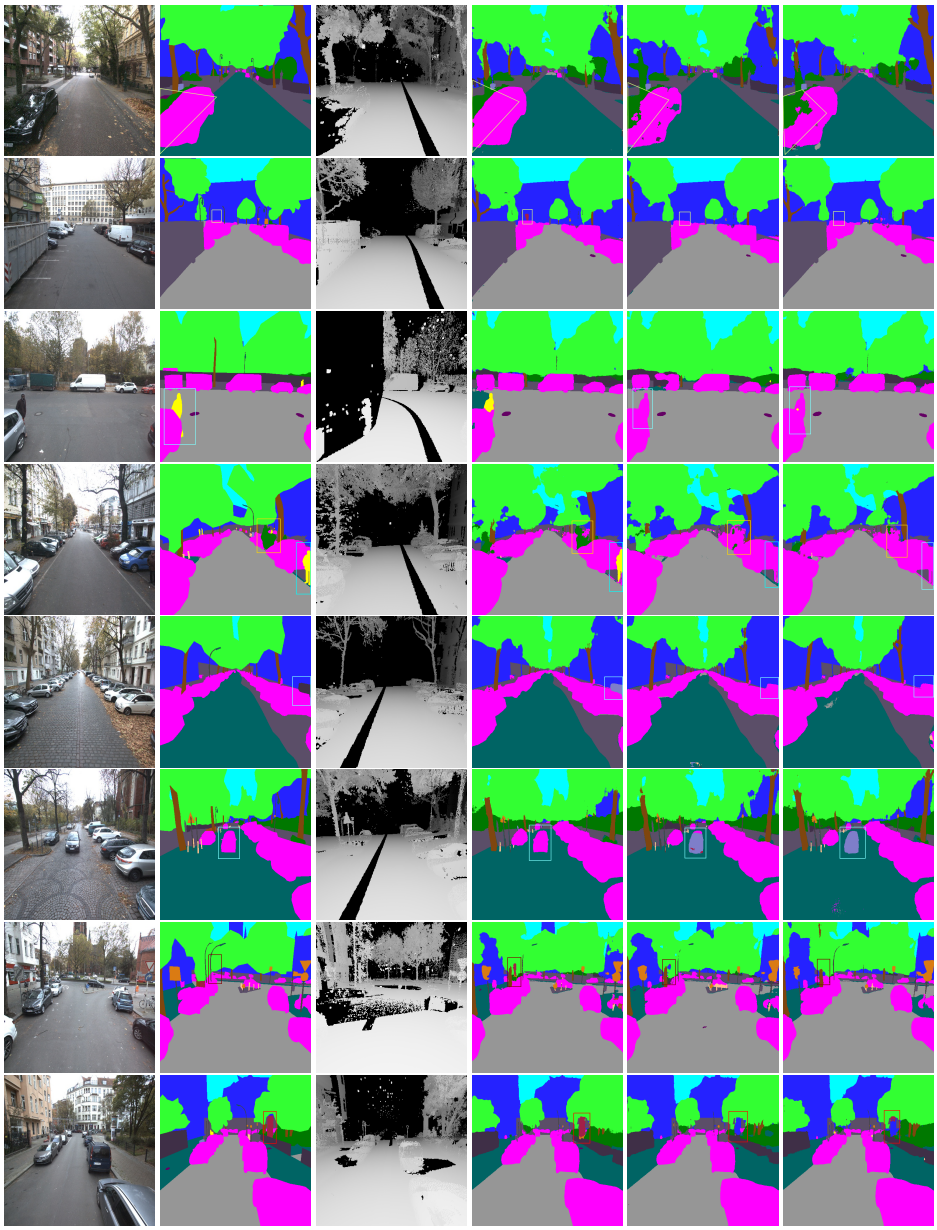
Stacked RGB-D vs Fused RGB-D (sum) vs Fused RGB-D (concat)						
	Stacked RGB-D		**Fused RGB-D (sum)**		**Fused RGB-D (concat)**	
Class Names	**Mean Acc**	**Mean IoU**	**Mean Acc**	**Mean IoU**	**Mean Acc**	**Mean IoU**
Asphalt	97.05	**95.66**	94.93	93.39	96.99	95.44
Pavement	92.12	**81.36**	93.14	77.33	91.60	80.29
Hole cover	30.55	**27.85**	27.59	24.3	30.02	27.09
Tree trunk	54.95	48.19	54.61	46.91	58.41	**49.04**
Trees/crown	91.04	**81.47**	88.2	80.76	87.85	80.74
Bush	74.44	**55.17**	61.12	50.19	63.60	51.33
Street sign	43.96	**35.79**	38.88	33.76	40.55	35.05
Street light	19.07	**16.07**	15.81	13.45	15.13	13.44
Traffic light	26.44	**23.62**	15.96	15.2	22.76	20.79
Guardrail	43.77	**40.96**	28.43	27.08	26.73	25.33
Encasement of equipment	43.35	**37.84**	38.37	33.42	31.15	29.29
Advertisement panel	39.31	**35.26**	35.46	32.01	30.66	28.77
Building	91.91	**84.95**	92	84.42	92.39	84.59
Enclosure	43.59	38.64	57.02	**43.01**	48.82	42.33
Metal pole	50.84	44.42	51.34	43.77	54.88	**45.83**
Sky	86.84	76.96	87.31	**77.06**	81.76	76.05
Vehicle	95.38	**89.13**	93.86	88.97	95.65	89.02
People	49.33	**41.97**	33.2	30.71	41.00	36.41
Bollard	36.77	**31.45**	37.84	30.78	30.94	27.45
Mean	**58.35**	**51.91**	55.00	48.77	54.78	49.38

Table 14: Comparing individual class scores between Stacked RGB-D to Fused RGB-D (element-wise sum) to Fused RGB-D (concatenation) approaches

We also color the cell green if the highest score differs by more than 2%. Among the Fusion based approaches, we highlight the highest scores (IoU) that differs by 2% or more by coloring the cell blue. The Stacked RGB-D has 15 classes out of 19 with IoU scores higher than the fusion based approaches. Rest of the classes, though having lower scores vary less than 2% in their IoU values. Although one thing to note here is depth fusion via concatenation works comparatively better than fusing via element-wise summation which we can see from their individual class scores.

In the figure below, we analyze the visual differences between the depth fusion approach based on concatenation to that of the approach based on element-wise summation and to the usual depth stacking.

5.4.6 Qualitative analysis of stacked RGB-D to Fused RGB-D using element-wise summation to Fused RGB-D using concatenation



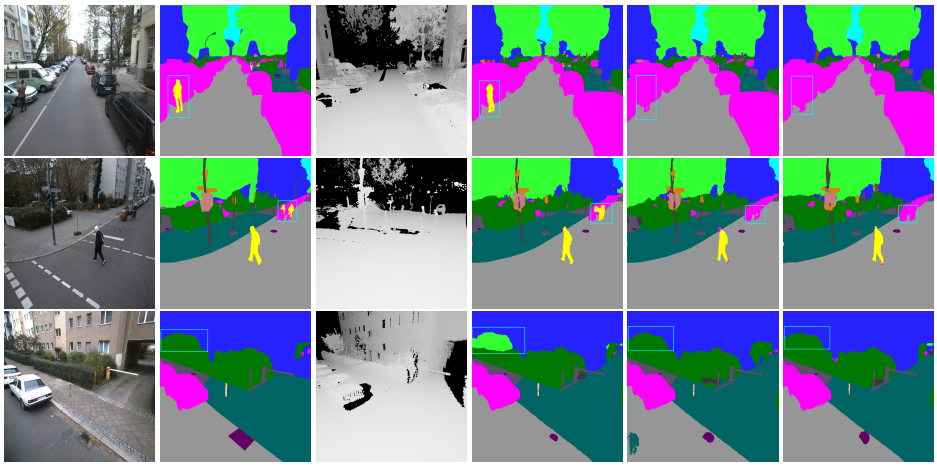


Figure 55: Qualitative comparison between Stacked RGB-D to Fused RGB-D using element-wise summation to Fused RGB-D using concatenation. First column holds images, second holds original mask, third holds the depth images, fourth, fifth and sixth column represents results from Stacked RGB-D, Fused RGB-D (element-wise sum) and Fused RGB-D (concatenation) respectively.

