Master's Thesis

CloudSeg: Semantic Segmentation of 3D Point Clouds using Deep Learning

---

Farooq Ahmed Zuberi

Examiners: Prof. Thomas Brox,
Dr. Joschka Boedecker

Adviser: Dr. Claudius Glaeser
(Robert Bosch GmbH)

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Machine Learning

March 22nd, 2017

Writing period

23. 09. 2016 – 23. 03. 2017

Examiner

Prof. Thomas Brox, Dr. Joschka Boedecker

Advisers

Dr. Claudius Glaeser (Robert Bosch GmbH)

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

3D LiDAR (Light Detection And Ranging) sensors have been widely used in numerous robotics platforms, and their usage in autonomous driving and advance driver assistance systems (ADAS) is becoming more and more common. Perception in LiDAR sensor data still comprise of classical machine learning pipeline, despite leaps of technical advances in perception methods through the rise of Deep Learning. As Deep learning is transforming the fields of computer vision and natural language processing, a similar breakthrough has not been seen with the methods of perception in 3D point cloud data. This lack of progress is due to the computational burden of 3rd spatial dimension and unavailability of adequate point wise annotated benchmark datasets.

This thesis aims to close this gap, by performing the semantic segmentation of 3D point cloud data from LiDAR sensors using deep learning. In order to train end-to-end deep neural networks, we present a dataset of 3D point clouds from 64 channel LiDAR sensor with point-wise annotations in urban environment. We transform our 3D point cloud data to 2D depth maps to release the computational burden of handling 3rd spatial dimensions. It aids the application of conventional deep convolutional neural networks with 3D point cloud data while keeping the storage complexity in check. Moreover, we design a deep convolutional neural network, CloudSeg, that performs novel 11 class point-wise semantic point cloud segmentation. To the best of our knowledge, this thesis is the first study to perform involved 11 class semantic segmentation of 3D point clouds in urban environment. CloudSeg shows promising quantitative and qualitative results for point-wise semantic point cloud segmentation in urban environment, reliably explaining around 85% of the point cloud in each LiDAR scan. Moreover, we show that our convolutional neural network model can detect large variety of diverse objects using only the data from LiDAR sensors.

Zusammenfassung

Deep Learning revolutioniert die Bereiche Computer Vision und natürliche Sprachverarbeitung. Dieses Verfahren ist noch nicht für die Anwendungen mit 3D-Daten weit verbreitet. Dieser Mangel an Fortschritt ist auf die zusätzlich benötigte Rechenkraft, die die dritte räumlichen Dimension der Daten verursacht, sowie den Mangel an adäquaten, punktweise annotierten Benchmark-Datensätzen zurückzuführen. Ziel dieser Arbeit ist es, die semantische Segmentierung der 3D-Point-Cloud-Daten von LiDAR Sensoren mit Deep Learning zu evaluieren. Zu diesem Zweck werden die konkrete Literaturrecherche des Themas durchgeführt, und einen Datensatz für das End-to-End-Training von Deep Networks auf LiDAR-Point-Cloud hergestellt und dementsprechend präsentiert. Darüber hinaus wird CloudSeg, ein neuronales Faltungsnetzwerk, das eine neuartige Point-wise semantische Segmentierung, basierend auf Convolution Neural Network, mit 11 Klassen entwickelt und präsentiert. CloudSeg zeigt hervorragende Ergebnisse, sowohl qualitative als auch quantitative Analyse für die Point-wise semantische Segmentierung im urbanen Umfeld. Darüber hinaus kann das Faltungsnetzwerk-Modell unterschiedliche Objekte mit unterschiedlichen Eigenschaften, nur mithilfe von LiDAR-Sensor-Daten, detektieren.

Contents

List of Figures

List of Tables

1. Introduction

The cognition in humans can be defined as a process of acquiring knowledge and understanding through thought, experience, and the senses. The realization of such cognitive abilities in technical systems falls under the umbrella of Artificial Intelligence. It encompasses the capabilities such as learning, reasoning, planning, executing actions, language processing and perception in agents like robots, autonomous cars or simple software.

Perception is the organization, identification, and interpretation of the sensory information in order to represent and understand the environment [1]. Similarly, machine perception is the capability of a computer system to interpret data in a manner that is similar to the way humans use their senses to relate to the world around them [2]. Perception plays a pressing role in the application of autonomous driving and advance driver assistance systems (ADAS). When a person drives a car, he can instantly perceive the environment at a glance. It is imperative for Autonomous driving and ADAS systems to know what are the objects on the road? What are the different lanes on the road? Is the obstacle in front of the car a human or a anything else? Developing a perception system, that can completely understand the scene in micro seconds is a challenging task and therefore an active area of research in computer vision and artificial intelligence communities.

The perception in autonomous driving and ADAS systems is carried out through the sensors mounted on the car, mostly cameras and ranging sensors. The camera images provide a very dense information of the environment and the objects in it, on the other hand the ranging sensors, like sonar, radar and laser, give accurate range measurements of the environment. Throughout years, scientists have been using machine learning algorithms to make sense of the environment using these camera and ranging sensors. The rise of a family of machine learning algorithms called Deep Learning have boosted the success of perception and scene understanding in camera images. Deep neural networks (DNN) have some distinct properties which gives it an advantage over other machine learning algorithms. This resulted in increased interest from the researchers and the industry and unprecedented success in many problems

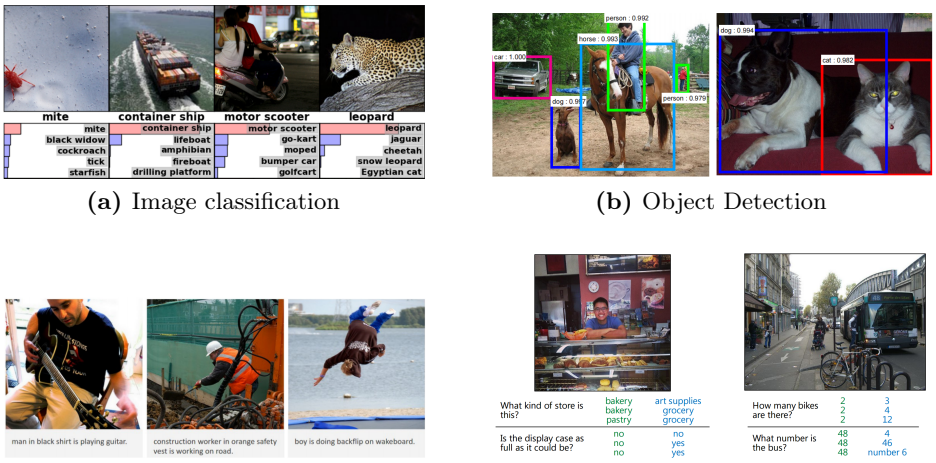
that previously used classical machine learning. One such task is semantic image segmentation, a perception problem, where DNNs identify and separate different objects presented in the camera image.

On the other hand, perception in ranging sensors still uses classical machine learning algorithm pipeline. The engineering of such pipelines is very tedious and the perception is not as informative as the semantic image segmentation using DNN for cameras. The application of DNNs for the perception of range sensors, especially Light Detection And Ranging (LiDAR) sensor, is not explored as much as some other topics. This thesis shows that deep learning (DNN) can successfully be applied in LiDAR range sensor to produce informative perception. This chapter presents motivation, purpose, delimitations and methodology of this thesis study.

1.1. Motivation

Deep learning algorithms have changed the landscape of artificial intelligence and revolutionized the fields of computer vision, natural language processing and machine learning in general [3]. They have proved to be far superior than preceding approaches in object detection, image classification, speech recognition and speech generation and have led to exploration of more advanced research topics. They have successfully shown human level efficiency in many classical computer vision (see Figure 1) and speech recognition tasks while also beating human experts in the game of Go [4]. This widespread success of machine learning has lead to a wider acceptance of AI based technologies in general applications. Google, Facebook, Microsoft and Apple, all use deep learning algorithms in their products, that are used by millions of users on daily basis [3]. Acceptance and integration of AI technology has also increased the demand and development more intelligent commercial machines especially self-driving autonomous vehicles.

Semantic Image Segmentation is another task in which deep learning has shown incredible promise. The aim of semantic image segmentation is to partition an image into semantically meaningful parts, and to classify each part into one of the predetermined classes. Many automatic intelligent systems relies on the semantic image segmentation for perception especially autonomous driving vehicles in Urban environments. Because of the early success of deep learning and wide variety of use-cases, semantic image segmentation has become a very active research problem. A number of benchmarks exist for its evaluation and their leader-boards are updated in the matter of week.



(a) Image classification **(b)** Object Detection

(c) Image Captioning **(d)** Question Answering from Images

Figure 1.: Success application of deep learning in Computer Vision Tasks. This figure shows the examples from success use deep learning for classical computer vision tasks like image classification (a), Object detection (b), Image captioning (c) and Question answering from images (d). Courtesy [5].

Classification of 3D point clouds is a similar perception problem like semantic image segmentation. 3D point clouds are an effective utility to represent the 3D structure of the environment. Usually obtained through a ranging sensor like LiDAR. They are integral part of perception in autonomous systems especially self-driving cars. The usual methods for understanding 3D point clouds consist of the three step pipeline comprising segmentation, feature extraction and classification. Segmentation separates the foreground and the background and cluster individual objects together, feature extraction step involves the tedious extraction of features, which are then used by machine learning algorithms for the classification of these individual objects. Such methods have some disadvantages. With such perception, object are found and classified in the scene rather than understanding the entire scene. Moreover development of such methods require a lot of tedious hand driven feature extraction. Errors in the early stages of the pipeline lead to mistakes in later stages, as each step in the pipeline acts as an individual algorithm. Moreover, the classification performance of these methods decreases, when the complexity of the scene increases, especially during segmentation of closed together objects [34].

One of the reasons of the success of deep neural networks, over other machine learning techniques, are the concept of representation learning and end-to-end training.

To the best of our knowledge, deep learning has not been used for Semantic Point Cloud Segmentation. Deep learning based solution for semantic point cloud segmentation, which is end-to-end and does not require hand crafted feature extraction is significant addition in the perception for autonomous systems.

1.2. Purpose

The goal of this thesis is to improve the quality and performance of perception for LiDAR, using deep learning. This include showing the proof of concept that deep learning can be used with the LiDAR sensors to produce an involved semantic point cloud segmentation for complete scene understanding. The scope of this study is limited to application of the automated driving in urban environment. The proof of concept formulates the task of 3D point cloud classification as a supervised learning problem and presents a solution that can perform novel 11 class semantic point cloud segmentation. Additionally, this study presents a detailed qualitative and quantitative analysis of the design choices and performance of the presented method.

1.3. Delimitations

The domain of thesis is limited to producing proof of concept with the LiDAR point clouds in the Urban environment, showing that supervised learning through deep convolutional neural networks can be used for semantic point cloud segmentation. The problem is formulated as supervised learning and requires a labeled dataset. Due to a lacking of a public dataset with point-wise annotations, the data is recorded in Stuttgart region and therefore only captures the dynamics of its urban setting. Labeling or annotation of LiDAR data, used for showing the proof of concept, suffers from errors and noises from multiple sources. Therefore all the reported evaluation measures of the presented method are close approximations of its performance.

1.4. Challenges and Contribution

The encoding of LiDAR data as input and designing of convolutional neural network (CNN) architecture to generate dense segmentations was major challenge of the thesis, as no prior studies had tackled 3D LiDAR data with deep networks for point-wise predictions. Moreover selecting an appropriate preprocessing technique for different modalities, handling of noise in LiDAR data, processing of a enormous dataset,

training and hyperparameter optimization of individual models and management of numerous huge deep CNN models were other challenges of this thesis. The major contribution of this thesis is CloudSeg CNN architecture, that incorporates elements from many diverse studies. On average, CloudSeg can reliably explain around 85% of the point clouds in each scan, despite its design just being the proof of concept for producing semantic segmentations with deep learning. While CloudSeg processes point cloud data as 2D depth maps, it's segmentations can be easily transfered to 3D space for involved 3D scene understanding.

In chapter 2, a detailed background is presented, including the literature review and the related work for point cloud segmentation classification and semantic image segmentation. Chapter 3 introduces the dataset and presents the method for generation of its ground truth. Chapter 4 presents the detailed description of the architecture of deep neural network (CloudSeg) and its training. Chapter 5 presents the evaluation of experiment and reports the analysis of the presented solution and results. Chapter 6 concludes the thesis and presents the future work.

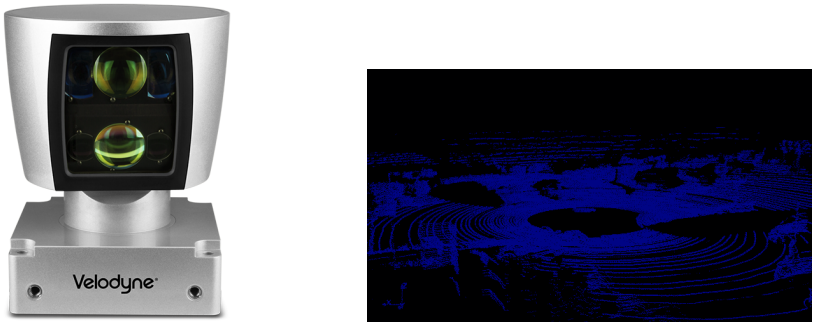
2. Background

This chapter introduces some basic concepts regarding LiDAR sensors and 3D point clouds. Moreover, it also covers introduction and detailed literature on the topics of semantic image segmentation, segmentation and classification of 3D point clouds and use of deep learning with LiDAR sensors.

2.1. LiDAR

Light Detection And Ranging (LiDAR) is a ranging sensor technology that measures distance to a target by illuminating that target with a laser light. It emits a focused beam of light (laser) and based on the returning beam, the distance between an object and the LiDAR Sensor is calculated. Typical LiDAR are capable of operating anywhere between 10Hz and 100Hz with accuracies in the decimeter and centimeter ranges. LiDAR is popularly used to make high-resolution maps, with applications in geodesy, geomatics, archeology, geography, geology, geomorphology, seismology, forestry, atmospheric physics, laser guidance, airborne laser swath mapping (ALSM), and laser altimetry [6]. Autonomous vehicles use LiDAR for obstacle detection and avoidance to navigate safely through environments, using rotating laser beams [7]. Cost map or point cloud outputs from the LiDAR sensor provide the necessary data for the robot software to determine where potential obstacles are located in the environment and where the robot is in relation to those potential obstacles.

LiDAR are classified by the number of dimensions (**D**) they measure, and are divided into 1-D, 2-D, and 3-D. LiDAR that measure a single distance, without moving beam mechanism are known as 1-D LiDAR, analogous to a laser-pointer that is capable of measuring distance only. LiDAR systems that employ a rotating mirror such that the laser beam is swept horizontally similar to a fan shape; are known as 2-D LiDAR and report both distance and the horizontal angle of the measurement. 3-D LiDAR either employ multiple lasers or a single laser or with the aid of a rotating mirror to make such common scan patterns as, horizontal sweeps at varying vertical angles, Z-shaped patterns, sinusoidal, and elliptical scans [9]. These 3D LiDAR often



(a) LiDAR Sensor **(b)** LiDAR Point Clouds

Figure 2.: Velodyne LiDAR Sensor Figure(a) show 64 channel Velodyne LiDAR sensor used in this study. Figure(b) show the sample scan of that is collected with Velodyne LiDAR Sensor. Courtesy [8].

adopt a polar coordinate system, and report both the distance, horizontal angle, and vertical angle of the measurement.

In addition to measuring distance, some LiDAR systems are also capable of measuring reflectivity, a measurement known as echo width. Reflectivity is often a function of the distance from the LiDAR, as well as a function of the surface of the object measured. Hence, an object that is highly reflective at a large distance may return the same echo width measurement by the LiDAR as an object that is less reflective, but closer.

Because the LiDAR system is actually transmitting a beam of light it is known as an active sensor, as opposed to a camera which would be a passive sensor. This often makes LiDAR more robust to changes in lighting, shadows, and environmental conditions, however a LiDAR scan is still capable of being "washed out" due to heavy precipitation. Additionally, despite being able to make highly accurate measurements, LiDAR often do not take in as much environmental information as cameras, and suffers from specular reflection, causing missing spaces in the laser scans.

This thesis uses a 3D Velodyne LiDAR comprising of 64 channels, meaning 64 lasers are mounted on the upper and the lower block of the sensor. This design allows the unit to spin at frequency of $10Hz$, recording approximately 130000 points per spin, delivering a 360° horizontal field of view (FOV) and a 26.8 degree vertical FOV. Additionally, sensor provides distance sensing and intensity data with high accuracy of approx $\sim 2cm$ sensitivity and a maximum range of $120m$. A 64 channel Velodyne sensor and the visualization of its single scan, is illustrated in Figure 2.

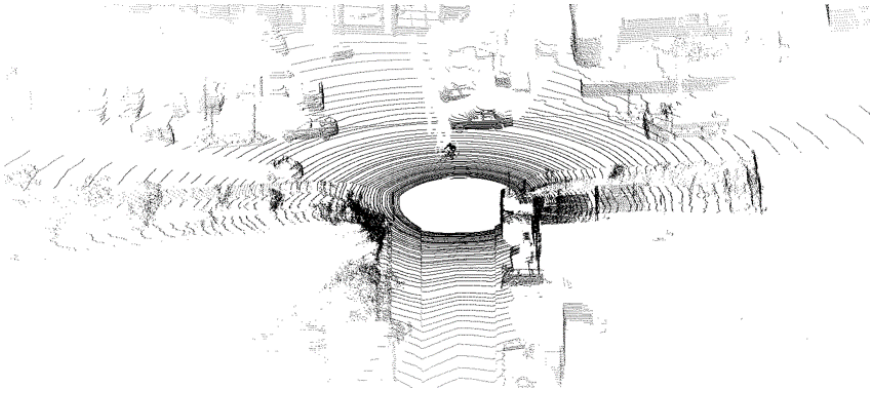


Figure 3.: Visualization of 3D Point Clouds in Urban Environment

2.2. Point Clouds

Point clouds are a set of vertices representing multi-dimensional structure. They are most commonly used in 2D and 3D data. In 3D space, point cloud data is defined by X, Y and Z geometric coordinates comprising an external surface of an object. Point clouds are created by 3D scanners like LiDAR. Point clouds are used in many application, including to create 3D CAD models for manufacturing parts, quality/metro-logy inspection, and a multitude of visualization, animation, rendering and mass customization applications. They are also used in the generation 3D models of urban environments, which are also used to represent the environment by mobile robots and autonomous cars. The visualization of point clouds from a LiDAR scan in Urban environment is illustrated in Figure 3.

2.3. Segmentation and Classification of 3D Point Clouds

Advances in 3D data acquisition technologies like LiDAR and demand for autonomous robotics applications has paved the way for increased interest in scene interpretation in different domains. Semantic scene understanding with 3D point clouds, recorded from the airplanes, mobile laser scanners, autonomous robots and the self-driving cars, have gained major research interest in recent years [10]. The segmentation and classification of 3D point clouds for the interpretation of urban scenes with LiDAR can be classified into two types: object-wise classification and point-wise classification.

Object-wise classification, identifying groups of point clouds into object categories, consisting of a pipeline of segmentation, feature extraction and classification (see

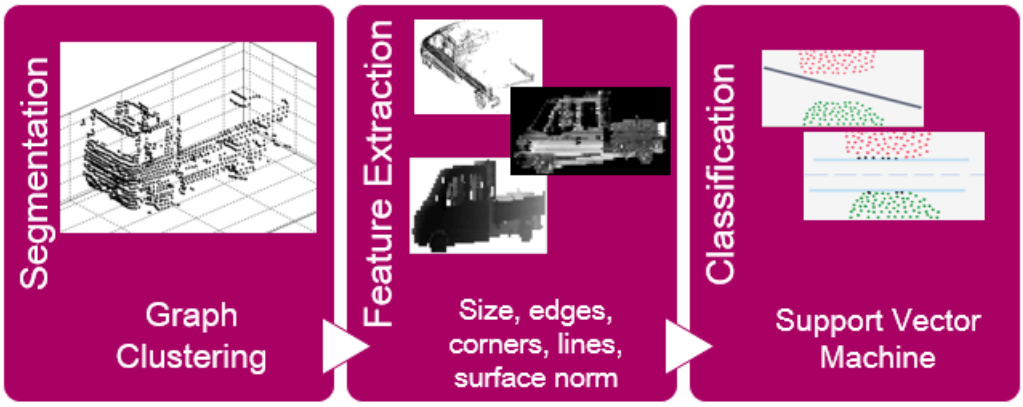


Figure 4.: Point Cloud Segmentation and Classification Pipeline This figure shows the classical segmentation, feature extraction and classification pipeline method used for segmentation and classification of 3D point clouds. Commonly during segmentation, a graph clustering algorithm is used to identify objects. Then geometric features are extracted. In last step, off the shelve machine learning algorithm is used to classify different objects in Point Cloud data. Courtesy [11].

Figure 4). During the segmentation step, the 3D point clouds are usually separated into logical groups of points or segments belonging to the same object. Sets of geometric features, like local point density and height from the ground, is used to segment object categories in outdoor scenes and to separate foreground from background [12, 13]. Moosmann et al. use surface discontinuities like surface convexity in a terrain mesh to separate various object classes [14]. A more common approach is the formulation of segmentation as a graph clustering problem using Min Graph Cuts and Normalized Graph Cuts [15, 16]. Moreover, Anguelov et al. and Munoz et al also use Markov Random Fields to segment and label 3D point clouds [17, 18].

The next step in the pipeline is the extraction of hand crafted features from the segmented 3D objects for classification. These may include geometrical features (shape, size, etc.), local descriptors (color, intensity, surface normals, etc.) or contextual features (position with respect to neighbors, etc.) of the segmented objects [19, 20, 21]. After the feature extraction, each of the 3D segmented objects are classified, using supervised machine learning algorithms or predefined geometrical models and thresholds [19, 20]. The machine learning algorithms used include Support Vector Machines, Gaussian Mixture Models, Random Forests, AdaBoost and Bayesian Discriminant Classifiers [22, 23, 24, 25, 26].

Point-based classification of 3D point clouds aim to classify the individual points

into separate object boundaries. Therefore the hand crafted features rely on the local 3D neighborhood, including the k-closest 3D points, fixed radius spherical neighborhood the or cylindrical neighborhood [27, 28, 29]. Associative and non-Associative Markov Networks and Conditional Random Fields are then used for classifying each point cloud [30, 31]. In addition, Dohan et. al. use a hierarchical segmentation algorithm, that learns random forest classifiers to predict if the two segments should be merged or they represents objects of the target class [32]. Hackel et. al. use 3D local neighborhood features and use random forests to learn semantic segmentation of point clouds [33]. Moreover Aijazi et. al. use combination of two techniques, morphological transformation and super-voxel based segmentation, for classification of point clouds in urban environment [34].

Common problems in this segmentation and classification of the 3D point clouds are caused due to the complexity of 3D scenes. In addition to irregular sampling and density variation of 3D point clouds, a large variety of objects and occlusions make is very difficult for any single approach to be successful in all the scenarios. An element that is shared by all of the above mentioned studies is that they rely heavily on expert knowledge to find the right features for the classification. Along with the computational burden of processing the large 3D point clouds, hand-driven feature extraction is a tedious process which requires invaluable domain experts. As the complexity of scene increases with diverse, closed-together and overlapping objects from many different classes, hand engineered feature extractors fail in robustly representing the complex 3D scene. On the other hand, Deep Artificial Neural Networks (see Appendix A), that use representation learning to automatically find the features from raw data, have also been used with 3D LiDAR Data (discussed in Section 2.5).

2.4. Semantic Image Segmentation

Perception with 2D images for scene understanding encompasses many computer vision tasks including image classification, object detection, image captioning as well as semantic image segmentation. Image classification classifies the image into some broader class category. Object detection distinguish different instances of the same objects in images and finds a bounding box around it. Image captioning aim to generate a sentence explaining activities in the image. While the semantic image segmentation clusters parts of the images together that belong to the same category.

Semantic Image Segmentation can also be understood as a pixel-level classification

task, where each pixel is classified into object classes to give the rich semantic information about the scene captured by the camera. This task has a variety of use-cases such as detecting road signs in urban scene understanding, detecting tumors medical instruments in operations and medical imaging, colon crypts segmentation, land use and land cover classification. Comparing to image classification and object detection, the background is no longer considered unimportant, and is divided into more informative classes. Being an active field of research, semantic image segmentation is evaluated on many benchmark dataset, including PascalVOC and Cityscapes Dataset [35, 36]. Pascal VOC 2012 consists of annotated photographs from $www.flicker.com$, including classes from indoor and outdoor scenes. On the other hand, Cityscapes dataset consist of semantic understanding of urban street scenes with 30 classes. Being an active research topic, the leader-boards of these benchmarks are updated in the matter of weeks.

The traditional methods for semantic segmentation, without using ANNs, comprise of diverse choice of feature like, pixel color, Histogram Of Gradients (HOG), Scale Invariant Feature Transform (SIFT), Bag-of-Visual-words (BOV), Poselets and Texons for image representation and use a wide variety of supervised and unsupervised machine learning techniques including the clustering algorithms, graph based segmentation, Random walks, Active Contour Models, Watershed Segmentation, Random Decision Forrest, Support Vector Machines (SVM), Markov Random Fields (MRF) and the Conditional Random Fields (CRF) [37]. In recent past, most of the afore mentioned techniques became obsolete with success of Deep Learning for semantic image segmentation.

2.4.1. Fully Convolutional Network

The most prominent breakthrough study of convolutional neural networks for semantic segmentation, in an end-to-end fashion, was conducted by Long et al. with Fully Convolutional Network (FCN) [38]. In this study, AlexNet [39] and VGG16 [40], the convolutional neural networks (CNN) used for classification, were adapted to perform semantic segmentation by replacing the fully connected layers with convolutional layer with a kernel size of 1×1. While the fully connected layers (inner products) of an image classification net completely discards spatial information and delivers only one feature vector for the entire image, the fully convolutional layer produces a feature vector for every pixel. Since the complete network is trained as an image filter, each pixel gets a probability distribution for each of the output classes and semantic segmentation is performed by taking the most likely class.

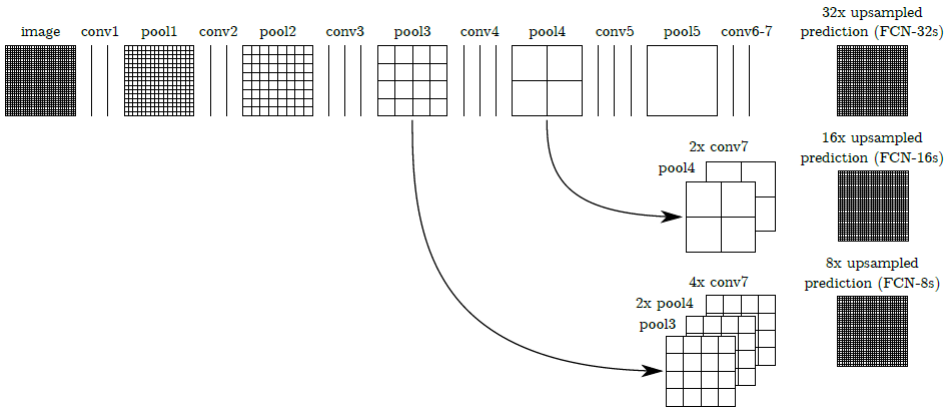


Figure 5.: Fully Convolutional Nerual Network (FCN) This figure illustrates the Fully Convolutional Network architecture developed by Long et al.. $Conv1$, $Conv2$, $Conv3$, $Conv4$, $Conv5$ are all 3×3 kernel convolutional layers, while $conv6(FC6)$ and $conv7(FC7)$ are 7×7 and 1×1 kernel convolutional layers, respectively, each followed by $ReLU$ non-linearity. Each pooling layer reduces the feature maps by factor of 2. The coarsest prediction is based on $pool5$ only (solid line) and is called $FCN-32$. The $FCN-16$ version (dashed line) first upsamples stride 32 predictions by a factor of 2 and then combine the output with the predictions of $pool4$. The $FCN-8$ variant (dotted line) combines predictions of stride 32, 16 and 8

While pooling improves classification accuracy, it causes loss in spatial information. The VGG16 and Alexnet include 5 pooling layers of stride 2, down-sampling by a factor of $32(2^5)$. In order to produce dense pixel wise prediction, the FCN upsample the feature maps by the so called deconvolution operations to restore the original image size. Given the weights, these deconvolution can also serve as a bilinear interpolation. Pooling also leads to coarse segmentation maps causing small objects to disappear in segmentation. FCN addresses this by fusing predictions of different strides to refine the segmentation maps. Moreover, FCN use shortcut connections that is outputs from earlier pooling layers fused with prediction through element-wise summation. The architecture of FCN is explained in Figure 5. With the stated features, FCNs were able to challenge the state of the art for many semantic image segmentation benchmarks.

2.4.2. Semantic Image Segmentation with Deep learning

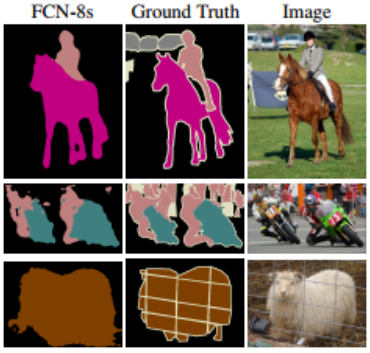
FCN were a breakthrough study and therefore it became the basis of many studies that came after it to fill up the left space for improvement. Badrinarayanan et al. used ideas from FCNs for their encoder-decoder architecture, using decoder network for producing dense pixel-wise prediction [41]. Chen et al. proposed Arteous convolutional operations in DeepLab CNNs to prevent the loss of spatial information and post-processing with Conditional Random Fields (CRFs) which improves the semantic segmentation performance while also doubles the number of parameters of the network [42]. Zhao et al. propose pyramid pooling as a component of CNN in their Pyramid Scene Parsing Network (PSP Net) while Wu et al.'s Deep Residual Network (ResNet) is currently the state of the art on Pasval VOC and Cityscapes semantic image segmentation benchmarks [43, 44]. Figure 6 highlight the performance of afore mentioned deep learning based semantic image segmentation algorithms.

2.5. Deep learning with LiDAR

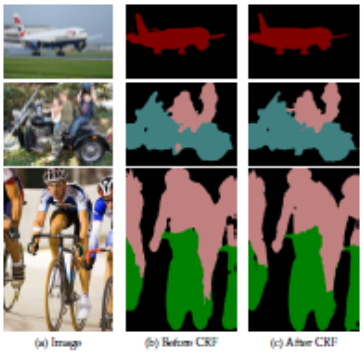
Computer vision is under a transformation through the use of deep convolutional neural networks in tasks like semantic image segmentation. However methods using 3D point clouds have not experienced a comparable breakthrough. Recent, few studies have attempted using CNNs with 3D point clouds.

A major hurdle for using Deep CNNs with point clouds attribute to computational burden introduced by the third spatial dimension and unavailability of densely annotated 3D point cloud datasets. 3D data as raw point clouds are sparse in nature and therefore the application of conventional CNNs is not trivial. Engelcke et al. address this by using CNNs constructed from sparse convolutions based on the voting scheme for object detection in 3D point clouds [45]. Their model detects cars, bicycles and pedestrians in 3D data and is the only study that used raw 3D point cloud in CNN. However they do not apply it to more advance perception tasks and limit them to fairly trivial three class problem. Maturana et al. study use of CNNs for dense 3D voxel occupancy grids computed from point clouds, however constructing dense voxel grids from point clouds adds huge preprocessing overhead. However, traditional CNNs have also been applied for dense 3D biomedical image analysis [46, 47].

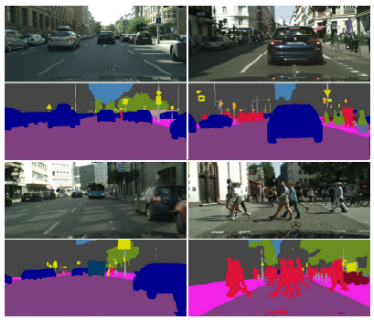
Similar to our approach dealing with 3D data, Li et al. project the 3D point cloud into a 2D depth map for their CNN-based approach for vehicle detection. In addition to depth maps, their model uses an additional channel for the height above the ground of a point for predicting detection scores and regressing to bounding boxes for vehicle



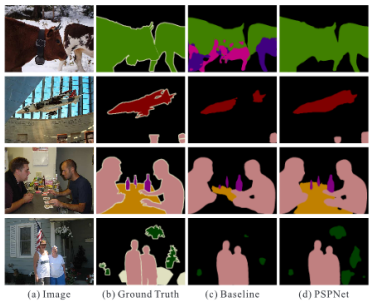
(a) FCN



(b) DeepLab CNN



(c) ResNet



(d) PSPNet

Figure 6.: Prominent Semantic Image Segmentation Studies Figure(a) shows qualitative performance of Fully Convolutional Network's semantic segmentation on Pascal VOC12 challenge benchmark[38]. Figure(b) shows qualitative performance of Deeplab's semantic segmentation on Pascal VOC 12 challenge benchmark with and without CRF post-processing[42]. Figure(c) shows state of the art qualitative performance of ResNet's semantic segmentation on Cityscape benchmark[44]. Figure(d) shows state of the art qualitative performance of PSPNet's semantic segmentation on Pascal VOC12 challenge benchmark[43].

in LiDAR data [48]. This model is limited to only detecting vehicles in point clouds while their CNN architecture is also limited, due to their down-sampling technique, to perform more involved detection or segmentation tasks which does not fully utilize the representation learning ability of CNNs.

Due to the projection of 3D LiDAR data to 2D depth maps, semantic image segmentation on RGB-D images also becomes a closely related topic [38, 41, 49, 50]. Numerous research studies have been conducted exploiting depth and RGB information for semantic image segmentation and object detection in images, however there is a lacking of studies for exploiting this relationship in perception of 3D point clouds. Therefore, we propose future work based on CNNs for LiDAR in the combinations with camera images in order to improve semantic point cloud segmentation.

3. Dataset

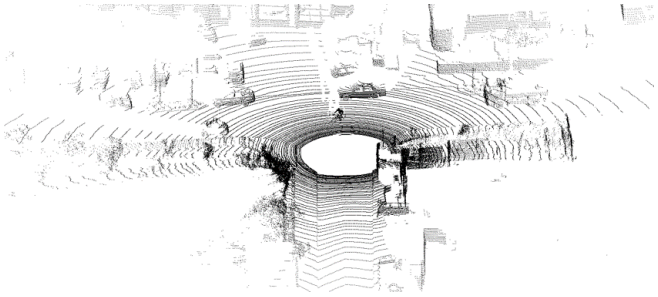
This chapter specifies the details of dataset that is used for semantic point cloud segmentation. There are many publicly availably datasets for perception in urban environment, like Cityscapes and Kitti, which also provide range data with RGB images [36, 51]. Due to the lack of annotations for 3D point clouds, these datasets could not be used in this study. For this purpose, collection and annotation of data is also presented as a part of this study. Method for annotating LiDAR point clouds for semantic point cloud segmentation is explained in this section, along with the analysis of these labels.