The figure above clearly illustrates the similar drawbacks that the concatenation based fusion faces to that of the fusion based on element-wise summation. The model still lacks the ability to differentiate objects with similar appearances and varying sizes. Although, fusing via concatenation produces lesser artifacts than it's former counterpart. Some of the images above are repeated from the figures 53 and 54 to make sure that the disadvantages of the independent depth processing and intermediate fusion are consistent across both the fusion approaches.

Following the above experiments, it is safe to conclude that processing RGB and depth independently and fusing them does not strengthen the model in any way that can facilitate better segmentation. The depth information is an additional channel that can aid a model along with RGB data to make quality decisions when both of them are used together. Quality segmentation is always dependent on RGB pixels which is more informative and depth alongside can bring out the structural differences between objects. In this way, having a good depth map along with the RGB data can avert a model from presuming one class to be another when defining region boundaries for classes that are blended together giving an impression of belonging to one class. When multiple modalities are used during early fusion, the model learns to use them smartly and weigh them according to their contribution to the classification decision. Any modal that exhibits a poor representation of certain feature can be outweighed or ignored and decision making is then based on the rest of the modals from which the model can infer sensible representations. If these modalities are used independently and forced fused, incorrect feature representation by one modal

fused to another act as a noise and mask the final signal finally weakening it's strength. The depth as an independent data source cannot suffice for a model to make correct feature representations. If these representations are not complementary to the RGB processing, any fusion can only weaken the final decision as we are trying to make a model learn two different representations of a single feature.

5.5 Comparing ResNet-34 to state-of-the-art DDNet Net

In this section we make a detail comparison of the ResNet-34 with the DPDB Net from [48] on our Berlin dataset. We make a couple of modifications on the ResNet-34 before proceeding. We remove the first *pooling layer* making the original image size to be processed by the the first residual block. Removing a downsampling step helps preserve information. Second, the decoder is made dense by introducing dense connections (concatenation operations) in between the CBR blocks. These connections can facilitate new feature exploration. Originally, a decoder branch consisted of a DBR block having *Deconvolution, Batch Normalization and ReLu* followed by a *Concatenation layer* and two CBR blocks having *Convolution, Batch Normalization and ReLu*. We modify these by concatenating the output from the *concatenation layer* to the output from the first CBR block and again concatenating this output to that of the output from the second CBR block. A visualization of this implementation is illustrated below. We are left with three branches in the decoder as *pooling* was removed.

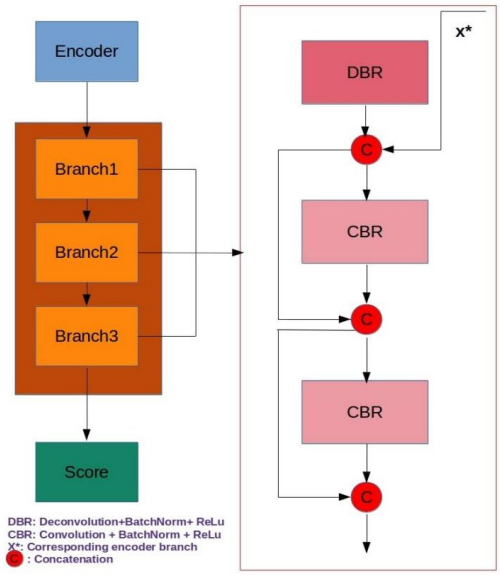


Figure 56: Dense connections in the decoder

5.5.1 Train & Test

The experiments were performed on 4894 training samples of size $304X304$. We performed dynamic augmentation that included random *translation, rotation, horizontal and vertical flips, brightness and contrast.* We validated the model on the validation set of 669 data samples for testing. We started with two basic experiments using ResNet-34 and DDNet with no change in parameters (using original models) followed by a fair comparison of both the models by reducing the number of parameters in the ResNet to match the DDNet. For a more detailed analysis we swapped our encoders and decoders to come up with a network with DPDB encoder and UNet decoder (DPDB-UNet) and a network with ResNet-34 encoder and DDNet decoder (Res-DDNet). We proceeded with further variations of the former model by changing the depth and width to conclude it's performance on the Berlin set. The experiments were evaluated using *Softmax Cross Entropy loss.*

The initial training learning rate was 0.0001 with an exponential decay of 0.9 after every epoch (4894 iterations). The batch size was 1. The optimizer used was *Adam* and momentum (1 and 2) were 0.9 and 0.999 respectively. We did not use any class balancing. We saved a model in every 5000 iterations. The train data was randomly shuffled and random dynamic augmentation was performed. The models were normally trained to a max of around $30-40$ epochs until convergence.

Testing was performed on the test set of 1353 data samples using the model with the minimum cross entropy loss and evaluated using the same metrics as described in section 5.2.1.

The experiments were conducted only on RGB data as we were more interested in the raw strength of the architectures and their respective topologies that can achieve good quality segmentation without additional assistance for improvement.

5.5.2 ResNet-34 vs DDNet

We compare the quantitative results from both the models in the Berlin testset. The models are of different sizes as ResNet-34 has more parameters than the DDNet. Here we try to analyze the strength of the original networks as both of them comes with varying depths and widths and network structures that can have diverse effects on a dataset.

5.5.2.1 Quantitative analysis of ResNet-34 and DDNet

The table below clearly indicates that ResNet-34 outperforms DDNet by a factor of about 14% in Mean accuracy and a facor of about 13% in Mean IoU. The Global accuracy is less informative as it does not consider the unbalance class distribution.

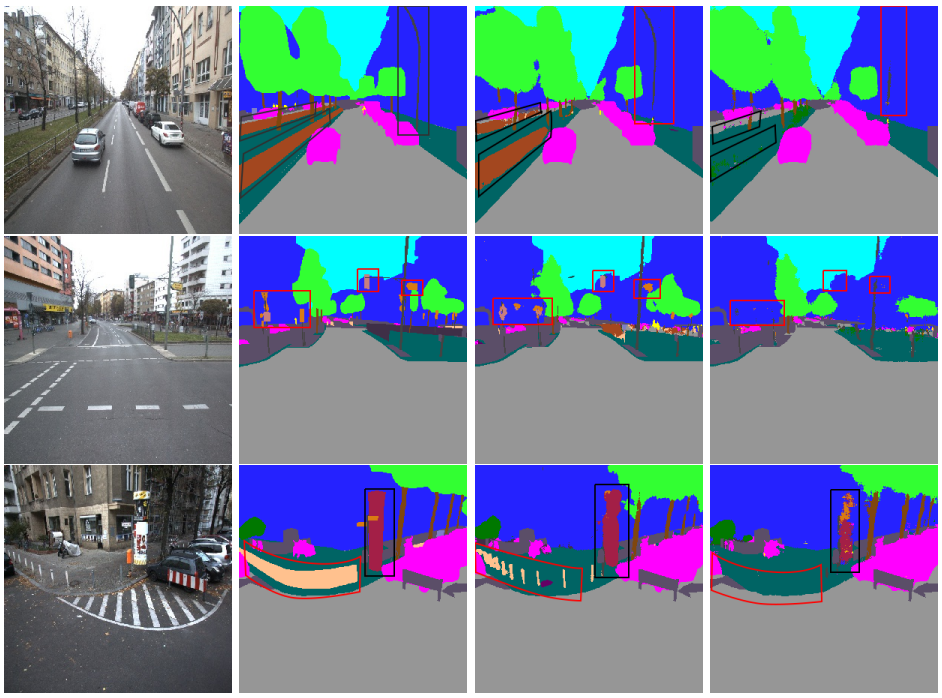
ResNet-34 vs DDNet				
Input type	**Parameters**	**Global Acc**	**Mean Acc**	**Mean IoU**
ResNet-34	105,294,420	**94.20**	**53.68**	**47.21**
DDNet	31,667,128	92.86	39.71	34.75

Table 15: Performance comparison of ResNet-34 and DDNet

It's possible that having almost more than 3 times the parameters, the ResNet-34 could have an advantage over DDNet. To analyze this better we then compare both the models again by equalizing their parameters. This is showcased in section 5.5.3.