3.1. Recording

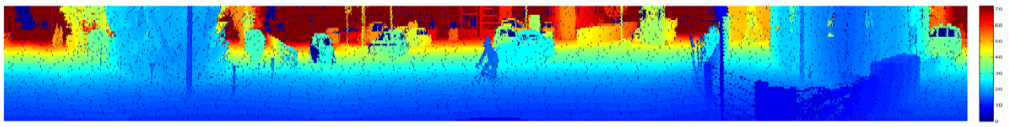
Data is recorded on the densely populated streets of Stuttgart. The goal is to include the dynamics of urban environment in the data, that include modern city settings with high population density, built infrastructure and a variety of vehicles on road. A Velodyne LiDAR Sensor, with 64 channel and approximately $120m$ range, is mounted at the top of the car to capture the 360° view around the vehicle. The data extracted from LiDAR sensor consist of position of point clouds in three dimensional space and their corresponding reflectance. Along with LiDAR sensor, a camera is also mounted which records video , covering roughly 45° angle of view, in the front of the car. Data is recorded for two hours of driving, accounting to 38400 LiDAR scans.

3.2. Preprocessing

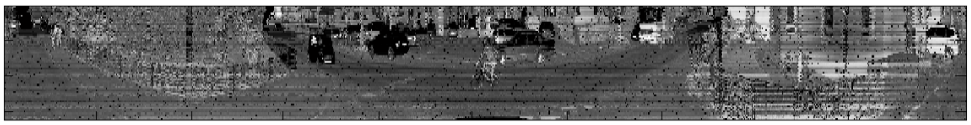
3D point clouds from Velodyne LiDAR sensors covers 360° field of view around the car, used for recording the data, with 64 channels. The 360° field of view is covered by 2048 lasers readings around the car, generating 3D positions of the total of 64×2048 points, with their respective reflectance, in each scan. From such a point cloud scan, 2D depth image is computed having dimensionality of 64×2048 . Each of 64×2048 pixel of this depth map corresponds to a single point in a 3D point cloud scan and



(a) 3D LiDAR point clouds



(b) 2D Panorama Depth map



(c) 2D Panorama Intensity map

Figure 7.: 3D to 2D projection of 3D point clouds This figure shows the conversion of 3D point clouds into 2D panorama images. 3D point clouds from Figure(a) would translate to a panorama depth image like Figure(b). Bicycle and a vehicle can be seen in both figure(a) and figure(b). Similarly, Figure(c), show intensity panorama image computed from the reflectance from LiDAR sensor, of the scan shown in Figure(a)

contains the distance of that respective point to the LiDAR sensor in meters (m). This conversion from 3D point cloud to 2D depth map is given by:

$$r_{i,j} = \sqrt{x_{i,j}^2 + y_{i,j}^2 + z_{i,j}^2} \tag{1}$$

where $r_{i,j}$ is a single pixel in the depth map and $(x_{i,j}, y_{i,j}, z_{i,j})$ are its corresponding coordinates in 3D space. Above formula is simply the distance of an individual point with the origin (sensor location in each scan). Covering the entire 360° scene, these depth maps become 360° panorama images. Similarly intensity image is also generated for reflectance of every point cloud, creating an intensity panorama image. The benefits of using this 3D to 2D transformation are as follows:

- 3D sparse representation of point clouds and reflectance is memory expensive as compared to its 2D workarounds.

- Such transformation fairly reduces the sparsity of 3D space and converts into 2D dense image.
- Deep CNN algorithms shows strong promise on 2D images and can be directly applied on the 2D depth maps.

However 2D depth and intensity maps are very different from RGB images, especially in terms of dimensionality, and content. Unlike images, such LiDAR depth image have different levels of density for environment representation, on different axises. Moreover, they are not as dense as RGB images and do not posses texture information of objects they represent.

The missing or invalid point are represented by zeros in such images. Moreover these depth images are normalized by dividing the distance value by 70. This 70 represents $70m$ range for depth under which the measurements are very accurate. Zero mean normalization, a popular normalization strategy, is not used because it will diminish the representation of missing point clouds and will not maintain same scale of environment for every scan. Depth images do not contain negative values and more than 99% of point-clouds lie are under $70m$ range threshold. Thus all the reading with in $70m$ threshold are normalized between 0 to 1 and reading beyond $70m$ are considered as outliers while the scale of the environment is maintained for all samples.

3.3. Generation of Semantic Ground Truth for Point Clouds

Since we formalize our problem as supervised learning, it is imperative to have the labels and the classes for the semantic segmentation of point clouds. These 3D point clouds from our LiDAR are to be classified in to the following 11 classes.

1. Bicycle / Motor Bike
2. Buildings
3. Bus / Car / Trailer / Truck
4. Fence / Wall
5. Person
6. Pole
7. Road
8. Sidewalks
9. Terrain

10. Traffic Lights / Traffic Signs
11. Vegetation

These classes are the subset of **cityscapes dataset** for urban environment [36]. Similar classes are merged in a single class to reduce the total number of classes from 19 to 11. This is done to simplify the problem as well as to help class imbalance between the frequent and rare classes.

3.3.1. Labeling method

The generation of ground truth or label for point clouds is not trivial. Unlike images, the hand labeling of these point clouds is not possible as their raw representation is not self explanatory to the human eyes. The method to generate the ground truth for point clouds make use of the images from the video camera mounted on the car during the recording of the data. The basic idea to generate labels for LiDAR point clouds can be understood in two steps: first generation of labels in camera image and second the projection of point clouds to that image to get the final labels.

Generation of Labels for 2D images

The first step is the generation of labels for camera image, which is the problem of semantic image segmentation in urban environment. For this purpose, a preexisting deep CNN is used to produce robust semantic image segmentations of the camera images, recorded through mounted video camera during the data collection. This algorithm produces the classes of city scape dataset which are merged in post processing to match previously mentioned classes. The details about this deep CNN can not be disclosed in this thesis as it is propriety of Robert Bosch GmbH and thus referenced as *Secret Network* through out the study. Figure 8(a) and (b) illustrates this process, where (a) is the raw camera image while (b) contain the segmented camera image, produced by the Secret network.

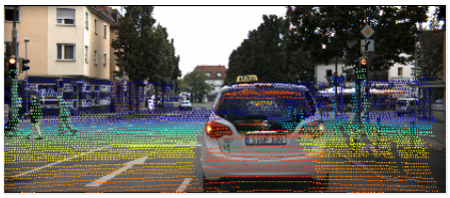
Projection of point clouds to 2D Plane

Second step is to compute the projection of 3D point clouds on 2D images. For a given 3D scan, the camera image with closest timestamps with in $100ms$ is chosen and projection of the 3D point clouds over that image is computed. This projection is computed by transforming the point clouds to camera coordinates. This transformation is corrected by manually calibrating the 3D point clouds on the camera image and computing the parameters for the transformation. Figure 8 (c) and (d)

(a) Segmented Camera Image

(b) Labels generated on images using Secret Network

(c) Raw Camera Image



(d) Question Answering from Images

Figure 8.: Generation of Labels These figures shows the two steps for the generation of Labels of 3D LiDAR data. Figure(a) and (b), show the conversion of raw 2D camera images to segmented image through secret network. Figure(c) and (d), show the projection of 3D point clouds on the 2D camera images in order to transfer labels from (b) to (d)

illustrates the projection of point clouds on camera image. Figure 8(c) contains the raw RGB image and (d) shows the projection of corresponding point cloud scans onto the RGB image.

The projection of point clouds to RGB images and semantic segmentation output for these images from secret network allows the computation of labels for each point cloud scans. The point clouds are projected to segmented images and labels are transfered from segmented image to point clouds. The overlap of field of view of camera and LiDAR sensor only allows the generation of labels for 256×40 point clouds in front of the car. Quality of these labels is discussed in the next section.

3.3.2. Analysis of labeling Method

The generation of the labels or ground truth with the afore mentioned method is in no way perfect. The problems and inaccuracies in such labels is discussed as follows:

- As described before, the point cloud data suffers from the specular reflection and generated infinite distances for few lasers, which can be seen as missing values. They account for around 10% points in each scan. These missing or invalid values are not interpolated in the labels, and therefore are added as

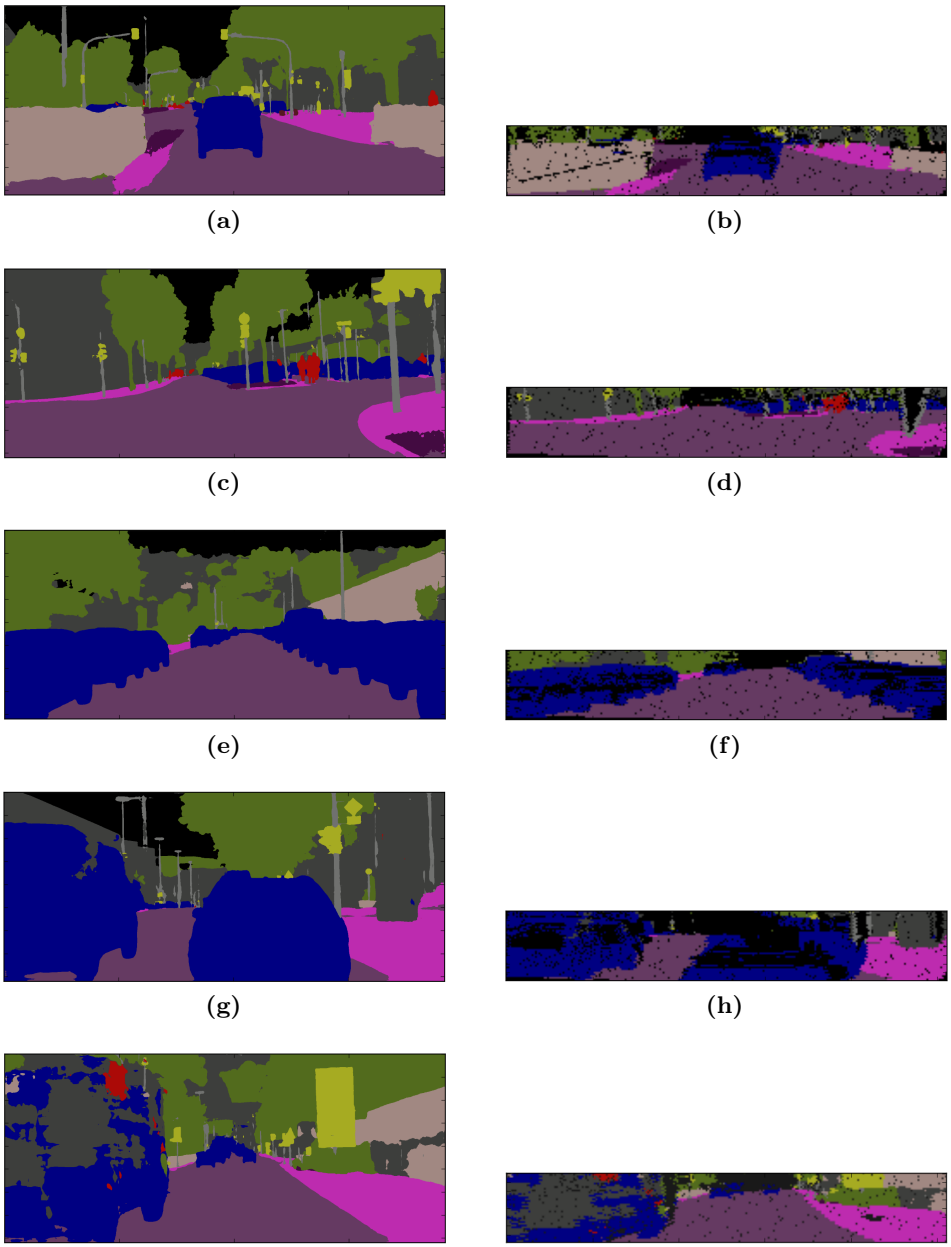


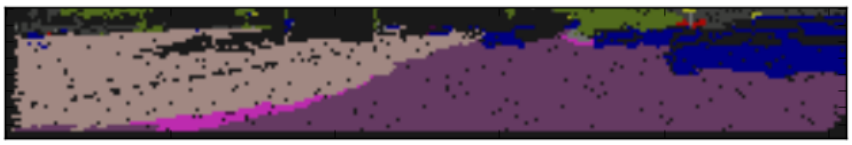
Figure 9.: Examples of good and bad labels from the dataset These figures shows the good and bad examples of the ground truth, generated using the method explained in Section 3.3. Figure (a,c,e,g,i) are the segmented camera images, and (b,d,f,h,j) are their corresponding ground truths for 2D projected point cloud data.

another class in the labels.

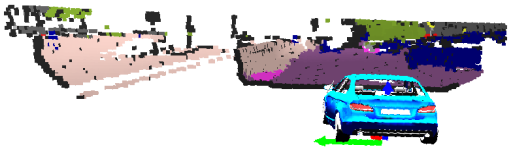
- Moreover, a fundamental conflict arises in transferring the labels from camera images to LiDAR point clouds. While video camera images can see long range entities like sky, LiDAR sensors can not detect any long range entity. In principle, none of the point clouds should be projected on an image where it is assigned a class of long-distance entity like sky. But in practice few of such labels do appear in point cloud annotations and none of them are used during the training or testing phases.
- Since the labels are generated using a secret network for semantic image segmentation, all the errors from that secret network would propagate in these labels. In other words, solution with such labels can not do better than the original secret network used for labeling.
- The LiDAR sensor and video camera used for recording the data are mounted at different positions on the car. While the LiDAR sensor is mounted on the center of car' roof, the video camera is mounted near the windshield on the left side of the car. Thus both camera and LiDAR sensor have different field of views that causes some occlusions in the labels projections. However these occlusions can be detected analytically and ground truth assigned to them are not used during the training or testing phases.
- It is also possible to have other incorrect labels due to any projection errors, caused by imperfect calibration of point clouds to camera images.

Despite the above mentioned inaccuracies in the labels, the ground truth annotations are more than capable for the formal proof of concept, showing that deep learning can be used for semantic point cloud segmentation. Figure 9 show the labels for 5 samples from the dataset in 2D representation with their corresponding segmented camera images. The first three sample show how the labels can effectively represent the scene in diverse scenarios. The black spots show the missing point clouds. Farther and small objects like poles are also represented nicely as shown in (c) and (d). The noise in labels due to occlusions is illustrated in fourth sample(Figure 9 (g,h)). The last sample(Figure 9 (g,h)) illustrates, how the mistakes from secret network are propagated in labels for point clouds, as the truck is segmented incorrectly in (i) and is carried forward in (j).

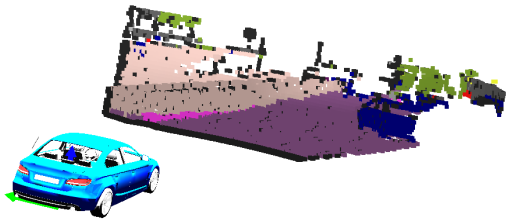
Figure 10 show the qualitative evaluations of labels in 3D space. 2 samples are illustrated, (a) show the ground truth of a scene, while (b and c) represent 2 point of views of its 3D representation. It can be seen that most of the structures like



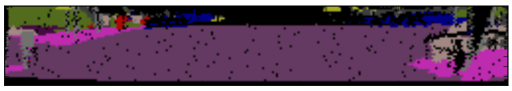
(a) 2D labels



(b) Labels viewed in 3D point of view

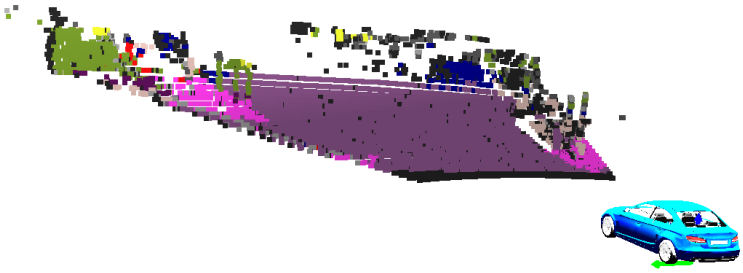


(c) Labels viewed in 3D point of view



(d) LiDAR Labels

(e) RGB labels



(f) Labels viewed in 3D point of view

Figure 10.: Visualization of generated labels in 3D These figures show visualize the generated labels in 3D, in order the analyze the quality of labels. Figure(a,b,c) represent first scene, where (a) is the 2D labeled image and (b,c) are its 3D representation in different point of views. Figure(d,e,f) represent second scene, where (d) is the 2D labeled LiDAR image, (e) is its corresponding segmented camera image and (f) is its corresponding 3D representation.

vehicle, road, vegetation is annotated correctly, while some labels are also wrong due to occlusions like the gray point clouds on the left side of 3D representations. Another, more traditional urban scene is presented by Figure 10 (d, e and f). The labels in 3D show that roads, vehicles are labeled correctly while certain errors can also be observed in poles.

4. Method

This chapter presents the proposed Deep Convolutional Neural network called **CloudSeg**, for Semantic Point Cloud Segmentation. The architecture of CloudSeg is explained along with the intuitive reasoning of its design choices. The specification regarding the training of CloudSeg is also explained along with the hardware and tools that were used to train it.

4.1. CloudSeg

In this section, design inspirations and details of CloudSeg are presented.

4.1.1. Problem Description

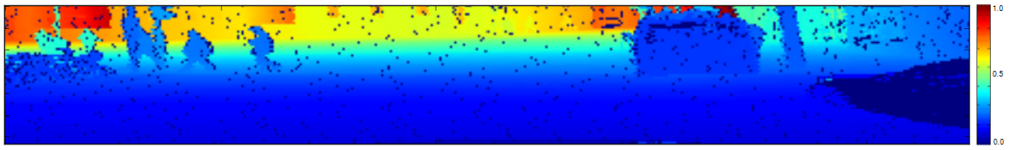
Semantic point cloud segmentation is defined as a point level classification problem, where the goal is to learn a model that can classify each point, from a given LiDAR scan, to predetermined 11 classes. Formally, given a LiDAR point cloud dataset X with labels Y, we aim to learn the mapping $X \rightarrow \hat{Y}$, such that the error between Y and $\hat{Y}$ is minimized. This mapping is learned by training a Convolutional Neural Network (see Appendix).

The handling of 3 dimensional sparse point cloud data in CNN is a challenge. Therefore 3D point clouds are converted into 2D panorama depth and intensity images. Another challenge is the unavailability of annotated data, therefore data is recorded and labeled. Our 3D point cloud dataset consist of channel-wise concatenated depth and intensity panorama images. The details about dataset are presented in chapter 3.

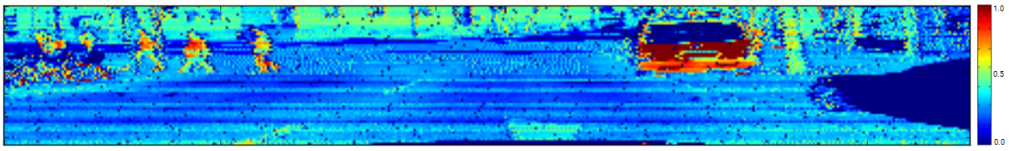
4.1.2. Intuition

In order to train 11 class point-wise classification model, 3D point cloud data is encoded and represented as 2D panorama depth and intensity images. By doing so, the problem of semantic point cloud segmentation is formalized similar to supervised learning problem of pixel-wise semantic scene understanding in images. Fully Convolutional

Neural Networks(FCN), explained in chapter 2, are a breakthrough study for semantic image segmentation [38]. In order to design CloudSeg, we derive inspiration from FCN's CNN architecture and other studies that build upon it. Additionally, inspiring from $SegNet$, we divide the CNN architecture into two separate parts, the Encoder and Decoder [41]. The Encoder learn the feature representation from input depth and intensity maps and produce the down-sampled prediction score maps. Decoder uses these down-sampled score maps and shortcut-connections to generate upsampled dense prediction score maps. Moreover, our CNN architecture design choices aim to reduce the number of model parameter to make CloudSeg practically robust. Given the ever growing innovations and developments in deep learning community, the design choices of CloudSeg aim to keep the model simpler wherever possible, as the final goal of this thesis is to present a working proof of concept.



(a) 2D Depth after cropping



(b) 2D Intensity after cropping

Figure 11.: Input of CloudSeg These figures shows cropped center from panorama depth(a) and intensity(b) image as the input of proposed CNN.

4.1.3. CloudSeg: Fully Convolutional Network for Point Cloud Segmentation

In this section, we explain the architecture of our fully convolutional network, CloudSeg.

Input to the Network

The input to our CNN architecture are panorama depth and intensity images. In depth image, each pixel represents the distance of point cloud to the sensor, while in intensity image, each pixel represents the reflectance computed through LiDAR sensor. The dimensions of input panorama images are 64×2048 out of which 64×448

center pixels are cropped. The depth and intensity images are then stacked across channel, forming the $2 \times 64 \times 448$ dimensions of input.

As explained in chapter 3, the ground truths for semantic segmentation are generated, using a camera mounted on the vehicle. The field of view (FOV) of this camera covers the front of the vehicle, which only overlaps 45° FOV with LiDAR's 360° FOV, does not cover lower laser beams. Thus generating labels for 40×256 points, lying in front of the vehicle, in the LiDAR scan. As compared to our 40×256 labels, the input to the network is 64×448. Additional 96 columns on each side and the 14 rows in the bottom of point cloud scans, are kept as part of the input to provide context information for the segmentation. In the next chapter, we also assess and analyze the different encodings of point cloud as input data.

Convolutional Layer and Non-linearity

Similar to tradition FCN, CloudSeg consist of sequence of convolutional layer followed by a non linearity ($ReLU$) and pooling layers. The convolutions are 3×3 kernel with one pixel padding, except the first convolutional layer which has 32 pixel horizontal and vertical 0-valued padding. Such padding are also a feature of original FCN and are useful for CNNs to detect borders of images. They also increase the total dimensionality of output feature map, providing more design flexibility for the rest of the CNN architecture.

Despite the encoding of LiDAR as depth images, as compared to camera image, their representation are very coarse with varying resolution across dimensions. However, similar to image based CNNs, a 3×3 kernel is chosen for convolutions. Alternatively, a non-square kernel or unequal strides across dimensions could be used. However, they were not found to be useful, as the point cloud representation in 2D depth maps does not represent the distance between the point clouds evenly. Such technique reduced spatial information and did not aid in the learning of good feature representations. Moreover each convolutional layer is followed by a Rectifier Linear Unit ($ReLU$) as non linearity.

Pooling

Pooling is another important feature of convolutional neural network. It is responsible for down-sampling of feature maps and loss of spatial information, while also providing the important robustness to small deviations in data. Maximum Pooling is used with 2×2 kernel and the stride of 1, which down samples the feature maps by a factor

of 2. The dimensions of input data limit the number of pooling operations in our architecture, hence it is also a limiting factor of how deep CloudSeg can actually be. FCN and other comparable deep networks have 5 pooling operations in total, which works nicely with dense images with texture information. For low dimensional LiDAR data this is reduced to 3 pooling operations while keep a receptive field size reasonably large. These 3 pooling operations ensure that our model is deep enough to learn good feature representation while also not loosing the spatial knowledge to an unrecoverable extent. The reduction in depth of the model is compensated by adding more output feature maps for convolutional layers than in a FCN (to learn more feature representation).

1×1 Convolutions

Similar to $FC6$ and $FC7$ layers of FCN, we have a convolutional layer $FC4$ with 7×7 kernel followed by another convolutional layer $FC5$ with 1×1 kernel, each accompanied by ReLU non-linearity respectively. These layers act similar to a fully connected layer with 4096 output feature maps. Oliveira et. al. propose to decrease output feature map size of $FC6$ and $FC7$ from 4096 to 1024, reducing parameters to save memory and computational power for a more practical use [52]. Motivating from them, CloudSeg use 2048 feature maps with our $FC4$ and $FC5$ convolutional layers.

Drop-out and Batch Normalization

Between $FC4$ and $FC5$, we have drop-out layers (see appendix A.3.1), with a dropping out probability of 50%. These dropout layers, prevent over fitting and can also be seen as data augmentation. Moreover, CloudSeg uses 3 Batch Normalization layers (see appendix A.3.1), one following each max-pooling layer. Batch normalization helps in training by keeping the feature maps zero-mean normalized. It also gives the network the ability to optimize an arbitrary mean and standard deviation for it's feature maps, that help learn better representations. We place the batch normalization after every max-pooling operation, where the dimensionality of the feature maps are reduced, which is more memory efficient.

Upsampling and Shortcut Connections

Following $FC4$ and $FC5$ layer, a convolutional layer is used to produce the scores for the prediction. This layer has 12 channels each representing a label plus an additional class for invalids noise. This score is however down-sampled due to pooling layers

and does not represent the segmentation output. This score map is required to be up-sampled to produce dense segmentation map for the point clouds. There are a number of methods for doing up-sampling in convolutional neural networks. The detail level and density of the semantic segmentations heavily depend on upsampling. Few possible methods can be, upsampling by bilinear interpolation, or learning the upsampling weights with deconvolutional layers or developing a complex sub network of convolutional and deconvolutional layers. We upsample our scores by using deconvolutional layer (see appendix A.3.1), initialized to bilinear interpolation weights, which are fine-tuned during training.

Shortcut connections are also used to aid this upsampling [38]. The shortcut connections are the prediction scores, computed through a scoring convolutional layer, at the output of different pooling layers. These prediction scores are element-wise added before upsampling, this helps in incorporating early feature representations which are lost due to spatial information loss. We have two upsampling operations using deconvolutional layers, with the introduction of shortcut connections from third and second pooling layer respectively. Each upsampling layer, upsamples the score-maps by a factor of 2. As compared to three pooling layers, that down-sample the original input dimensions by a factor of 8, we use two upsampling layers, upsampling by the factor of 4 in total, the difference is covered by additional contextual data as part of input and padding included in the convolutional layers. Moreover, two deconvolutional layers also prevents extra-smoothing effect which appears due to upsampling with a high factor. The result of final upsampling is the score map, for each class in the dimension 64×256, which is cropped to produce 40×256 dimensional score, corresponding to each of the point in output point clouds and matching the dimensions of our ground truth labels.

4.1.4. Encoder

As shown in Figure 12, The Encoder architecture of CloudSeg processes the input data as follows.

- The cropped input data is fed into the first 3×3 convolutional layer ($Conv1\_1$) with additional padding, outputting 32 feature maps, and is followed by a $ReLU$. Its output is fed to another convolutional layer ($Conv1\_2$) that outputs 84 feature maps and is followed by a $ReLu$, max pooling operation and batch normalization ($BN1$).

- It follows two more convolutional layers ($Conv2\_1$ and $Conv2\_2$) each out-

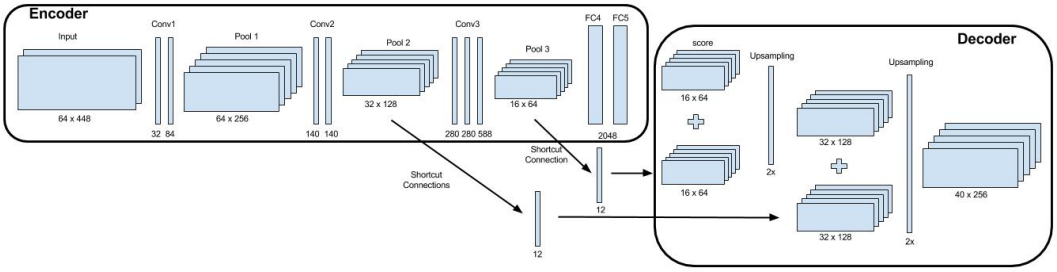


Figure 12.: CloudSeg Architecture This figure is an illustration of CNN architecture of CloudSeg. The encoder and decoder sections in the network are highlighted and explained in Section 4.1.4 and 4.1.5 respectively.

putting 140 feature maps and followed by $ReLU$. It is then extended by the max pooling and the batch normalization ($BN2$) operations.

- It is then followed by three more convolutional layers ($Conv3\_1$, $Conv3\_2$ and $Conv3\_3$) each outputting 280, 280 and 588 feature maps and followed by $ReLU$. They are extended by last max pooling and batch normalization ($BN3$) layer.

- Finally it is followed by a 7×7 convolutional layers ($FC4$), $ReLU$, Dropout, a 1×1 convolutional layers and another $ReLU$ layer. It is extended by last score convolutions generating 12 output feature maps.

4.1.5. Decoder

As shown in Figure 12, data passing through the Decoder architecture of CloudSeg during training goes through the following operations.

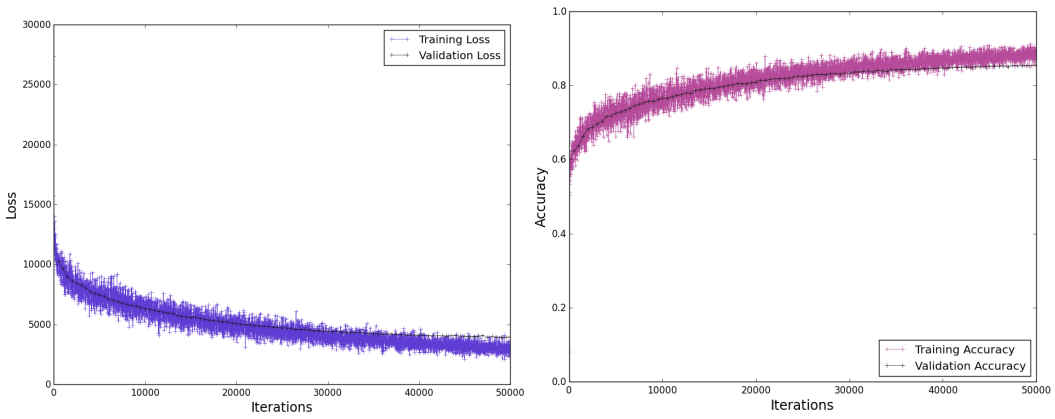
- A shortcut connection from $BN3$ is added to computer score from Encoder. It is then upsampled using deconvolutional layer ($US1$) doubling the size of the feature maps.

- Another shortcut connection from $BN2$ is added to the upsampled score of $US1$. It is upsampled again using another deconvolutional layer ($US2$) effectively reaching the required size for generating dense prediction of point clouds.

- The Upsampled score is then cropped to match the exact size of the labels and Softmax Cross Entropy Loss is computed on it for back propagation.

4.2. Training and Validation

The dataset of 38400 LiDAR scans is divided into 3 sets, namely training, validation and testing sets, each containing 30000, 800, 7600 samples respectively. This test set represent 20% of the total data, while validation data is deliberately chosen to be small as it validated after every 100 iteration during the training period. Training set with its respective segmentation labels is used to train CloudSeg using Adam Optimizer (see appendix A.2.2) with its default parameters $\beta 1 = 0.9, \beta 2 = 0.999, \epsilon = 10^{-8}$. A mini batch size of 20 is used, accommodating to available GPU memory. The learning rate of 10^{-4} is used for the 50000 fixed training iterations, recording training loss and accuracy every 10 iterations and validation loss and accuracy every 100 iterations (see Figure 13). The weights of all the convolutional layers are initialized from Xavier distribution.



(a) Training/Validation Loss **(b)** Training/Validatiom Accuracy

Figure 13.: Learning Curves of CloudSeg Plot(a) shows the training loss and validation loss of 50000 training iterations. Plot(b) shows the training accuracy and validation set accuracy through the training iterations.

The loss function that is optimized during the training is unnormalized weighted softmax cross entropy loss, it is pixel wise softmax over the final feature map combined with cross entropy loss function. Since the training data is highly skewed towards few classes, the weighting in loss function is added to compensate the less occurring classes in the data. The purpose for weighting the loss is to present a remedy for dataset class skew and therefore a very simple weighting strategy is used by doubling the weight of three less occurring classes, namely people, poles and traffic signs. The loss function is given as:

$$J(\theta) = -\sum_{i=1}^{m}\sum_{k=1}^{K} W_k I\{y^{(i)} = k\} \log p(y^{(i)} = k \mid x^{(i)}; \theta) \tag{2}$$

where m is the number of samples and K is the number of classes. W_k is the class weight, the indicator function ($I\{y^{(i)} = k\}$) determines whether the x belongs to class K and $p(y^{(i)} = k \mid x^{(i)}; \theta)$ is the conditional likelihood of the softmax.

Moreover a weight decay of 10^{-6} is added for regularization in excess of dropouts. Finally, The CNN is trained for 50000 iterations covering 33.3 epochs in total snapshotting every 10000 iteration. Figure 13 show how the training loss and accuracy on training set is changed with respect to the validation set. We stop the training of the network after fixed 50000 iterations, after which the networks starts to show signs of overfitting. The training time of CloudSeg takes approximately 41.8 hours in total.

4.3. Implementation Details

All the implementation of CloudSeg was done in *Caffe*, deep learning framework [53]. The implementation of weighted cross entropy softmax loss was adapted from [54] and merged into later version of *Caffe*. All the experiments are conducted on a Nvidia Titan X Graphic Processing Unit (GPU) with memory of $12GB$. $Matlab$ is used for all the preprocessing and conversion of data into $HDF5$ format while all the post training is done in Python and *PyCaffe*, which is the python interface with for *Caffe*. The evaluation metrics reported in the next chapters are calculated using $Scikit-learn$ [55].

5. Results and Analysis

This chapter presents the quantitatively and qualitatively evaluation of CloudSeg. Moreover it also presents additional experiments to perform quantitative analysis of certain design choices in CloudSeg.

5.1. Results

CloudSeg is evaluated on the test dataset, as described in chapter 3. The 7400 samples of the test set were not seen by CloudSeg during training, therefore the measures reported on it will show the generalization ability of CloudSeg. The metrics computed to evaluate the performance are reported in Table 1.

Measure	value
Accuracy	85.72
Mean Precision	70.7
Mean IoU	57.5
Frequency Weighted IoU	76.1

Table 1.: Overall Performance Evaluation of CloudSeg

Let n_{ij} be the number of pixels of class i predicted to belong to class j, with n_{cl} different classes, and let $t_i = \sum_j n_{ij}$ be the total number of pixels of class i. The point-wise Accuracy is given by:

$$Accuracy = \frac{\sum_i n_{ii}}{\sum_i t_i} \tag{3}$$

The overall pixel accuracy of CloudSeg is 85.72%, meaning that generally the algorithm can correctly explain 85.7% of the points in every scan.