5.5.2.2 Qualitative analysis of ResNet-34 and DDNet

The figure below draws the qualitative contrast between the two models. The differences are highlighted using colored boxes. The performance of the DDNet suffers extremely when it comes to classes like *enclosure, traffic lights, street lights, hole covers, metal poles, encasement of equipment, bollards* etc. These classes are near to not detected which reflects it's huge drop in the IoU and accuracy scores.



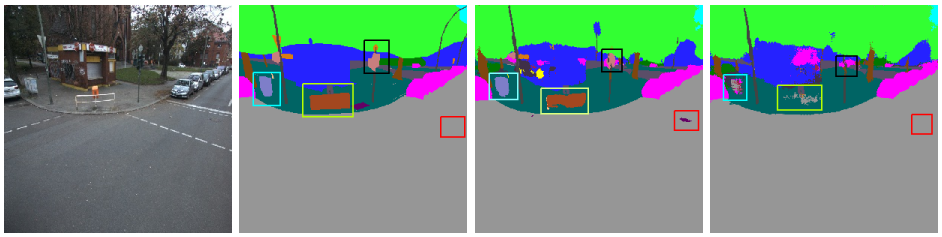


Figure 57: Qualitative comparison between ResNet-34 and DDNet. First column holds images, second holds original mask, third and fourth column represents results from ResNet-34 and DDNet respectively.

5.5.3 ResNet-34 (reduced) vs DDNet

In order to bring out a fair comparison of both the architectures and eradicate any additional advantage from having additional parameters we compare both the models again by reducing the size of the ResNet-34 to match approximately the size of the DDNet. The table below showcases the quantitative differences between the two models with approximately equal set of parameters.

5.5.3.1 Quantitative analysis of ResNet-34 (reduced) and DDNet

From the table below, we can see that ResNet-34 still outperforms the DDNet irrespective of similar sizes. Reducing the parameters resulted in a small drop in it's quantitative metrics but still the model surpasses the DDNet by a wide margin.

ResNet-34 vs DDNet				
Input type	**Parameters**	**Global Acc**	**Mean Acc**	**Mean IoU**
ResNet-34	31,239,636	**94.14**	**52.67**	**46.25**
DDNet	31,667,128	92.86	39.71	34.75

Table 16: Performance comparison of ResNet-34 and DDNet

From the table 15 and 16, we can see that both the ResNet-34 models has a very small difference in their performance although the ResNet-34 in table 15 has almost 3 times the parameters to it's smaller counterpart in the table 16. This can be attributed to the reduced variability of the Berlin set as the images corresponding to the different scenes does not vary much in their visual patterns.

5.5.3.2 Qualitative analysis of ResNet-34 and DDNet

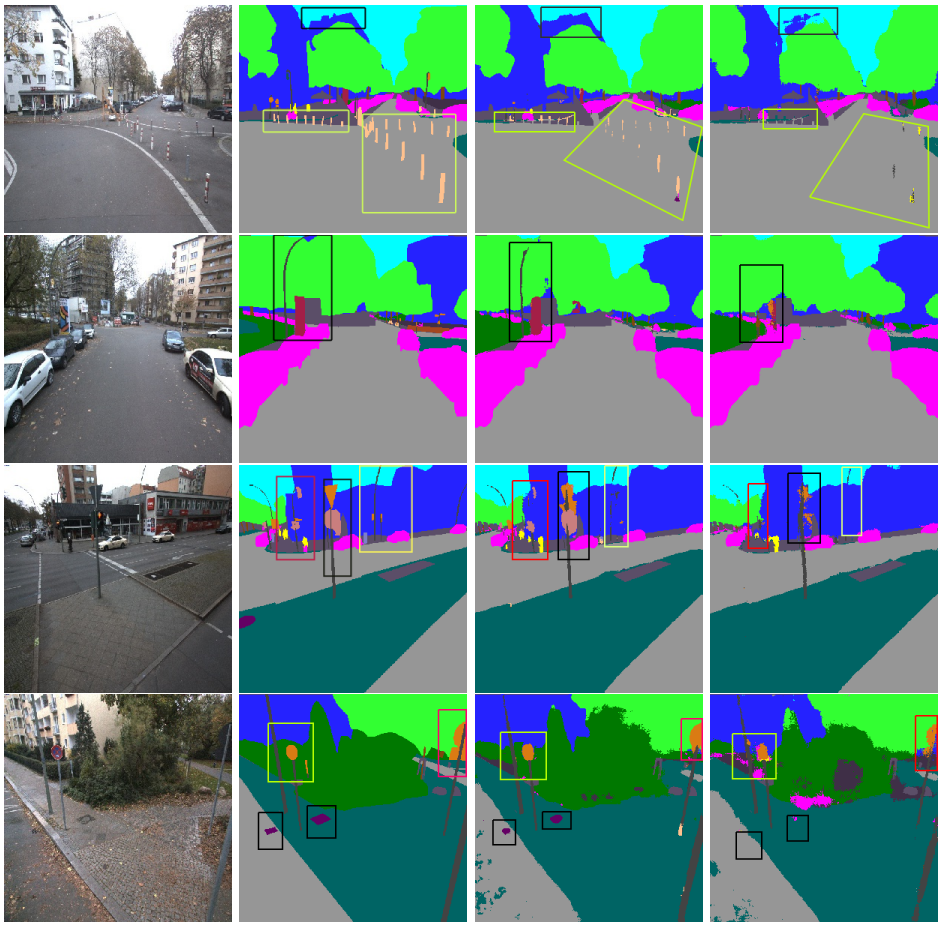


Figure 58: Qualitative comparison between ResNet-34 and DDNet. First column holds images, second holds original mask, third and fourth column represents results from ResNet-34 and DDNet respectively.

From the figure above, it is clear that the DDNet is not a suitable model for the Berlin set. It's capacity to segment challenging classes that are small and has a lesser frequency of occurrence is extremely poor. Moreover, the model produces a lot of artifacts and the region boundaries for couple of major classes like *trees, pavement, building* etc are not well defined.

In an attempt to find the source of the problem, we decided to swap the encoders and decoders of both the architectures that can ease locate the drawback of the DDNet that is responsible for the deteriorating the quality of segmentation.

5.5.4 DPDB-UNet vs Res-DDNet

We come up with two architectures where we club the DDNet encoder (DPDB encoder) to the UNet decoder in short *DPDB-UNet* to make one model and ResNet-34 encoder to DDNet decoder to make another model in short *Res-DDNet*. This can provide an insight to the independent functioning and performance of the encoders and decoders of the DDNet.