We also report the Mean Precision of CloudSeg on the test set. Mean Precision is

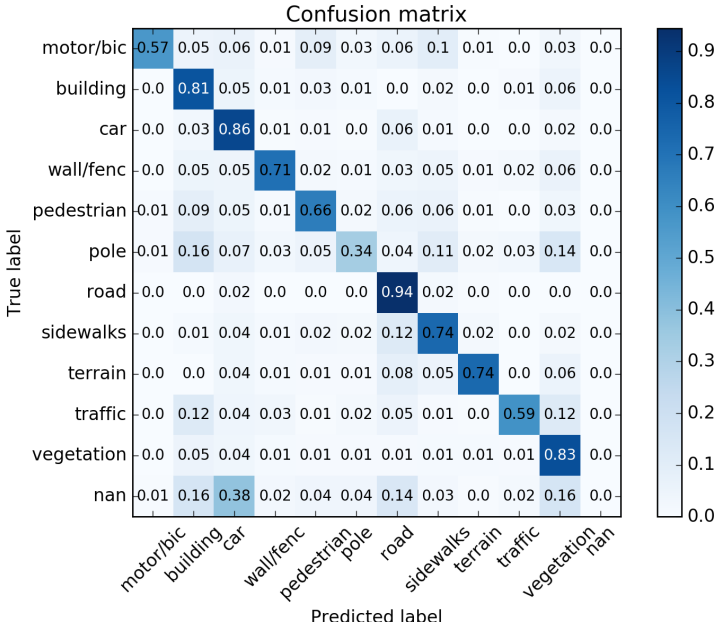


Figure 14.: Confusion Matrix of CloudSeg

the mean of all the per-class precisions, which is given as:

$$Mean\ Precision = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i} \tag{4}$$

This measure gives better estimate of the performance as it refers to the closeness of measurements for each class. The mean precision of CloudSeg is 70.7%.

Another important measure, that is often reported with Semantic Image Segmentation is the Mean Intersection Over the Union (IoU). For each class, this metric computes the intersection over union of the predicted pixels and ground truth pixels over the entire dataset.

$$Mean\ IoU = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}} \tag{5}$$

Mean IoU of CloudSeg is reported to be 0.575. We also report the Frequency Weighted Intersection over Union (IoU), which is given as:

$$Frequency\ Weighted\ IoU = (\sum_k t_k)^{-1} \sum_i \frac{t_i n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}} \tag{6}$$

The frequency weighted IoU is the average of per-class IoU, weighted by the frequency

Classes	Precision
Bicycle/Motor Bike	0.57
Buildings	0.81
Vehicle (Bus/Car)	0.86
Fence/Wall	0.71
Pedestrian	0.66
Pole	0.34
Road	0.94
Sidewalks	0.74
Terrain	0.74
Traffic Signs	0.59
Vegetation	0.83

Table 2.: Per-class precisions of CloudSeg.

of that class in the dataset. The frequency weighted IoU of CloudSeg is 0.761.

In order to perform a more detailed analysis of CloudSeg, we also report the per-class level performance metrics. The per-class accuracy presented in Table 2, clearly shows that the best performing classes are the *road*, *vehicles*, *vegetation* and *buildings*, which are also the most occurring classes in the dataset. Additionally, the least performing class is the *poles*. The *poles* in the dataset can be considered a special challenge for the algorithm due its shape and nature of material. The shape and material of the poles cause the laser sensors to generate many invalid point clouds around it due to specular reflections, which also generates uncertainty in its structure. While these poles also occur infrequently in the dataset, the situations where these poles are located at a larger distance to the sensor also cause a challenge, since only a very few point clouds fall on their surface, which can force the algorithm to miss its presence altogether during the down-sampling. However the use of shortcut connections and fine-tuning weights in the loss can help significantly to improve the identification of these poles in the data.

Note that all the metrics that are reported here are computed using the ground truth described in chapter 3. As explained before, the ground truth generated for the data have errors and noise from multiple sources and therefore all the measurements are mere estimations of performance. For these reasons, we also evaluate the qualitative results of CloudSeg in next section

5.2. Qualitative Results

This section presents the qualitative evaluation of CloudSeg. We evaluate CloudSeg by looking into visual feature kernels and output heatmaps of the deep network as well as the outputs generated by the CloudSeg in 2D and 3D space.

5.2.1. Feature Kernels

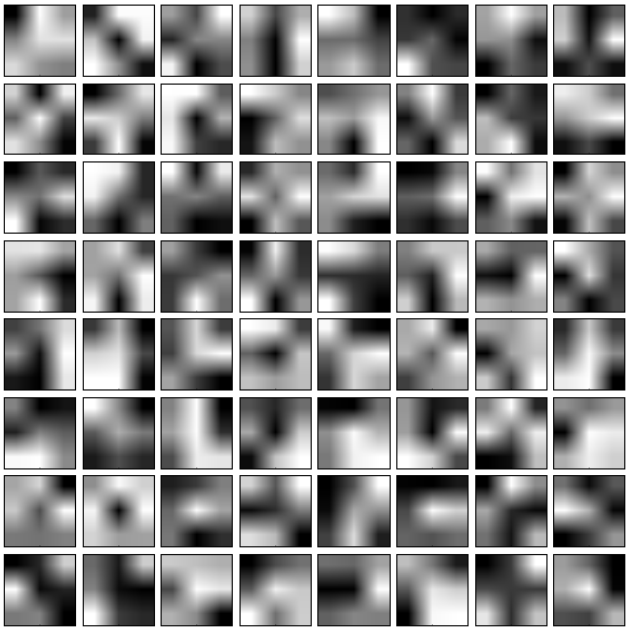
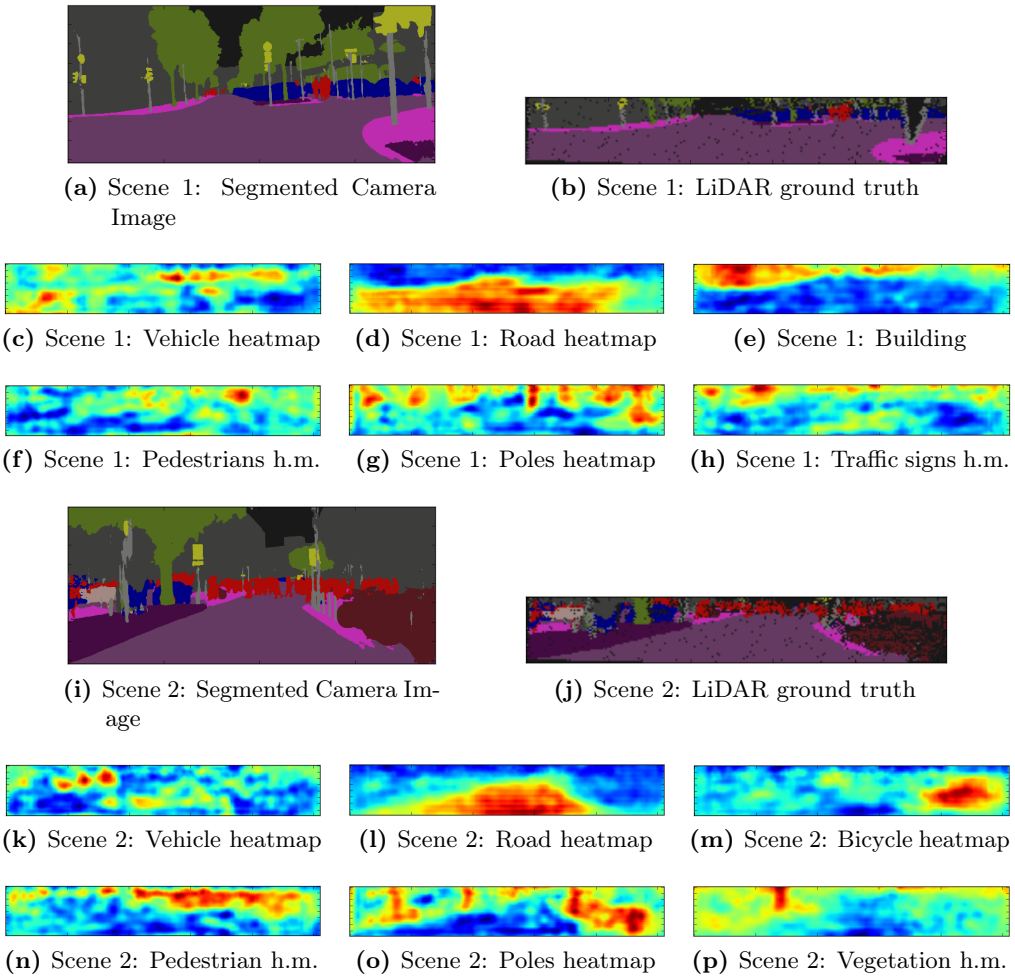


Figure 15.: Convolutional Filters of first convolutional layer of CloudSeg

Features learned in the first layers of a convolutional neural network are perhaps the most important features in any deep learning algorithm. These features represent the early abstract representations of the data, which constitute the basis for building higher abstractions in deeper layers of the network. In case of urban environment and semantic point cloud segmentation, where detailed scenes are made of diversely shaped objects and complete scene understanding is required, these physical building blocks are the edges around different objects that form the building blocks of more abstract features. Therefore the features kernels from the first convolutional layer are presented in Figure 15. In this figure, each small image represents a kernel weight learned by the first convolutional layer. It is evident each of this kernel attempts to detect edges in the input data, upon which more abstract features representation of the scene are extracted.

5.2.2. Output heat maps

The output of CloudSeg is a 11 channel probability map, with each channel corresponding to one of the 11 classes we aim to predict. Each of these output channels is a heatmap for each individual class. In such an output heatmap of a particular class, the pixels where the probability of the belonging to that class is highest are illustrated by red color and lowest are illustrated by blue color. The output heat-maps of 2 samples from test set are depicted in Figure 16.



(a) Scene 1: Segmented Camera Image

(b) Scene 1: LiDAR ground truth

(c) Scene 1: Vehicle heatmap

(d) Scene 1: Road heatmap

(e) Scene 1: Building

(f) Scene 1: Pedestrians h.m.

(g) Scene 1: Poles heatmap

(h) Scene 1: Traffic signs h.m.

(i) Scene 2: Segmented Camera Image

(j) Scene 2: LiDAR ground truth

(k) Scene 2: Vehicle heatmap

(l) Scene 2: Road heatmap

(m) Scene 2: Bicycle heatmap

(n) Scene 2: Pedestrian h.m.

(o) Scene 2: Poles heatmap

(p) Scene 2: Vegetation h.m.

Figure 16.: Output heat-maps highlighting detection of individual classes

Figure 16 (a - h) correspond to the first scene, where (a) shows the segmented scene as through the video camera, (b) shows its corresponding ground truth for 2D depth map and (c - h) show the heat-maps for *vehicles*, *road*, *building*, *pedestrians*, *poles* and *traffic signs* respectively. Figure 16 (i - p) correspond to the second scene, where

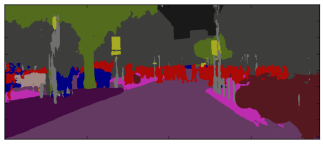
(i) shows the camera segmented image, (j) shows its corresponding ground truth for 2D depth map and (k - p) show the heat-maps for *vehicles*, *road*, *motorcycles*, *pedestrians*, *poles* and *vegetations* respectively. These output heat-maps can also be seen as individual class segmentation results. The heat-maps for individual classes show CloudSeg's ability to robustly detect diversely shaped objects belonging to different classes in the complex urban scenes. For example, Figure 16(d and l) show heat maps for pixels that belong to *road* in both scene while Figure 16(g and o) show vertically shaped objects for detected *poles* in the scene.

5.2.3. 2D Qualitative Results

In this section, we present some good and bad examples from the test set to evaluate the visual qualitative performance of CloudSeg. The 2D semantic segmentation results are presented in Figure 17. This figure illustrates 6 diverse urban scenes, each represented with dense segmented camera image (a,d,g,j,m,p), it's corresponding LiDAR ground truth (b,e,h,k,n,q) and semantic segmentations generated by CloudSeg (c,f,i,l,o,r) using maximum a posteriori. The segmented image is presented in order to comprehend the each individual scene, as it is more informative explanatory to human eye as compared to its corresponding LiDAR ground truth and therefore, when presented with segmented image, it is easy to judge the qualitative performance of CloudSeg prediction.

From the presented images, it is evident that the CloudSeg is able to robustly identify and segment the LiDAR scans into individual classes of complex and diverse urban scenes. *Pedestrians* crossing the road are segmented robustly in (c), *poles*, *traffic signs* and *vegetation* are also identified and localized in (f and i) along with correctly segmenting most commonly occurring classes like *road* and *buildings*.

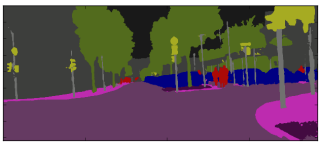
Although the semantic segmentation generated by CloudSeg are able to classify complex urban scenes into diverse classes, they are far from perfect with mistakes and inaccuracies. They are found to be smoothed as the shapes of the individual objects are not highlighted along with segmentation mistakes with adjacent classes. For example the segmentation in (r), a *pedestrian* is merged with adjacent *building* hence its shape is deformed. Keen observation also shows that, objects in a close range are segmented better than far away objects, as CloudSeg rely heavily on depth data. For example in (k), the *poles* and *traffic signs* are merged and smoothed into yellow *traffic sign* prediction because they are too far away to be segmented accurately. The semantic segmentations also show that *vehicles* are over-segmented. The reason of this over-segmentation are the noise in the ground truth due to specular reflective point



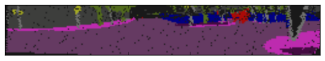
(a) Scene 1: Segmented Camera Image

(b) Scene 1: LiDAR ground truth

(c) Scene 1: CloudSeg prediction

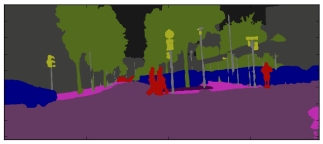


(d) Scene 2: Segmented Camera Image

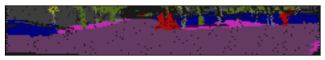


(e) Scene 2: LiDAR ground truth

(f) Scene 2: CloudSeg prediction

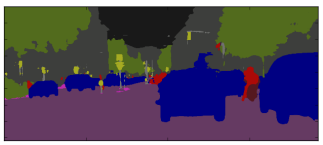


(g) Scene 3: Segmented Camera Image

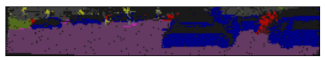


(h) Scene 3: LiDAR ground truth

(i) Scene 3: CloudSeg prediction



(j) Scene 4: Segmented Camera Image



(k) Scene 4: LiDAR ground truth

(l) Scene 4: CloudSeg prediction

(m) Scene 5: Segmented Camera Image

(n) Scene 5: LiDAR ground truth

(o) Scene 5: CloudSeg prediction

(p) Scene 6: Segmented Camera Image

(q) Scene 6: LiDAR ground truth

(r) Scene 6: CloudSeg prediction

Figure 17.: 2D Qualitative Results of CloudSeg

cloud noise and occlusions. In (o), where the *bicycles* in brown and the *vegetation* in green are segmented correctly, CloudSeg segment the *road* as a *vehicle* on the right. In order to improve these mistakes, we propose incorporation latest innovations from recent studies in future work while also introducing post processing of predictions in next section.

5.2.4. Post-processing with confidence threshold

We also introduce some post-processing of the final output, which enables the algorithm to perform reliable perception, even with just the presented proof of concept. As post-processing, we then introduce a confidence threshold over the output of CloudSeg. The output consists of probability map for each class, and we can perform post-processing by choosing a confidence threshold and ignoring all predictions under that confidence threshold. This post-processing ensures that all the predictions, about which the network is not sure should be rejected. These qualitative results with this post-processing are presented in Figure 18.

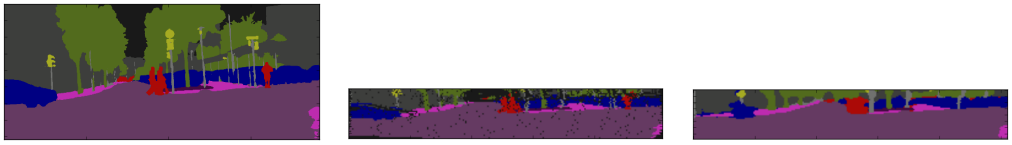
Four individual scenes are presented in this Figure 18, with each scene containing 6 images. Figure 18 (a,g,m,s) are segmented camera images, (b,h,n,t) are LiDAR ground truth and (c,i,o,u) are CloudSeg predictions. Figure 18 (d - f),(j - l),(p - r)and(v - x) are post-processing results with different confidence thresholds for each scene respectively. It is evident that the CloudSeg is not confident about the predictions are the segmentation boundaries and corners of the objects, therefore after post-processing boundaries of different object classes disappear and their segmentation become more distinct.

In post-processing of the first scene, the figures (d - f) represent the confidence thresholds of 0.7, 0.8 and 0.9 respectively. The *poles* and *vegetation* class objects are misclassified as *building* in their background, and with post processing, it can be seen that the prediction for misclassified objects disappear gradually at increasing confidence thresholds. Moreover, entire structures of misclassified and mis-segmented objects can be retrieved. In post-processing of the second scene, the figures (j - l) represent the confidence thresholds of 0.7, 0.8 and 0.9 respectively. A *vehicle* in this scene is mis-segmented as the *building* and even at a confidence threshold of 0.7, the shape of the misclassified objects can be retrieved.