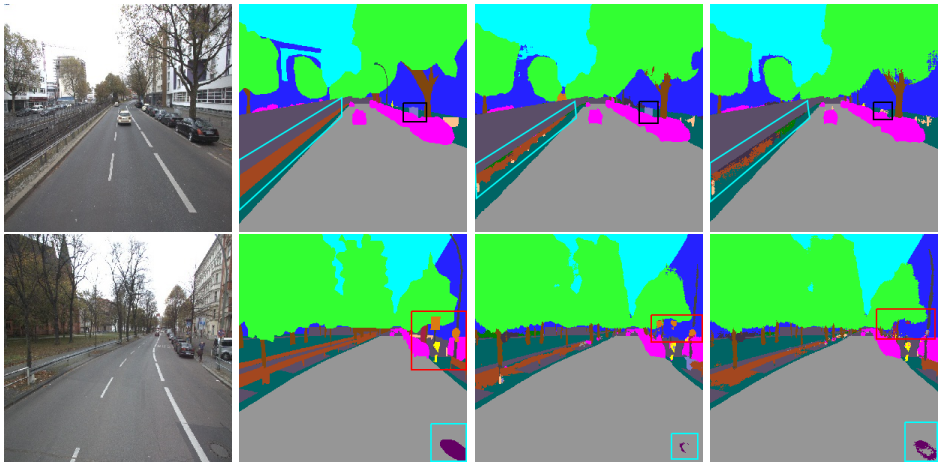
5.5.4.1 Quantitative analysis of DPDB-UNet and Res-DDNet

The table below illustrates the quantitative differences between the two models. The DPDB-UNet is smaller in size (less parameters) compared to the Res-DDNet but both the models shows similar performances with slight variations.

DPDB-UNet vs Res-DDNet				
Input type	**Parameters**	**Global Acc**	**Mean Acc**	**Mean IoU**
DPDB-UNet	72,576,904	93.55	48.21	42.30
Res-DDNet	107,134,716	**93.86**	**48.73**	**42.45**

Table 17: Performance comparison of DPDB-UNet and Res-DDNet

5.5.4.2 Qualitative analysis of DPDB-UNet and Res-DDNet



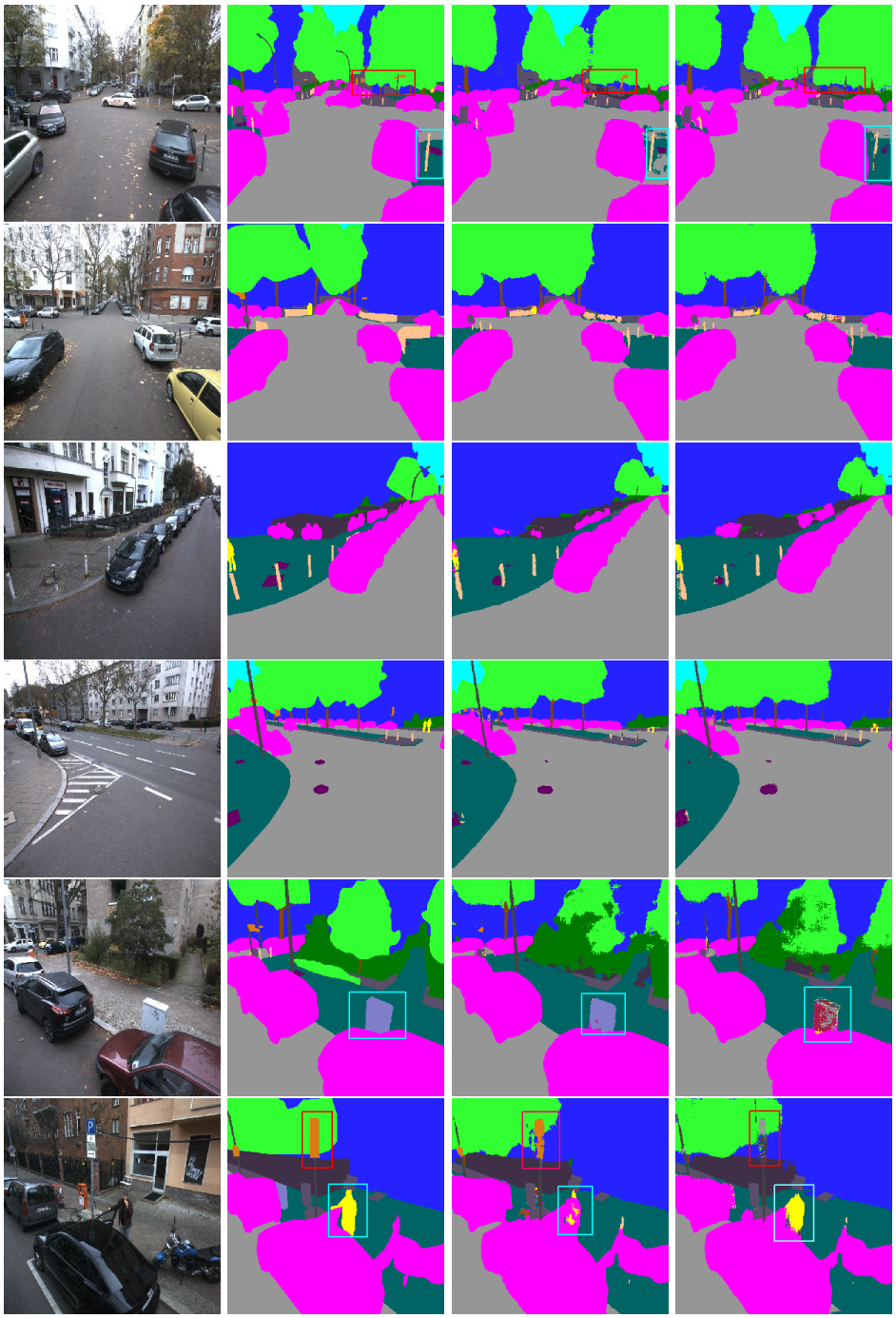


Figure 59: Qualitative comparison between DPDB-UNet and Res-DDNet. First column holds images, second holds original mask, third and fourth column represents results from DPDB-UNet and Res-DDNet respectively.

The figure above illustrates the qualitative differences between both the models. The performance of the models look similar with very slight variations as both the models exhibit different strengths in segmenting different classes. If we look at the 2nd, 3rd and 8th row, the DPDB-UNet is able to segment the class *street sign* better than the Res-DDNet but the score for the class *hole cover* goes to the Res-DDNet in the 2nd row. The class *encasement of equipment* is better segmented by DPDB-UNet as highlighted in the figure at the 1st and the 7th row but the class *people* is better segmented by Res-DDNet as shown in the 8th row.

Looking into both the models, we can see that both of them outperforms the DDNet again by almost a margin of around $8 - 9\%$ in both IoU and accuracy. Hence, from this, we can conclude that one of the factors that could have been responsible for the DDNet's poor performance can be the depth of the network as it is way deeper than it's modified counterparts. Moreover, both DPDB-UNet and Res-DDNet has similar performance on the Berlin set. Hence both the encoder and decoder of the DDNet in itself, when used separately does not reveal any weakness but as a combination (DDNet) may not be a right fit to learn the Berlin set.

To continue with the experiments, we decided to proceed with the DPDB-UNet as it achieves the same level of performance to that of the Res-DDNet with lesser parameters.

5.5.5 DPDB-UNet variations

We made some modifications in this architecture by decreasing the network depth in the 3rd branch of the encoder to have only 6 blocks instead of 9. The rest of the variations were made to the width of the network by widening the residual blocks or the dense blocks or both. Let us refer to these variations as *DPDB-UNet-1*, *DPDB-UNet-2*, *DPDB-UNet-3*, *DPDB-UNet-4* and *DPDB-UNet-5*.

5.5.5.1 DPDB-UNet-1

Here, we widen the output feature channels in residual branches from $(128, 256, 512, 1024)$ to have $(224, 448, 896, 1792)$ channels in the four branches respectively. The decoder is widen to have $(896, 448, 224)$ channels in the three branches respectively. The dense blocks are untouched.

5.5.5.2 DPDB-UNet-2

Here, we double the output feature channels in residual branches from $(128, 256, 512, 1024)$ to have $(256, 512, 1024, 2048)$ channels in the four branches respectively. The decoder is widen to have $(1024, 512, 256)$ channels in the three branches respectively. The dense blocks are untouched.

5.5.5.3 DPDB-UNet-3

Here, we widen the output feature channels in dense branches from $(96, 112, 288, 144)$ to have $(640, 1536, 4096, 5120)$ channels in the four branches respectively. The decoder and the residual blocks are untouched.

5.5.5.4 DPDB-UNet-4

Here, we widen the output feature channels of both the dense and the residual branches. The residual branches are widened from $(128, 256, 512, 1024)$ to have $(192, 384, 768, 1536)$ channels and dense branches are widened from $(96, 112, 288, 144)$ to have $(240, 576, 1536, 1920)$ feature channels. The decoder is widened to accommodate $(768, 384, 192)$ feature channels.

5.5.5.5 DPDB-UNet-5

Here again, we widen the output feature channels of both the dense and the residual branches. The residual branches are widened from $(128, 256, 512, 1024)$ to have $(224, 448, 896, 1792)$ channels and dense branches are widened from $(96, 112, 288, 144)$ to have $(280, 672, 1792, 2240)$ feature channels. The decoder is widened to accommodate $(896, 448, 224)$ feature channels.

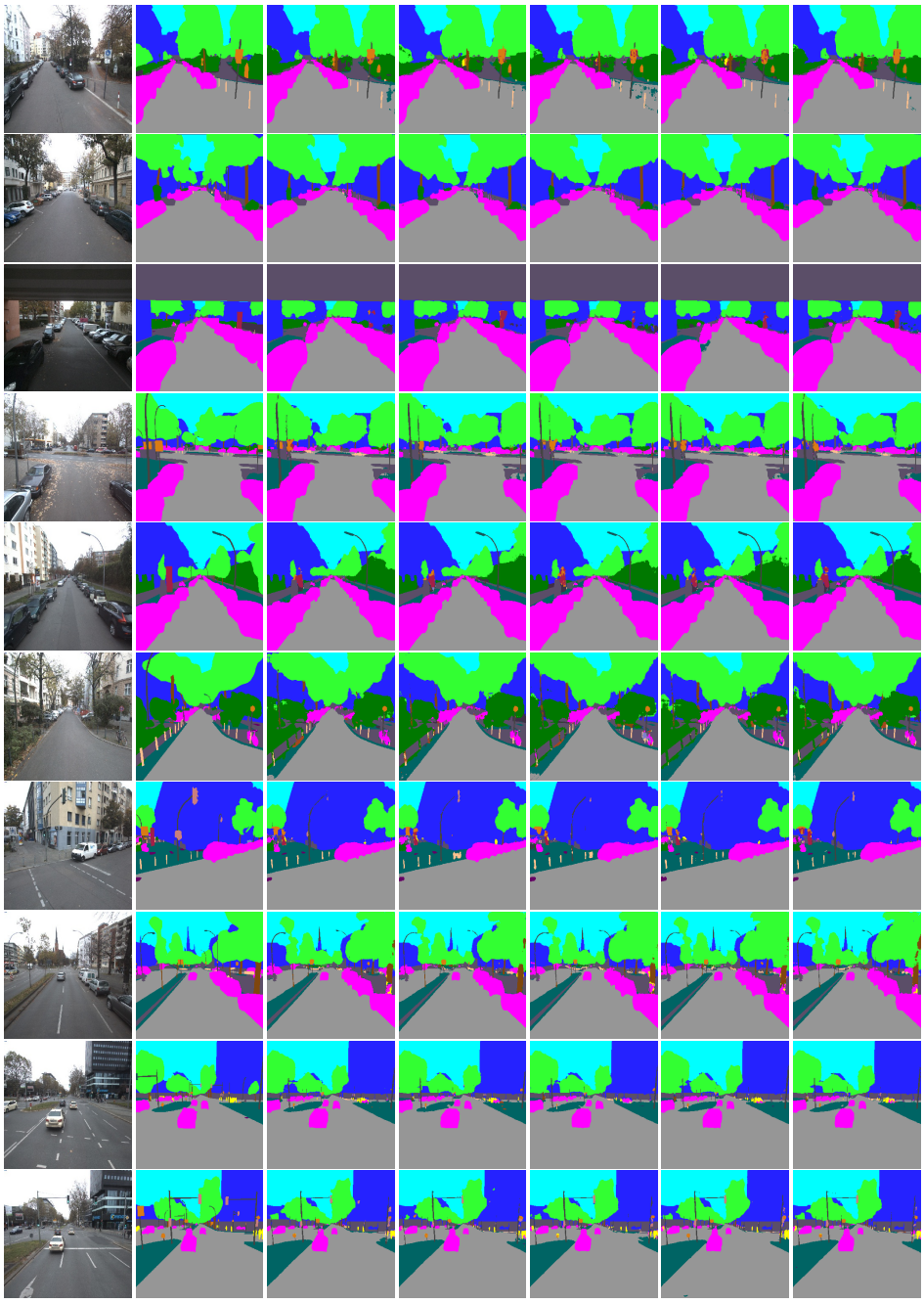
5.5.5.6 Quantitative analysis of DPDB-UNet variations

The table below illustrates the quantitative differences between the five models. The models vary in the number of learnable parameters. The depth of the architecture remains the same with varying width.

Evaluation of DPDB-UNet variations				
Input type	**Parameters**	**Global Acc**	**Mean Acc**	**Mean IoU**
DPDB-UNet-1	104,915,600	**93.96**	**51.10**	**44.98**
DPDB-UNet-2	198,590,576	93.95	50.86	44.70
DPDB-UNet-3	49,870,112	93.63	49.02	42.99
DPDB-UNet-4	84,087,872	93.85	50.59	44.44
DPDB-UNet-5	112,799,792	93.93	51.07	44.93