This post-processing also helps with over-segmentation and better shape formation of objects in segmentation. For example, in post-processing of the third scene, the figures (p - r) represent the confidence thresholds of 0.7, 0.8 and 0.9. After post-processing, the shape of the bikes in brown become clearly visible and over-

(a) Scene 1: Segmented Camera Image **(b)** Scene 1: LiDAR ground truth **(c)** Scene 1: Prediction

(d) Scene 1: 0.7 Confidence **(e)** Scene 1: 0.8 Confidence **(f)** Scene 1: 0.9 Confidence



(g) Scene 2: Segmented Camera Image **(h)** Scene 2: LiDAR ground truth **(i)** Scene 2: Prediction

(j) Scene 2: 0.7 Confidence **(k)** Scene 2: 0.8 Confidence **(l)** Scene 2: 0.9 Confidence

(m) Scene 3: Segmented Camera Image **(n)** Scene 3: LiDAR ground truth **(o)** Scene 3: Prediction

(p) Scene 3: 0.7 Confidence **(q)** Scene 3: 0.8 Confidence **(r)** Scene 3: 0.9 Confidence

(s) Scene 4: Segmented Camera Image **(t)** Scene 4: LiDAR ground truth **(u)** Scene 4: Prediction

(v) Scene 4: 0.8 Confidence **(w)** Scene 4: 0.9 Confidence **(x)** Scene 4: 0.99 Confidence

Figure 18.: 2D Qualitative Results of CloudSeg with post-processing

segmentation is removed, as the wrong segmentation of *road* as *vehicle* is rejected. In post-processing of fourth and the last scene, the figures (v - x) represent the confidence thresholds of 0.8, 0.9 and 0.9999 respectively. This high threshold post-processing show that the maximum use of probabilistic reasoning with CloudSeg segmentation, as the over-segmentation due to the noise in the labels is also removed, revealing the intrinsic shape of the *vehicle* in the last scene at high confidence of 0.9999.

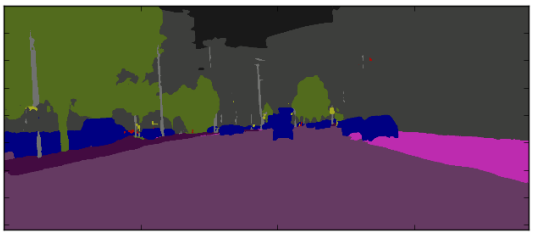
5.2.5. 3D Visualization of Results

In this section, we evaluate CloudSeg in by looking at the predictions from test set in 3D representation for qualitative performance. We also evaluate the effect of post-processing on the algorithm with respect to different confidence thresholds in 3D representation.

Figure 19 and Figure 20 represent the 3D qualitative results and present 3 scenes in total. The first scene is presented by Figure (a - c), where (a) presents 2D segmented camera image and (b) and (c) present CloudSeg segmentation results from two different point of views. It is difficult to judge the performance in the corners and edges through a 3D plot, for example, the horizontal pole shaped objects are classified as *vegetation*, *pole* and *vehicle* and it is difficult to validate them. However, the correct segmentation of *vegetation*, *road*, *sidewalks* and *building* in a broader sense are evident.

The second scene is illustrated through images (d-g) in Figure 19. (d) shows the 2D segmented camera image, (e) shows its corresponding LiDAR ground truth in 3D, (f) shows CloudSeg segmentation result in 3D and (g) shows its 3D representation with confidence threshold of 0.6 where all brown point clouds represent rejected segmentations. The representation of the segmentation in (f) as compared to ground truth in (e) shows similarity, but the further away point clouds are misclassified. These misclassification can be reduced used our proposed post-processing as shown by the point clouds in brown in (g).

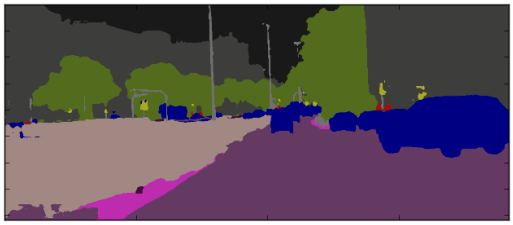
The third scene presents the complex urban scene, with diverse *poles*, *traffic sign*, *vegetation*. Figure 20(a) shows its 2D segmented camera image, (b and c) present corresponding CloudSeg prediction in 2D, without and with post-processing respectively, while (d and e) present their corresponding 3D representation. From (b) to (d), the performance is not translated to 3D perfectly, due to mis-classification between adjacent classes and over segmentation by CloudSeg. However, this is greatly improved from (c) to (e), with the introduction of confidence threshold. The brown point clouds represent the points about which the network is not 85% confident, while



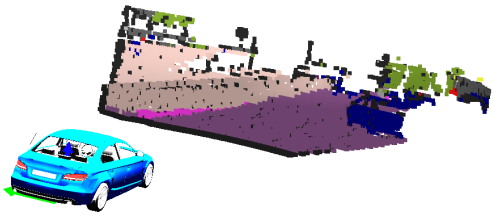
(a) Scene 1: Segmented Camera Image

(b) Scene 1: 3D view 1

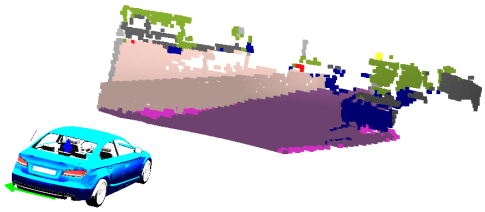
(c) Scene 1: 3D view 2



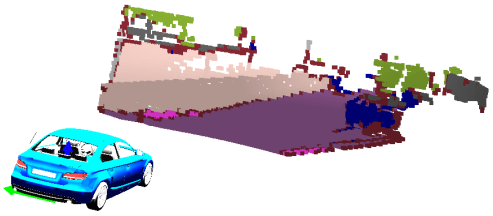
(d) Scene 2: Segmented Camera Image



(e) Scene 2: 3D LiDAR Ground Truth

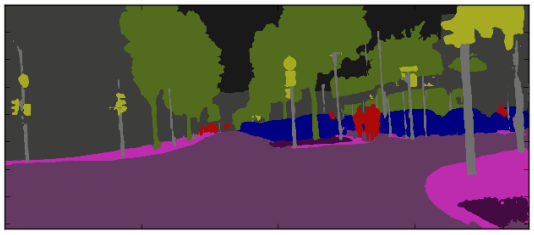


(f) Scene 2: 3D CloudSeg Prediction



(g) Scene 2: 3D CloudSeg Prediction with 0.6 Confidence

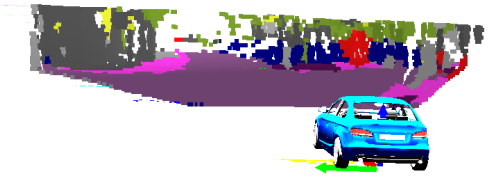
Figure 19.: Qualitative Results in 3D These figures show the qualitative performance of CloudSeg, visualized in 3D.



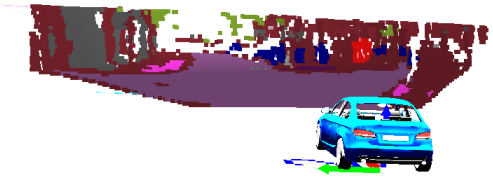
(a) Scene 3: Segmented Camera Image

(b) Scene 3: 2D CloudSeg Prediction

(c) Scene 3: 2D CloudSeg Prediction with 0.85 Confidence



(d) Scene 3: 3D CloudSeg Prediction



(e) Scene 3: 3D CloudSeg Prediction with 0.85 Confidence

Figure 20.: Cont. Qualitative Results in 3D These figures show the qualitative performance of CloudSeg, visualized in 3D.

other predictions give a very informative 3D perception of the environment.

5.3. Quantitative Analysis of CloudSeg Architecture

In this section we present the results of the series of experiments conducted to qualitatively analyze the architecture of CloudSeg. We train few instances of CloudSeg with same training data and training configuration, making some architectural changes to evaluate the effect of that specific change on the overall performance. We study the effects of these design changes in CloudSeg architecture and compare its results with base CloudSeg. Moreover in the following sections, the CloudSeg architecture presented in Chapter 4 will be referred as base CloudSeg while the updated models will be referred as new CloudSeg.

5.3.1. Removing Intensity from the Input

Not all LiDAR sensors are equipped to calculate the reflectance. Moreover the reflectance achieved from LiDAR is highly variable. They are effected by sun exposure and different levels of lightning available in the scene. Reflectance of intensity images in CloudSeg can be understood as vague form of texture information associated with the point cloud depth. In order to evaluate the usefulness of Intensity, we evaluate a model by removing the intensity image from input of CloudSeg. All the other configuration for training is kept the same as for the base CloudSeg. The quantitative results of this model is explained in Table 3.

Accuracy and other metrics for new CloudSeg by removing the intensity, as shown in Table 3, does not show drastic reduction in the performance. The overall accuracy is decreased from 85.7% to 84.1% while precision, Mean IoU and frequency weighted IoU all decrease by 2-3%. The difference of performance being this low, reassures the intuition that CloudSeg should solely depend on depth from LiDAR data and only use intensity as extra texture information to identify objects which are difficult to detect with depth data. Additionally we also compare the per class statistics with base CloudSeg. The per class precision for almost all the classes is decreased by margin as big as 7% for *wall / fence* class. The texture knowledge from *pole*, *people*, *bicycles*, *traffic sign* and *vehicles* improve the overall classification performance. The only class for which the performance does not increase is *road*, which is also the most occurring class in the dataset. This shows that reflectance or intensity helps to differentiate diverse objects within a CNN with uncertain texture information through reflectance. Moreover intensity and depth are different modalities and this experiment validate

Measure	Base CloudSeg	New CloudSeg
Accuracy	85.72%	84.1%
Mean Precision	70.7	67.9
Mean IoU	57.5	54.6
Freq. Weighted IoU	75.9	73.7

(a) Overall Measures

Classes	Base CloudSeg	New CloudSeg
Bicycle	0.57	0.54
Buildings	0.81	0.8
Vehicle (Bus/Car)	0.86	0.84
Fence/Wall	0.71	0.65
Pedestrian	0.66	0.63
Pole	0.34	0.31
Road	0.94	0.94
Sidewalks	0.74	0.68
Terrain	0.74	0.69
Traffic Signs	0.59	0.58
Vegetation	0.83	0.81

(b) Per-class Precision

Table 3.: Evaluation of CloudSeg, with out intensity image in input Table (a) and (b) present the comparisons of overall evaluation and per-class precisions of base CloudSeg and new CloudSeg.

that CNNs can effectively learn combined feature representation with their fusion.

5.3.2. Adding missing data as another Input

The noise due to specular reflection as missing information is a special property of LiDAR data. While this noise is uniformly distributed through out the laser scans, it is specially concentrated at edges of objects. For example around the edges of poles or trees (*vegetation*) or the missing data though the glasses of *vehicles*, in the scene. In a way, this missing data can be seen as informative about the shapes of the objects. To evaluate the usefulness of this noise it is added as another channel in the input as a map of missing point clouds to CloudSeg, to see if such correlation helps the scene understanding.

Table 4 compares its results with base CloudSeg. The accuracy, precision, mean IoU and frequency weighted IoU do not show any improvement in the overall performance.

Measure	Base CloudSeg	New CloudSeg
Accuracy	85.72	85.4
Mean Precision	70.7	70.2
Mean IoU	57.5	54.6
Freq. Weighted IoU	75.9	75.8

(a) Overall Measures

Classes	Base CloudSeg	New CloudSeg
Bicycle	0.57	0.56
Buildings	0.81	0.81
Vehicle (Bus/Car)	0.86	0.85
Fence/Wall	0.71	0.69
Pedestrian	0.66	0.67
Pole	0.34	0.33
Road	0.94	0.95
Sidewalks	0.74	0.71
Terrain	0.74	0.71
Traffic Signs	0.59	0.6
Vegetation	0.83	0.84

(b) Per-class Precision

Table 4.: Evaluation of CloudSeg, with missing point cloud noise in input
Table (a) and (b) present the comparisons of overall evaluation and per-class precisions of base CloudSeg and new CloudSeg.

However, the performance is decreased, but the decline is insignificant in quantity. We also present the per-class statistics of new CloudSeg compared to base CloudSeg. As expected the precision of *pedestrian*, *road*, *traffic signs*, and *vegetation* is increased by very small margin, while the precision for other classes have decreased or remained same. The overall analysis suggest that its addition could be useful for the detection and segmentation for certain classes, but for an eleven class problem it takes more than it adds to the performance of CloudSeg. Moreover, depth, intensity and missing data are all different modalities and are represented at different scales in input. Learning representations through them is not trivial with a CNN, while most of the missing point clouds would disappear during the pooling stages in CloudSeg.

Measure	Base CloudSeg	New CloudSeg
Accuracy	85.72	85.8
Mean Precision	70.7	71.5
Mean IoU	57.5	58.0
Freq. Weighted IoU	75.9	76.4

(a) Overall Measures

Classes	Base CloudSeg	New CloudSeg
Bicycle	0.57	0.61
Buildings	0.81	0.83
Vehicle (Bus/Car)	0.86	0.88
Fence/Wall	0.71	0.69
Pedestrian	0.66	0.66
Pole	0.34	0.37
Road	0.94	0.94
Sidewalks	0.74	0.71
Terrain	0.74	0.75
Traffic Signs	0.59	0.62
Vegetation	0.83	0.82

(b) Per-class Precision

Table 5.: Evaluation of CloudSeg, with 3 channel Jet Encoding of depth image in input Table (a) and (b) present the comparisons of overall evaluation and per-class precisions of base CloudSeg and new CloudSeg.

5.3.3. Encoding LiDAR depth into 3-channels (Jet encoding)

A number of studies in literature use depth data, such as diffusion images, for classification and segmentation of RGB image data through CNNs. These CNNs can be divided into two groups, early and late fusion approach. In early fusion approach, both depth and RGB data are channel wise concatenated before learning representation on them. On the contrary, late fusion techniques, separately learn feature representation from depth and RGB data and merge their abstract representation in their CNN architecture. Late Fusion approaches have shown better performance as compared to early fusion approaches [49]. These late fusion approaches tend to encode the depth data in three channels, making it more informative and similar to 3 channel image data, that shows better results than one channel depth image. The most used approach for encoding is *HHA* that uses horizontal disparity, height above ground, and the angle of the local surface normal with the inferred gravity. Eitel et al. introduce another approach by computing *Jet* color map representation of disparity depth image that outperforms *HHA* [49]. To evaluate the effectiveness of 3 channel encoding of depth data, we encode our LiDAR depth maps into three channels with Jet color encoding and analyze the performance with respect to base CloudSeg.

Table 5 compares the semantic segmentation results of new CloudSeg with Jet encoding of LiDAR depth images and base CloudSeg. The overall accuracy, mean precision, mean IoU and frequency weighted IoU for new CloudSeg is improved. When depth is encoded in three channels via Jet color-map, the third spatial dimension is represented in three channels, which makes it easier for the network to learn the representation. Although an increase in performance is observed, the scale of the improvement is insignificant with the overhead of preprocessing required to convert LiDAR depth maps to Jet encodings. The per-class precisions, reported in Table 5(b), show improvement in classes like *pole*, *bicycle* and *traffic signs*. Jet encoding can efficiently represent farther away objects with introduction of 3 spatial dimension through Jet encoding and therefore can improve the classification of *poles*, *bicycles* and *traffic sign* by small margins. Overall, this improvement comes at the cost of preprocessing of depth images and in order to keep the model simple, this Jet encoding is not made part of base CloudSeg.

5.3.4. Weighted Loss

Weighted cross entropy loss is an important design component of CloudSeg. In chapter 4, we presented the intuition beside using weighting and what it adds to the

overall architecture. In this section, we present its effectiveness with empirical results. Therefore, we train the same network architecture using cross-entropy softmax loss with equal weights for every class. The weighting used for training base CloudSeg doubles the loss for *pedestrian*, *poles* and *traffic sign* class.

Measure	Base CloudSeg	New CloudSeg
Accuracy	85.72	85.9
Mean Precision	70.7	68.9
Mean IoU	57.5	56.7
Freq. Weighted IoU	75.9	76.1

(a) Overall Measures

Classes	Base CloudSeg	New CloudSeg
Bicycle	0.57	0.63
Buildings	0.81	0.83
Vehicle (Bus/Car)	0.86	0.86
Fence/Wall	0.71	0.74
Pedestrian	0.66	0.53
Pole	0.34	0.21
Road	0.94	0.94
Sidewalks	0.74	0.75
Terrain	0.74	0.74
Traffic Lights	0.59	0.48
Vegetation	0.83	0.87

(b) Per-class Precision

Table 6.: Evaluation of CloudSeg, with equal weight for every class in Loss Function. Table (a) and (b) present the comparisons of overall evaluation and per-class precisions of base CloudSeg and new CloudSeg.

Table 6(a) compare the Accuracy, mean precision, mean IoU and frequency weighted IoU of new CloudSeg with base CloudSeg. We see overall measure show improvement in results, as these metrics are dominated by classes which occur more in dataset. In order to really analyze the use of weighting, the per-class precision is presented in Table 6(b). The precision of *pedestrian*, *pole* and *traffic sign* is decreased by huge margins. A very simple weighting strategy drastically improves the performance of base CloudSeg and help it to generalize. Since the goal of this study is to a show a proof of concept, the more combinations of weighting were not explored, but the overall benefit is shown with the results.

5.3.5. Shortcut Connections

Shortcut connections are a heavily featured design choice in many modern CNN architectures. Huang et al. take it a step further in the so called Densely Connected Convolutional Network, in which all layers of the CNN are connected with all the previous convolution layers [56]. In this section, we evaluate these shortcut connection with regard to LiDAR data and CloudSeg. The goal of shortcut connections is to merge predictions from different levels to aid upsampling and decrease the overall depth of early convolutional layers to learn better feature representation.