Table 18: Quantitative evaluation of DPDB-UNet models

5.5.5.7 Qualitative analysis of DPDB-UNet variations



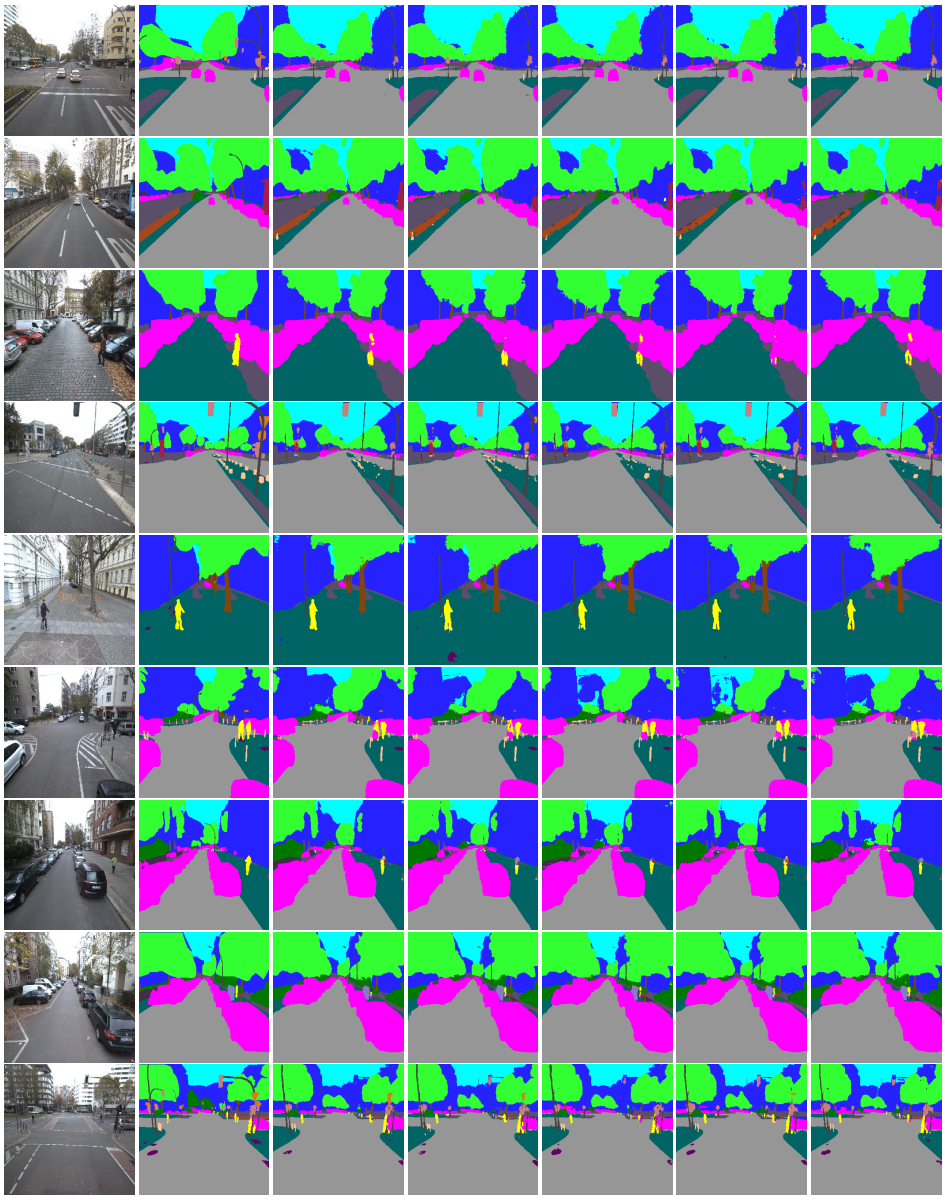


Figure 60: Qualitative visualization of the variations of DPDB-UNet on test set. The first column contain the images, the second column are the original masks, the third holds prediction masks from DPDB-UNet-1, the fourth column holds prediction masks from DPDB-UNet-2, the fifth column refers to DPDB-UNet-3, the sixth column has prediction masks from DPDB-UNet-4 and the last column has prediction masks from DPDB-UNet-5.

From the figure above, we can see that the models have very slight variation in quality with the variation in parameters. This can also be verified from their quantitative results. Couple of examples we can consider here are like in row 1 we can see slight artifacts produced by DPDB-UNet-1, DPDB-UNet-3 and DPDB-UNet-5 whereas DPDB-UNet-2 and DPDB-UNet-4 confuses the class *street sign* to be class *people*. In row 3 the class *encasement of equipment* is better classified using DPDB-UNet-2 and DPDB-UNet-5, in row 7 DPDB-UNet-2 amd DPDB-UNet-5 produces a better quality for the class *traffic light* but the class *bollards* is better segmented using DPDB-UNet-1 and DPDB-UNet-4. In row 8, DPDB-UNet-3 achieves better quality segmentation for class *enclosure* but again DPDB-UNet-2 segments the class *tree trunk* comparatively better than the rest of the models. In row 9, DPDB-UNet-3 produces poor quality segmentation for the class *people* etc. The list of examples can go on. The performance of DPDB-UNet-5 is similar to that of DPDB-UNet-1 which is also reflected in their quantitative scores. Considering overall performance, for all the five models, we come to a neutral conclusion as they are independently strong in different classes.

With the increase in parameters, we do see a $2-3\%$ rise in accuracy and almost 2% rise in IoU when compared to the original DPDB-UNet but among it's variations, the improvements differs by small margins until it hits a maximum as illustrated by DPDB-UNet-1 in table 18 and starts dropping as showcased by DPDB-UNet-5 and DPDB-UNet-2 in the same table which has approximately 1.1M and $2M$ parameters respectively. This might imply that the performance might hit a saturation point and increasing the parameters will not yield any significant performance gain. Any modifications to the residual connections or the dense connections has a marginal effect on the performance scores. This can be caused due to lesser variability in the Berlin set which in turn implies that the learning is more dependent on the capacity of the model rather than it's topology.

However, we achieve approximately equivalent performance using DPDB-UNet-3 which is widely based on dense connections and consumes $3-4$ times lesser parameters than the rest of the above four models. Hence, we can also argue that the residual blocks can also be replaced with dense blocks as they are more parameter efficient and performs equally well.

Both the variations of ResNet-34 still has a better score than any of the DDNet models. It might be possible that a small model with residual connections might suffice for the Berlin set as we have seen in the above experiments that the ResNet-34 is comparatively thinner and shallower than the DDNet or any of the above models.

In the figure below, we illustrate the performance change of the models in terms of their quantitative metrics with the change in parameters. We can see how the performance of the networks increases with increase in parameters until it hits a maximum and starts dropping with further parameter increase.

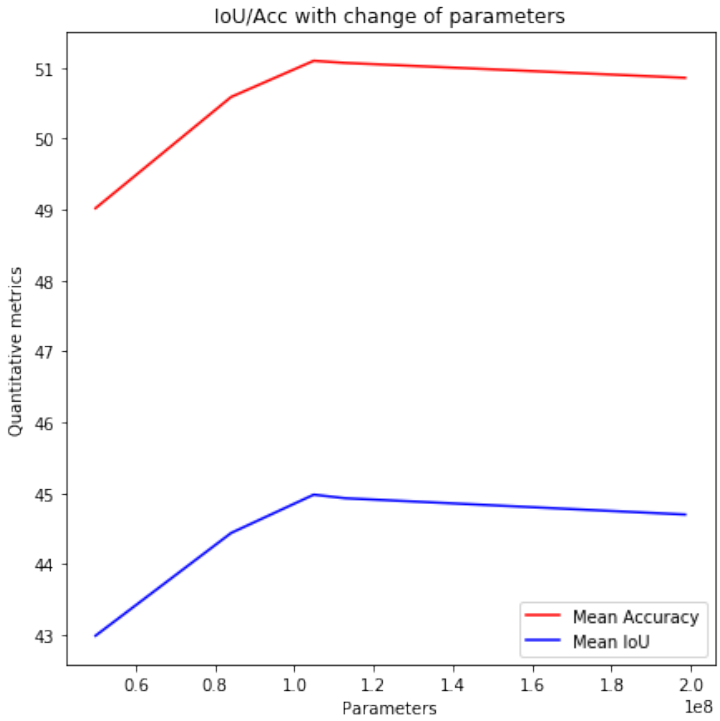


Figure 61: Performance comparison of the models with change in parameters

From the set of experiments above, we also decided to conduct few experiments using DPDB-UNet-1 as it had the highest quantitative scores. These experiments were carried out by changing the size of the grouped convolutions. All the models used so far had a group size of 1. The later experiments were conducted with group size of $4, 8$ and 16. But for the models with altered group size and the given dataset we did not see any significant gain in the quality of segmentation, though increasing the group size improved computational efficiency of the models. Never the less, it might be premature to make a performance comparison of two different architectures using a single dataset and arrive at a definitive conclusion as diverse data can introduce diverse features to learn which can in return reflect the strengths and weaknesses of various architectural topologies. Our dataset is car-centric with constant feature representations as major classes occupy almost similar macro regions in the images. This can be a vital factor why models with varying architectural modifications does not contribute to much quality changes as in theory these models seems to be overfitting. Hence the Berlin set may not be a suitable dataset for making design choices for models.

6 Conclusion

Segmentation based on depth can bring significant advantages in classifying challenging classes that the model fails to learn only from RGB data. Depending on the creation and processing of depth information, good depth maps can bring out better quality segmentation where as depth maps with artifacts can also impede the performance of the model. Depth as a whole can prevent a model from misclassifying classes that exhibit similar appearances in the images as early fusion of depth helps the model learn the structural variances in different classes.