Measure	Base CloudSeg	New CloudSeg
Accuracy	85.7	85.8
Mean Precision	70.7	71.0
Mean IoU	57.5	57.8
Freq. Weighted Iou	75.9	76.3

(a) Overall Measures

Classes	Base CloudSeg	New CloudSeg
Bicycle	0.57	0.59
Buildings	0.81	0.82
Vehicle (Bus/Car)	0.86	0.86
Fence/Wall	0.71	0.7
Pedestrian	0.66	0.65
Pole	0.34	0.34
Road	0.94	0.95
Sidewalks	0.74	0.73
Terrain	0.74	0.72
Traffic Signs	0.59	0.62
Vegetation	0.83	0.84

(b) Per-class Precision

Table 7.: Evaluation of CloudSeg, without shortcut connections. Table (a) and (b) present the comparisons of overall evaluation and per-class precisions of base CloudSeg and new CloudSeg.

Table 7(a) shows that overall measures of new CloudSeg without Shortcut connections is almost similar. This observation is surprising because the removal of shortcut connection would increase the depth of early layers and learning good features would be difficult. No decrease in performance indicates that the overall depth of the CloudSeg allow learning of good representation features without shortcut

connections. Table 7(b) shows improved per class precision for *bicycle*, *building*, *road*, *traffic sign* and *vegetation* classes as compared to base CloudSeg. This improvement in performance shows that the amalgamation from early layers can also cause loss of precision as more abstract representations are not developed in early stages. However, this experiment shows that, CloudSeg without shortcut connections can also learn good representations from LiDAR data, but due to strong precedent in literation and intuition shortcut connections are kept are as an essential part of CloudSeg architecture. In order to fully utilize shortcut connections, we also validate, two-step learning scheme in next subsection.

5.3.6. Effects of using different learning Scheme

In order to fully utilize and efficiently use shortcut connections, we try a two stage learning scheme. The major role of shortcut connections in our architecture is to use low level representations to enhance upsampling. To supplement the representations learned by early layers following two stage scheme is used. In first stage, the $FC5$ and $FC6$ layers are removed and the network is trained for around 10 epochs only using the shortcut connections. By doing so we decrease the overall depth of the network, which would help to learn to good representations from the data through shortcut connection. In second stage, we train the network in its full architectural configuration and fine-tune the already learned feature representation. We compare the effect of this two-step scheme learning with in single step training of base CloudSeg in this section.

The intuition behind the development of this learning scheme derives from the fact, that many deep CNNs finetune pre-trained models, and shortcut connection transfer representations from early pretrained components of CNN to more abstract and less deeper classification or/and upsampling layers. Following the intuition, the two-step learning scheme was able to successfully identify feature representation to aid the classification of less occurring classes like *poles*, *pedestrians* and *vegetation* with a version of CloudSeg that did not use weighted Loss function. Table 8(a) shows the overall performance metric, of new CloudSeg training with two-step learning scheme and base CloudSeg. The accuracy, mean precision, mean IoU and frequency weighted IoU of both models show very similar performance. Similarly the per class precision only improve performance of three classes, namely *bicycle*, *buildings* and *fence / wall*, while for all other classes the performance is almost same or little worse. All in all, it is observed that two-step learning scheme did not add a lot to base CloudSeg, which already use weighted loss to properly segment less occurring classes. Two-step

Measure	Base CloudSeg	New CloudSeg
Accuracy	85.7	84.9
Mean Precision	70.7	70.0
Mean IoU	57.5	56.1
Freq. Weighted IoU	75.9	75.0

(a) Overall Measures

Classes	Base CloudSeg	New CloudSeg
Bicycle	0.57	0.6
Buildings	0.81	0.88
Vehicle (Bus/Car)	0.86	0.82
Fence/Wall	0.71	0.72
Pedestrian	0.66	0.65
Pole	0.34	0.3
Road	0.94	0.94
Sidewalks	0.74	0.74
Terrain	0.74	0.7
Traffic Signs	0.59	0.58
Vegetation	0.83	0.77

(b) Per-class Precision

Table 8.: Evaluation of CloudSeg, with two stage learning scheme. Table (a) and (b) present the comparisons of overall evaluation and per-class precisions of base CloudSeg and new CloudSeg.

learning scheme is an effective tool for the training of CNNs with class skew, but in case of CloudSeg its effect is neutralized by weighted loss.

5.3.7. Effects of Learned Up sampling

The choice of upsampling is also an important aspect of our presented CNN architecture. In our model, we train the deconvolution layers by fine-tuning the bilinear interpolation filter. In this section, we evaluate the effectiveness of this fine-tuning, in other words, we replace the original model's upsampling by bilinear interpolation and compare the results with the original. The aim of fine-tuning the bilinear interpolation deconvolution, is to learn individual upsampling interpolation kernels for every class. If the kernels are not fine-tuned, then all the classes are upsampled through bilinear interpolation.

Table 9 shows the comparison of base CloudSeg with new CloudSeg with upsampling

Measure	Base CloudSeg	New CloudSeg
Accuracy	85.7	85.5
Mean Precision	70.7	70.6
Mean IoU	57.5	57.2
Freq. Weighted IoU	75.9	75.9

(a) Overall Measures

Classes	Base CloudSeg	New CloudSeg
Bicycle	0.57	0.57
Buildings	0.81	0.81
Vehicle (Bus/Car)	0.86	0.85
Fence/Wall	0.71	0.71
Pedestrian	0.66	0.68
Pole	0.34	0.33
Road	0.94	0.94
Sidewalks	0.74	0.74
Terrain	0.74	0.7
Traffic Signs	0.59	0.61
Vegetation	0.83	0.84

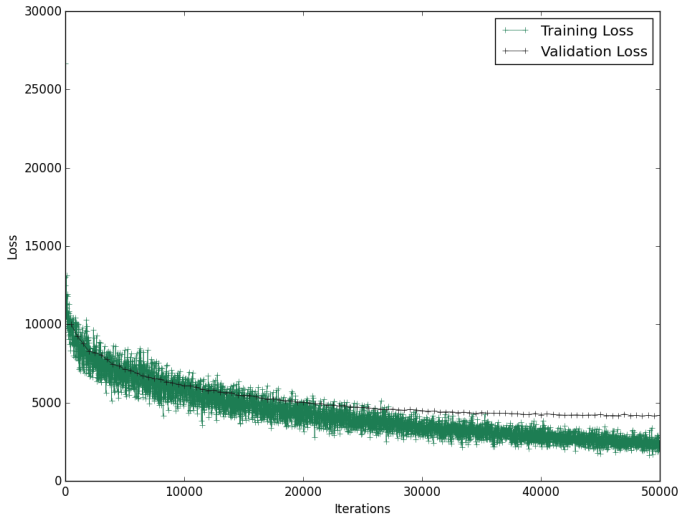
(b) Per-class Precision

Table 9.: Evaluation of CloudSeg, with upsampling through bilinear interpolation. Table (a) and (b) present the comparisons of overall evaluation and per-class precisions of base CloudSeg and new CloudSeg.

through bilinear interpolation. The decrease in overall accuracy, mean precision and mean IoU is observed, as shown in Table 9(a), however the margin of reduction of performance is also insignificantly small. This shows that, learning of upsampling in CloudSeg, is only responsible for a small margin of overall performance and improves the classification performance on the edges of objects, which results in well formed shape of the individual class objects in any scene. Table 9(b) shows the per-class precision comparison between new CloudSeg and base CloudSeg. It is observed, that precision of *vehicle*, *pedestrian*, *pole*, *terrain* and *traffic sign* is decreased, without fine-tuning of bilinear interpolation. The classes for which the performance is decreased all share the property of being diverse in shape and geometry. This fulfills the intuition, of learning individual class-level upsampling deconvolution filters.

5.3.8. Dropouts

Dropouts are an important component of convolutional neural networks. They are very effective regularization method and also act as an implicit data augmentation within the network. It is further explained in detail in appendix A. This section evaluates the effectiveness of dropouts by training the same CloudSeg without dropout units and then comparing the training curves with the original architecture.



(a)

Figure 21.: Learning curve of CloudSeg without Dropouts This plot shows the training and validation loss of CloudSeg without dropouts. Training loss is in green and validation set loss is in black. Around 30000 iterations, signs of overfitting begin to appear.

Figure 21(a) shows the training and validation loss over 50000 iterations of training of CloudSeg without dropouts. It is observed that the loss over training and validation set remain in overlapping range some 15000 iteration, after which the validation and training loss start to diverge. This divergence of training and validation curve shows over-fitting. Compared to base CloudSeg, signs of over-fitting does not appear until 50000 iteration, showing generalizing ability of models with dropouts. Given this experiment, dropouts prove to be useful component of CNN to prevent over-fitting LiDAR point cloud data.

6. Conclusion and Future Work

This section presents the conclusion of this thesis while pointing out the major areas of future work.

6.1. Conclusion

The aim of this thesis was to study the application of deep convolutional neural networks with the LiDAR point clouds by presenting a proof of concept of semantic point cloud segmentation. While computer vision and natural language processing are going through a transformation due to deep learning, similar breakthrough has not been seen with methods that process 3D data. This lack of progress is due to the computational burden of 3rd spatial dimension and lack of adequate annotated benchmark datasets. This thesis conducts concrete literature review of the topic and presents an imperfect dataset for end-to-end training of deep networks on LiDAR point clouds. In addition, to facilitate the application of conventional CNNs, the LiDAR point clouds are encoded in 2D panorama images.

In order to show the proof of concept, we train a CNN, CloudSeg, for point-wise semantic segmentation of LiDAR point clouds. Unlike any existing study, CloudSeg performs novel 11 class semantic segmentation of 3D point clouds and is able to explain roughly 85% of the point clouds in each scan. Moreover, the qualitative analysis of CloudSeg is presented, showing robust 2D and 3D segmentations results and individual class detectors as well as a trivial post-processing technique for practical use. This thesis also evaluates certain design features of CloudSeg in order to analyze and identify effective design principles that help in the designing CNNs for 3D point clouds.

The work presented in this thesis, has huge potential to be used in autonomous driving and advance driver assistance systems (ADAS). With more accurately annotated data, CloudSeg can be trained to perform intricate perception tasks in complex urban environments. Moreover, CloudSeg can be adapted to perform perception in other robotics domains as well.

6.2. Future Work

With the work presented in this thesis, there is a huge potential of future work. The lacking of benchmark dataset with accurate point-wise semantic annotations of point clouds is an important area. Being a proof of concept, architecture of CloudSeg is kept simpler, by preventing the use of many new innovation that feature in recent CNNs. Given a dataset with accurate point-wise annotations, future work can be conducted in following areas:

- Integration of CNN components like dilated convolutions, residual blocks, pyramid pooling and other innovations in CloudSeg.
- Similar to RGB-D classifications, the combination of video camera and LiDAR point clouds in a deep CNN for semantic point cloud segmentation is another area.
- Point clouds are often validate on a sequence of scans rather than a single scan. This provides opportunity to exploit temporal relationship between consecutive LiDAR scans with a recurrent neural network.

7. Acknowledgment

I would like to thank my thesis advisors Dr. Claudius Glaeser and Dr. Volker Fischer of Robert Bosch GmbH, for their support, guidance and technical help throughout the thesis work. I would also like to thank my academic advisor and examiner Dr. Joschka Boedecker of the Machine Learning Group at University of Freiburg, for his invaluable guidance and input during our work. Their passionate participation and input, drive the innovations employed in this study.

I would also like to thank my colleagues at Robert Bosch GmbH, with whom I shared the working space during the course of this research project, for their assistance and appreciation of my work. Moreover, I would like to thank Ulrich Baumann and Tayyab Naseer for the insights I received through our discussions on the thesis.

Finally, I must express my very profound gratitude to my spouse, Anam, for the support, care and invaluable insight on my work, throughout the course of this thesis. I would also like to mention my parents for providing me with their unfailing support and encouragement throughout my years of study. This accomplishment would not have been possible without them.

Farooq Ahmed Zuberi

Bibliography

[1] D. Schacter, *Psychology*. Worth Publishers., 2011.

[2] A. Serov, "Subjective reality and strong artificial intelligence," 2013.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015. Insight.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[5] J. Kim, "Awesome deep vision." `https://github.com/kjw0612/awesome-deep-vision`.

[6] A. P. Cracknell, *Introduction to remote sensing*. CRC press, 2007.

[7] S. Taranovich, "Autonomous automotive sensors: How processor algorithms get their inputs," 2016. [Online; accessed 3-February-2017].

[8] "Velodyne LiDAR hdl-64e." `http://velodynelidar.com/hdl-64e.html`. Accessed: 2017-02-17.

[9] E. Baltsavias, "Airborne laser scanning: Basic relations and formulas," *ISPRS Journal of Photogrammetry & Remote Sensing*, vol. 56, pp. 199–214, 1999.

[10] M. Lehtomäki, A. Jaakkola, J. Hyyppä, J. Lampinen, H. Kaartinen, A. Kukko, E. Puttonen, and H. Hyyppä, "Object classification and recognition from mobile laser scanning point clouds in a road environment," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, 2016.

[11] A. Pilehvar, "Object Classification using 3D LiDAR Point Clouds," Master's thesis, Friedrich Alexander University, Erlangen, Germany, 2015.

[12] v. d. H. F. V. G. Rabbani, T., "Segmentation of point clouds using smoothness constraint," 2006.

[13] G. Sithole and G. Vosselman, "Automatic structure detection in a point-cloud of an urban landscape," *2nd GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas*, pp. 67–71, 2003.

[14] F. Moosmann, O. Pink, and C. Stiller, "Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion," *IEEE Intelligent Vehicles Symposium*, p. 215–220, 2009.

[15] A. Golovinskiy and T. Funkhouser, "Min-cut based segmentation of point clouds," *IEEE Workshop on Search in 3D and Video (S3DV) at ICCV*, pp. 39–46, 2009.

[16] J. Schoenberg, A. Nathan, and M. Campbell, "Segmentation of dense range information in complex urban scenes," *Int. Conf. on Intellig. Robots and Systems (IROS)*, p. 2033–2038, 2010.

[17] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng, "Discriminative learning of markov random fields for segmentation of 3d scan data," *IEEE Conf. on CVPR*, vol. 2, p. 169–176, 2005.

[18] D. Munoz, N. Vandapel, and M. Hebert, "Onboard contextual classification of 3-d point clouds with learned high-order markov random fields," *IEEE Int. Conf. on Robotics and Automation*, pp. 2009–2016, 2009.

[19] O. Elberink, S., and B. Kemboi, "User-assisted object detection by segment based similarity measures in mobile laser scanner data," *ISPRS - Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XL-3*, pp. 239–246, 2014.

[20] A. Aijazi, P. Checchin, and L. Trassoudaine, "Segmentation based classification of 3d urban point clouds: A super-voxel based approach," *Remote Sensing 5*, vol. 5, p. 1624–1650, 2013.

[21] A. Serna and B. Marcotegui, "Detection, segmentation and classification of 3d urban objects using mathematical morphology and supervised learning," *Journal of Photogrammetry and Remote Sensing 93(0)*, pp. 243–255, 2014.

[22] J. Secord and A. Zakhor, "Tree detection in urban regions using aerial lidar and image data," *Geoscience and Remote Sensing Letters, IEEE 4(2)*, p. 196–200, 2007.

[23] J. Lalonde, R. Unnikrishnan, N. Vandapel, and M. Hebert, "Scale selection for classification of point-sampled 3d surfaces," *5th Int. Conf. on 3-D Digital Imaging and Modeling*, p. 285–292, 2005.

[24] N. Chehata, L. Guo, and C. Mallet, "Airborne lidar feature selection for urban classification using random forests," *The Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 38(3)*, p. 207–212, 2009.

[25] S. Lodha, D. Fitzpatrick, and D. Helmbold, "Aerial lidar data classification using adaboost. in: 3-d digital imaging and modeling," *6th Int. Conf. on 3DIM'07*, p. 435–442, 2007.

[26] D. Munoz, J. Bagnell, N. Vandapel, and M. Hebert, "Contextual classification with functional max-margin markov networks," *IEEE Conf. on CVPR*, p. 975–982, 2009.

[27] L. Linsen and H. Prautzsch, "Global versus local triangulations," *Proc. of Eurographics*, pp. 257–263, 2001.

[28] I. Lee and T. Schenk, "Perceptual organization of 3d surface points," *The Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 193–198, 2002.

[29] S. Filin and N. Pfeifer, "Segmentation of airborne laser scanning data using a slope adaptive neighborhood," *Journal of Photogrammetry and Remote Sensing 60(2)*, p. 71–80, 2006.

[30] R. Shapovalov, A. Velizhev, and O. Barinova, "Non-associative markov networks for 3d point cloud classification," *Photogrammetric Computer Vision and Image Analysis (PCV 2010)*, p. 103–108, 2010.

[31] J. Niemeyer, F. Rottensteiner, and U. Soergel, "Conditional random fields for lidar point cloud classification in complex urban areas," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences I-3*, p. 263–268, 2012.

[32] D. Dohan, B. Matejek, and T. Funkkhouser, "Learning hierarchical semantic segmentations of lidar data," *International Conference on 3D Vision (3DV)*, pp. 273–281, 2015.

[33] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3d point clouds with strongly varying density," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 3, p. 12–19, 2016.

[34] A. Aijazi, A. Serna, B. Marcotegui, P. Checchin, and L. Trassoudaine, "Segmentation and classification of 3d urban point clouds: Comparison and combination of two approaches," *Field and Service Robotics*, vol. 113, pp. 201–216, 2016.