Late fusion (Fusenet) based on *element-wise summation* did not yield any promising results. Independent depth processing is an unproductive approach where the model fails to exploit the most crucial benefit of adding the depth component. Different classes blended together can easily be misclassified and labelled as one (particularly the dominant class). Fusion via *element-wise summation* can also introduce artifacts in the segmentation due to incompatible feature representation as depth and RGB inputs are processed independently, although fusion via *concatenation* still can achieve comparatively better results.

Comparison study of our model to the DDNet proved that our architecture (ResNet-34) based on residual connections proved efficient in bringing out good quality segmentation on the Berlin dataset. A shallow and wide model could be an architecture choice for the current set of data.

7 Acknowledgements

This project has been a collective effort of several individuals that made a unique contribution in bringing out a significant value in the work. I am very grateful to my Prof. Dr. Thomas Brox for his motivation and guidance that helped me direct the project to achieve a qualitative goal. I would also like to thank Prof. Dr. Alexander Reiterer for his patience and understanding about my situation and work and his generous support in providing the necessary essentials that helped me complete the project. My supervisors Christian Koch and Dominik Störk have been extremely helpful in providing me with ideas and sincerely attended to any minor or major help during my tenure. I am also thankful to PhD Student Gabriel Leivas Oliveira for his detail explanation and analysis of few topics that helped me grasp the ideas behind them. At last but not the least I am grateful to have the emotional and mental support of my family and my girlfriend Mirjam Störk and their patience until the very end of the project.

References

[1] DeepLearning 0.1.
Fully Convolutional Networks (FCN) for 2D segmentation.
URL: http://deeplearning.net/tutorial/fcn_2D_segm.html.

[2] Algorithmia. *Introduction to Unsupervised Learning.* Apr. 2018.
URL: https://blog.algorithmia.com/introduction-to-unsupervised-learning/.

[3] andyandy.
What are the effects of depth and width in deep neural networks?
May 2016.
URL: https://stats.stackexchange.com/questions/214360/what-are-the-effects-of-depth-and-width-in-deep-neural-networks (visited on 10/18/2018).

[4] Sidath Asiri. *Machine Learning Classifiers.*
URL: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623.

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla.
"Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *arXiv preprint arXiv:1511.00561* (2015).

[6] Suvro Banerjee. *An Introduction to Recurrent Neural Networks.*
URL: https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912 (visited on 10/18/2018).

[7] R Bertolami et al. "A Novel Connectionist System for Improved Unconstrained Handwriting Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.5 (2009).

[8] Jason Brownlee.
Difference Between Classification and Regression in Machine Learning.
Dec. 2017.
URL: https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/.

[9] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.* July 2017.
URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[10] Jason Brownlee.
Supervised and Unsupervised Machine Learning Algorithms. Mar. 2016.
URL: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[11] Peter J Burt and Edward H Adelson.
"The Laplacian pyramid as a compact image code".
In: *Readings in Computer Vision.* Elsevier, 1987, pp. 671–679.

[12] Liang-Chieh Chen et al.
"DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs".
In: *arXiv:1606.00915* (2016).

[13] Davide Chicco, Peter Sadowski, and Pierre Baldi. "Deep autoencoder neural networks for gene ontology annotation predictions".
In: *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM. 2014, pp. 533–540.

[14] Camille Couprie et al.
"Indoor semantic segmentation using depth information".
In: *arXiv preprint arXiv:1301.3572* (2013).

[15] Paras Dahal.
Classification and Loss Evaluation - Softmax and Cross Entropy Loss.
May 2017. URL: https://deepnotes.io/softmax-crossentropy.

[16] David Dao. *deconvolution*. June 2016. URL: https://datascience.stackexchange.com/questions/6107/what-are-deconvolutional-layers (visited on 10/02/2018).

[17] Yujun Deng. *Why does batch normalization help?* Oct. 2017.
URL: https://www.quora.com/Why-does-batch-normalization-help.

[18] Arden Dertat. *Convolution*. Nov. 2017.
URL: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2 (visited on 10/02/2018).

[19] Adit Deshpande. *filters*.
URL: https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/ (visited on 10/02/2018).

[20] Michael Dietz. *Basic vs Bottleneck*. May 2017.
URL: https://blog.waya.ai/deep-residual-learning-9610bb62c355 (visited on 10/02/2018).

[21] Niklas Donges. *Gradient Descent in a Nutshell*. Mar. 2015.
URL: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0 (visited on 11/18/2018).

[22] Luke Dormehl.
What is an artificial neural network? Here's everything you need to know.
Sept. 2018. URL: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/.

[23] Arthur Douillard. *3 Small But Powerful Convolutional Networks*.
URL: https://towardsdatascience.com/3-small-but-powerful-convolutional-networks-27ef86faa42d.

[24] Daniel Faggella. *What is Machine Learning?* Sept. 2016. URL: https://emerj.com/ai-glossary-terms/what-is-machine-learning/.

[25] Vincent Fung. *An Overview of ResNet and its Variants*. July 2017. URL: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035 (visited on 10/02/2018).

[26] Hao Gao. *The Efficiency of Densenet*. Aug. 2017. URL: https://medium.com/@smallfishbigsea/densenet-2b0889854a92.

[27] Dishashree Gupta. *Fundamentals of Deep Learning – Activation Functions and When to Use Them?* URL: https://towardsdatascience.com/unsupervised-learning-with-python-173c51dc7f03.

[28] Caner Hazirbas et al. "Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture". In: *Asian Conference on Computer Vision*. Springer. 2016, pp. 213–228.

[29] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[30] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[31] Xuming He, Richard S Zemel, and Debajyoti Ray. "Learning and incorporating top-down cues in image segmentation". In: *European conference on computer vision*. Springer. 2006, pp. 338–351.

[32] Gao Huang et al. "Densely Connected Convolutional Networks." In: *CVPR*. Vol. 1. 2. 2017, p. 3.

[33] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[34] Fraunhofer IPM. *CPS Scanner mounted on the vehicle*. URL: https://www.ipm.fraunhofer.de/en.html.

[35] Shruti Jadon. *Activation Functions*. Mar. 2016. URL: https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092 (visited on 10/02/2018).

[36] Vishakha Jha. *Machine Learning*. July 2017. URL: https://www.techleer.com/articles/203-machine-learning-algorithm-backbone-of-emerging-technologies/ (visited on 10/02/2018).

[37] Andrej Karpathy. "Cs231n convolutional neural networks for visual recognition". In: *Neural networks* 1 (2016).

[38] Illarion Khlestov. *Convolutions Types*. July 2017.
URL: https://ikhlestov.github.io/pages/machine-learning/convolutions-types/#grouped-convolutions.

[39] Jack Kiefer, Jacob Wolfowitz, et al.
"Stochastic estimation of the maximum of a regression function".
In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 462–466.

[40] Diederik P Kingma and Jimmy Ba.
"Adam: A method for stochastic optimization".
In: *arXiv preprint arXiv:1412.6980* (2014).

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

[42] Vihar Kurama. *Unsupervised Learning with Python*.
URL: https://towardsdatascience.com/unsupervised-learning-with-python-173c51dc7f03.

[43] Jesse Levinson and Sebastian Thrun.
"Automatic Online Calibration of Cameras and Lasers."
In: *Robotics: Science and Systems*. Vol. 2. 2013.

[44] Xiangang Li and Xihong Wu.
"Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition".
In: *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE. 2015, pp. 4520–4524.

[45] 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*. 2015, pp. 3431–3440.

[46] AZ Makhmudov and SS Abdukarimov.
"Speech Recognition using deep learning algorithms". In: *: , ,* 2016, pp. 10–15.

[47] Assaad MOAWAD. *Neural networks and back-propagation*. Feb. 2018.
URL: https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e
(visited on 10/02/2018).

[48] Gabriel Leivas Oliveira, Wolfram Burgard, and Thomas Brox.
"DPDB-Net: Exploiting Dense Connections for Convolutional Encoders".
In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[49] Xiaofeng Ren and Jitendra Malik.
"Learning a classification model for segmentation". In: *null*. IEEE. 2003, p. 10.

[50] Herbert Robbins and Sutton Monro.
"A stochastic approximation method".
In: *Herbert Robbins Selected Papers*. Springer, 1985, pp. 102–109.

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

[52] Margaret Rouse. *machine learning (ML)*. URL: `https://searchenterpriseai.techtarget.com/definition/machine-learning-ML`.

[53] skymind Rouse. *A Beginner's Guide to Deep Reinforcement Learning*.
URL: `https://skymind.ai/wiki/deep-reinforcement-learning`.

[54] Stuart J Russell and Peter Norvig.
Artificial intelligence: a modern approach.
Malaysia; Pearson Education Limited, 2016.

[55] Haşim Sak, Andrew Senior, and Françoise Beaufays.
"Long short-term memory recurrent neural network architectures for large scale acoustic modeling". In: *Fifteenth annual conference of the international speech communication association*. 2014.

[56] Ausif Mahmood Saleh Albelwi. *CNN architecture*. May 2017. URL:
`https://www.mdpi.com/1099-4300/19/6/242` (visited on 10/02/2018).

[57] Michael Wilber Sam Gross. *A standard Residual Block*. Feb. 2016.
URL: `http://torch.ch/blog/2016/02/04/resnets.html` (visited on 10/02/2018).

[58] Madhu Sanjeevi. *Different types of Machine learning and their types*.
Sept. 2017. URL:
`https://medium.com/deep-math-machine-learning-ai/different-types-of-machine-learning-and-their-types-34760b9128a2`
(visited on 10/18/2018).

[59] Jürgen Schmidhuber. "Deep learning in neural networks: An overview".
In: *Neural networks* 61 (2015), pp. 85–117.

[60] Shai. *What is "semantic segmentation" compared to "segmentation" and "scene labeling"?* May 2016. URL: `https://stackoverflow.com/questions/33947823/what-is-semantic-segmentation-compared-to-segmentation-and-scene-labeling`.

[61] SAGAR SHARMA. *relu*. Sept. 2017.
URL: `https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6` (visited on 10/02/2018).

[62] John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. 2001.

[63] Jianbo Shi and Jitendra Malik.
"Normalized cuts and image segmentation". In: *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), pp. 888–905.

[64] Mandeep Sidana. *Types of classification algorithms in Machine Learning.* Feb. 2017. URL: https://medium.com/@sifium/machine-learning-types-of-classification-9497bd4f2e14.

[65] Nathan Silberman et al.
"Indoor segmentation and support inference from rgbd images".
In: *European Conference on Computer Vision*. Springer. 2012, pp. 746–760.

[66] David Silver et al.
"Mastering the game of Go without human knowledge".
In: *Nature* 550.7676 (2017), p. 354.

[67] skymind. *DA Beginner's Guide to Neural Networks and Deep Learning.* URL: https://skymind.ai/wiki/neural-network#define (visited on 10/02/2018).

[68] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao.
"Sun rgb-d: A rgb-d scene understanding benchmark suite".
In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 567–576.

[69] Danny Sullivan. *How Machine Learning Works, As Explained By Google.* Nov. 2015. URL: https://martechtoday.com/how-machine-learning-works-150366 (visited on 11/18/2018).

[70] Alex Teichman, Jesse Levinson, and Sebastian Thrun. "Towards 3D object recognition via classification of arbitrary object tracks".
In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 4034–4041.

[71] u/idg101. *How does DenseNet compare to ResNet and Inception?* URL: https://www.reddit.com/r/MachineLearning/comments/67fds7/d_how_does_densenet_compare_to_resnet_and/.

[72] user20160.
How does the Adam method of stochastic gradient descent work? Aug. 2017.
URL: https://stats.stackexchange.com/questions/220494/how-does-the-adam-method-of-stochastic-gradient-descent-work.

[73] Avinash Sharma V.
Understanding Activation Functions in Neural Networks. Mar. 2017.
URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[74] Irene Walde et al. "Graph-based mapping of urban structure types from high-resolution satellite image objects—Case study of the German cities Rostock and Erfurt". In: *IEEE Geoscience and Remote Sensing Letters* 10.4 (2013), pp. 932–936.

[75] Anish Singh Walia. *Activation functions and it's types-Which is better?* May 2017. URL: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f.

[76] Wikipedia contributors. *Automotive industry — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/w/index.php?title=Automotive_industry&oldid=870415653. [Online; accessed 27-November-2018]. 2018.

[77] Wikipedia contributors. *Bioinformatics — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/w/index.php?title=Bioinformatics&oldid=870192795. [Online; accessed 4-December-2018]. 2018.

[78] Wikipedia contributors. *Construction — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/w/index.php?title=Construction&oldid=870363069. [Online; accessed 27-November-2018]. 2018.

[79] Wikipedia contributors. *Feedforward neural network — Wikipedia, The Free Encyclopedia.* [Online; accessed 26-October-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Feedforward_neural_network&oldid=856029366.

[80] Wikipedia contributors. *Hinge loss — Wikipedia, The Free Encyclopedia.* [Online; accessed 25-October-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Hinge_loss&oldid=864504306.

[81] Wikipedia contributors. *Inertial measurement unit — Wikipedia, The Free Encyclopedia.* [Online; accessed 31-October-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Inertial_measurement_unit&oldid=864934959.

[82] Wikipedia contributors. *Machine learning — Wikipedia, The Free Encyclopedia.* [Online; accessed 19-October-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=864673085.

[83] Wikipedia contributors. *Mean absolute error — Wikipedia, The Free Encyclopedia.* [Online; accessed 25-October-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Mean_absolute_error&oldid=859130031.

[84] Wikipedia contributors.
Mean squared error — Wikipedia, The Free Encyclopedia.
[Online; accessed 25-October-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=857685443.

[85] Wikipedia contributors.
Rectifier (neural networks) — Wikipedia, The Free Encyclopedia.
https://en.wikipedia.org/w/index.php?title=Rectifier_(neural_networks)&oldid=869532192.
[Online; accessed 3-December-2018]. 2018.

[86] Wikipedia contributors.
Remote sensing — Wikipedia, The Free Encyclopedia.
https://en.wikipedia.org/w/index.php?title=Remote_sensing&oldid=868174959. [Online; accessed 27-November-2018]. 2018.

[87] Wikipedia contributors.
Space Imaging — Wikipedia, The Free Encyclopedia.
https://en.wikipedia.org/w/index.php?title=Space_Imaging&oldid=822158753. [Online; accessed 27-November-2018]. 2018.

[88] Wikipedia contributors.
Telecommunication — Wikipedia, The Free Encyclopedia.
https://en.wikipedia.org/w/index.php?title=Telecommunication&oldid=869881298.
[Online; accessed 27-November-2018]. 2018.

[89] Daniel J Withey and Zoltan J Koles. "A review of medical image segmentation: methods and available software". In: *International Journal of Bioelectromagnetism* 10.3 (2008), pp. 125–148.

[90] How It Works. "Roberts Cross Edge Detector". In: ().

[91] Fisher Yu and Vladlen Koltun.
"Multi-Scale Context Aggregation by Dilated Convolutions". In: *ICLR*. 2016.

[92] Sergey Zagoruyko and Nikos Komodakis. "Wide residual networks".
In: *arXiv preprint arXiv:1605.07146* (2016).

[93] Andreas Zell. *Simulation neuronaler netze*. Vol. 1.
Addison-Wesley Bonn, 1994.

[94] Yu-Jin Zhang. "Image segmentation in the last 40 years".
In: *Encyclopedia of Information Science and Technology, Second Edition*.
IGI Global, 2009, pp. 1818–1823.