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

[36] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016.

[37] M. Thoma, "A survey of semantic segmentation," 2016.

[38] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.

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

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for largescale image recognition," 2014.

[41] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, 2015.

[42] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016.

[43] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *CoRR*, vol. abs/1612.01105, 2016.

[44] Z. Wu, C. Shen, and A. van den Hengel, "Wider or deeper: Revisiting the resnet model for visual recognition," *CoRR*, vol. abs/1611.10080, 2016.

[45] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," *CoRR*, vol. abs/1609.06666, 2016.

[46] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," *IROS*, p. 922–928, 2015.

[47] . Çiçek, A. Abdulkadir, S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (M. S. G. U. S. Ourselin, W.S. Wells and L. Joskowicz, eds.), vol. 9901 of *LNCS*, pp. 424–432, Springer, Oct 2016. (available on arXiv:1606.06650 [cs.CV]).

[48] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *CoRR*, vol. abs/1608.07916, 2016.

[49] A. Eitel, J. T. Springenberg, L. Spinello, M. A. Riedmiller, and W. Burgard, "Multimodal deep learning for robust RGB-D object recognition," *CoRR*, vol. abs/1507.06821, 2015.

[50] A. Valada, G. Oliveira, T. Brox, and W. Burgard, "Towards robust semantic segmentation using deep fusion,"

[51] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[52] G. L. Oliveira, W. Burgard, and T. Brox, "Efficient deep models for monocular road segmentation," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 4885–4891, IEEE, 2016.

[53] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[54] G. Larsson, "caffe weighted samples." `https://github.com/gustavla/caffe-weighted-samples`.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine

learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[56] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[57] D. Anderson and G. McNeill, "Artificial neural networks technology," *Kaman Sciences Corporation*, vol. 258, no. 6, pp. 1–83, 1992.

[58] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[59] A. Krishnan and J. Larsson, "Vehicle detection and road scene segmentation using deep learning," Master's thesis, Chalmers University of technology, Gothenburg, Sweden, 2016.

[60] S. Haykin and N. Network, "A comprehensive foundation," *Neural Networks*, vol. 2, no. 2004, p. 41, 2004.

[61] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

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

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

[64] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

A. Theory of Deep Artificial Neural Network

A.1. Artificial Neural Network

A biological neural network is a set of connected, interacting nerve cells. These networks transmit electrical signals from some neurons through axon terminals of the synapses (output) to the dendrites (input) of other neurons. Each of these neurons have an activation threshold. The sum of all input signals must surpass this threshold in order the neuron to fire and transmit the signal through the network. Biological neural networks is the human brain roughly consist around 10^{10} neurons, each connected to 10^4 other neurons [57].

Artificial neural networks (ANN) are simulated networks, inspired by biological neural networks. The connection strength is modeled with a numeric weight between the neurons and the activation threshold is encoded by a numeric bias. Additionally, each neuron is associated with a nonlinear activation function. The network of neurons are used as a non-linear function, by setting some neurons as input neurons and output neurons. An artificial neural network (ANN) can have any number of input and output neurons, which makes it a highly diverse operational structure, used in numerous ways for many diverse tasks.

A.2. Deep Feed-forward Networks

Deep feed-forward networks are a set of modeling tools, which have been successful employed in pattern recognition, classification, regression and other tasks generic to machine learning [58]. The information in a feed-forward network flows along a single direction, without any self-connections or feedbacks. The network is arranged in layers, having dedicated neurons in input layer, output layer and some hidden layer. Given the input, weights and biases of all the layers, this provides a systematic method of calculating the activations of the output layer.

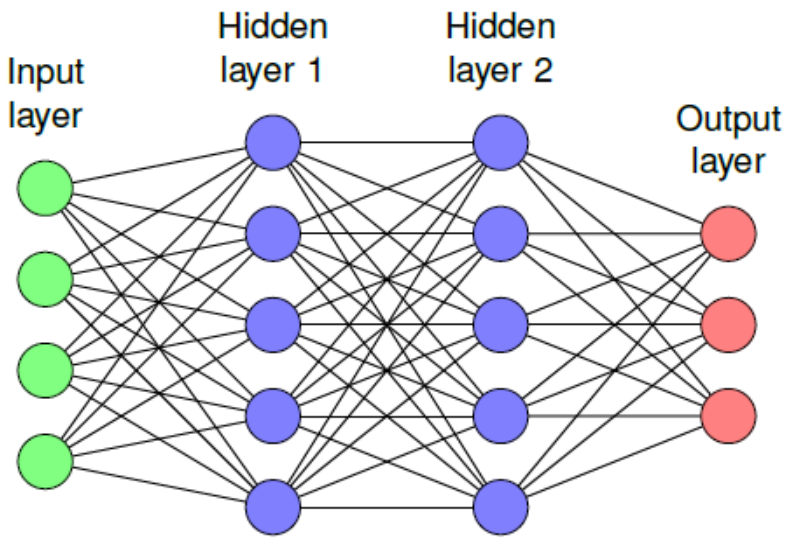


Figure 22.: Example of a simple feed forward neural network. Courtesy [59].

An example of a feed-forward neural network is presented in figure 22 with *fully connected layers*. Mathematically, the process of feeding information through a network with fully connected layers is represented by a series of matrix-vector multiplications and element-wise activation functions. Let x be the input arranged in an N dimensional vector, W_1 be the matrix of weights $w_{i,j}$ connecting input neuron j with hidden neuron i, b_1 be a vector with the biases of the neurons in the first hidden layer and σ_1 be an activation function. Then, the output vector a_1 from the first hidden layer is given by

$$a_1 = \sigma_1(W_1 x + b_1) \tag{7}$$

The output vector a_1 will act as the input to the second hidden layer of the network. Analogously, the output of n^{th} layer is given by:

$$a_n = \sigma_n(W_n a_{n-1} + b_n) \tag{8}$$

A deep feed-forward neural network is simply a feed-forward network with many layers. As the information flows through the layers, it forms a rich hierarchical representation of the input features. Hence, it can be seen as an artificial neural network with a high modeling capacity and allows greater nonlinearity mapping.

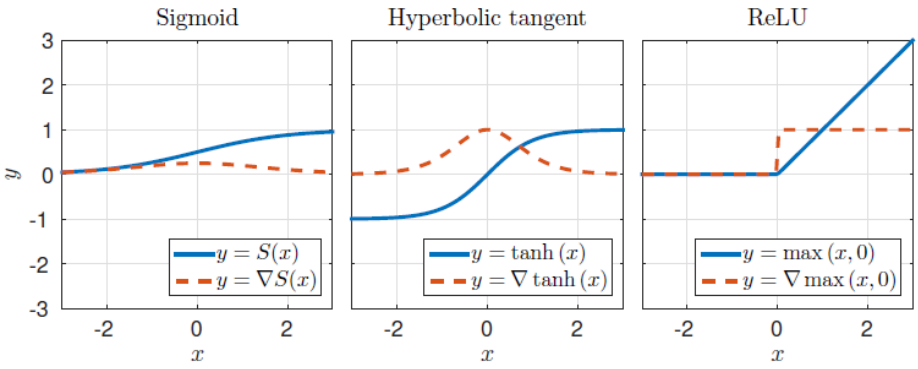


Figure 23.: Activation Functions The sigmoid, hyperbolic tangent and ReLU activation functions and their derivatives. Courtesy [59].

A.2.1. Activation Function

The activation functions are essential components of ANNs. They are used to perform nonlinear mappings of the input data and are typically applied element-wise to all neurons in a hidden layer. This section describes the most commonly used activation functions and their properties (See figure 23).

Sigmoid

The sigmoid activation function is given by

$$S(x) = \frac{1}{1 + \exp^{-x}} \tag{9}$$

It takes values from 0 to 1 and is linear close to the origin. It has a squashing property such that large positive or negative input values result in values close to 1 or 0 respectively.

Hyperbolic Tangent

The hyperbolic tangent is given by

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{\exp^{x} - \exp^{-x}}{\exp^{x} + \exp^{-x}} \tag{10}$$

It is also commonly used as activation function in ANNs. It has similar properties to the sigmoid function except $\tanh(x)$ takes values from -1 to 1.

Rectified Linear Unit

The Rectified Linear Unit (ReLU) is an activation function commonly used with deep neural networks, and is given by

$$f(x) = \max(x, 0) \tag{11}$$

Compared to the commonly used sigmoid activation function, the ReLU has the advantage of not saturating for the large inputs. While sigmoidal functions take values in the range $(0, 1)$, ReLUs take values in $[0, 1)$ making them less prone to be either on or off.

Softmax

The softmax function is a normalizing function commonly used in the last layer of a neural network. Given an input vector x with values x_i, the corresponding output element y_i is calculated

$$y_i = \frac{\exp^{x_i}}{\sum_{j-1}^{N} \exp^{x_j}} \tag{12}$$

By construction, the elements of the Softmax output vector sums to 1. This property is useful as the output of the neural network can interpreted as probabilities for different cases, such as in classification tasks.

A.2.2. Back propagation

To make the output of a neural network a useful mapping of its input, a common strategy is to train its parameters using supervised learning. This requires having predetermined desired outputs for given input data, which are compared to the actual output of the ANN. The desired output y, along with the actual output $\hat{y}$, is passed to a differentiable *Loss function* C, which is minimized by tuning the parameters of the network.

Let Θ be the set of all parameters in the network; then the objective when training in a supervised manner is to minimize

$$\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} C(y_i, \hat{y_i}|\Theta) \tag{13}$$

where $\hat{y_1} \dots \hat{y_N}$ and $y_1 \dots y_N$ are the network outputs and labels corresponding to

input training data $x_1 \dots x_N$. The process of passing the data through the network, calculating the loss and tunning the parameters continues until the network has acceptable performance on the validation dataset.

For a feed-forward neural network, the method used for adjusting the parameters is referred to as *back-propagation*. With gradient descent, back-propagation propagates the gradients of the loss function with respect to the parameters back through the network using the chain rule [60].

Stochastic Gradient Descent

While only one sample of input data, desired output and the actual output is required to calculate the gradients for all parameters of the network, in common practice several samples are included and an average of obtained gradients is used. The set of samples used for a single update of weights is called a *mini-batch*. Training networks using mini-batches is beneficial in several ways. An average of gradients is a better approximation to the gradients of the minimization problem (equation 13), which includes gradients over the entire training set. Moreover, computing the gradients of several inputs at the same time is typically more efficient when using the high level of parallelism available in modern computers and GPUs.

Stochastic Gradient Descent (SGD) is the process of randomly selecting samples from the training set, computing the gradients and then updating the parameters as

$$\Theta \leftarrow \Theta - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial C(y_i, \hat{y}_i|\Theta)}{\partial \Theta} \tag{14}$$

where α is the learning rate and m is the mini-batch size.

Typically, the amount of training done is measured in *epochs*, defined as the number of times all training samples have been used to update the network parameters. If the number of available training samples is N, one epoch is completed after using N/m mini-batches for training. If the input data is subject to some augmentation before being passed to the network, the number of training samples actually used might be intractable and thus the measure of one epoch cannot be defined as in the original sense. In such cases, the amount of training done is measured directly by the number of mini-batches used.

Adam

Similar to stochastic gradient descent, Adam is another gradient based optimization technique [61]. While the use of mini-batch can often result in noisy estimations of the gradients, Adam relies on the adaptive moment estimation (m_t, v_t) of the gradients over the mini-batches, for robust estimation of true gradients. It can also be regarded as a generalization of AdaGrad (Another optimization technique). The adaptive moments are updated as follows:

$$(m_t)_i = \beta_1(m_{t-1})_i + (1-\beta_1)(\nabla L(W_t))_i, \tag{15}$$

$$(v_t)_i = \beta_2(v_{t-1})_i + (1-\beta_2)(\nabla L(W_t))_i^2 \tag{16}$$

and accordingly the weight update equation changes as follows:

$$(W_{t+1})_i = (W_t)_i - \alpha \frac{\sqrt{1-(\beta_2)_i^t}}{1-(\beta_1)_i^t} \frac{(m_t)_i}{\sqrt{(v_t)_i} + \epsilon} \tag{17}$$

Adam introduces 3 additional hyperparameters and Kingma et al. proposes to use $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ as default values, which work fairly well in practice. As compared to stochastic gradient descent, Adam requires less hyper parameter optimization. It is also used in many recent studies to train deep neural networks.

A.3. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are feed-forward neural networks with a layout and architecture specifically designed to handle data arranged in a spatial grid, such as $2D$ or $3D$ images. The inspiration of the architecture comes from the mechanism of biological visual perception [57]. The networks, like any other ANN, are composed of neurons with learn-able weights and biases. Each neuron receives some inputs, performs a dot product. The architecture is typically composed of several layers, which gives them the characterization of being *deep* and thus research work on CNNs fall under the domain of deep learning. Essentially, the network computes a mapping function that relates image pixels to a final desired output.

In a general CNN, the input is assumed to be a RGB image, i.e. consisting of three channels, corresponding to the red, green and blue color intensity values. Consecutive layers of the CNN may consist of even more channels referred to as feature maps. The number of feature maps typically increase through the layers of a CNN, while the spatial dimension of them decreases until reaching the desired output size. The

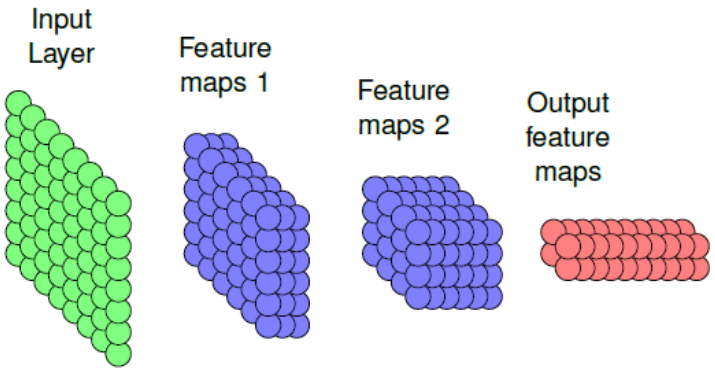


Figure 24.: Structure of Convolutional Neural Network(CNN) Illustration of how neurons are structured in a CNN. This simple example shows a 8×8 single channel (gray-scale) image input and the arrangement of intermediate feature maps of the network. Connections between neurons have been removed for clarity. Courtesy [59].

over-all idea behind this structure is that the representation of the input image is gradually increased in abstraction as it progresses through the layers. Later layers contain more information about the "what" and "how" of objects and things in an image, and less of "where". Similar to the definition of a feature map, a feature vector at a specific layer of a CNN is defined to be the elements across all feature maps at a given spatial location. An example of the structure of a CNN is presented in

A.3.1. Components of CNNs

Besides the fully connected layer presented in appendix A.2, there are other types of layers and connections that can be used to construct deep convolutional networks. Typically, the purpose behind some of these components is to reduce the dimensions of intermediate layers, reshaping spatial dimensions, simulating fully connected layers and more. Following are some sections describing important components in the CNN framework used in this thesis.

Convolutions

The discrete 2D convolution operation, illustrated in figure 25, is defined by a convolution kernel k of size $k \times k$. Given an $N \times M$ input image (tensor) X, the convolution kernel is run along all the pixels in the image, multiplying the surrounding pixel values with the kernel and adding them. The resulting output

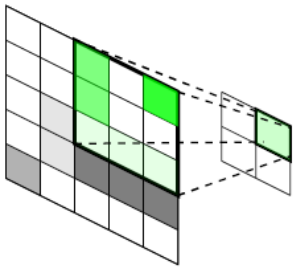


Figure 25.: Convolutional Operation Illustration of the convolution operation using a 3×3 kernel, where multiple input activations are associated to a single output activation. Courtesy [59].

is a $(N - k + 1) \times (M - k - 1)$ image Y, where the output at pixel location i, j is calculated

$$Y_{i,j} = \sum_{i'=1}^{k} \sum_{j'=1}^{k} k_{i',j'} X_{i-(k+1)/2+i',j-(k+1)/2+j'} \tag{18}$$

The operator is denoted using $*$ and thus

$$Y = k * X \tag{19}$$

The convolution operation between two layers in a CNN is defined by a kernel k_{ij} for each pair of feature maps in the two layers. Similar to the fully connected layer, the result from all convolutions related to feature map i in the later layer is summed and a bias b_i is distributed and added across the entire feature map. Let X_j , be a feature map in a layer with $N \geq j$ feature maps, then the feature map Y_i in the next layer is calculated.

$$Y_i = \sigma_i \left(\sum_{j=1}^{N} k_{ij} * X_j + b_i 1 \right) \tag{20}$$

where 1 is an all-ones matrix and σ_i is an element-wise activation function. The elements of all kernels k_{ij} and the biases b_i are learn-able parameters and are updated through the process of training the network.

As mentioned, the convolution operation has the effect of reducing the dimensions of the feature maps. However, there are some methods which can be used to avoid or modify this effect. A common strategy to preserve the spatial dimensions is to apply

zero-padding, i.e. covering the feature maps before the convolution operation with a border of $(k-1)/2$ zeros, which counters the spatial reduction exactly. On the other hand, to further reduce the spatial dimensions of intermediate layer activations, a stride can be associated with the convolution. The stride is applied by sliding the convolution kernel by s number of steps between each *multiply and sum* operation and the result is that the feature maps are reduced in size by a factor s. Most commonly, the convolutions are applied without stride, i.e. $s = 1$. An important concept introduced by applying a convolution layer to a CNN is the *receptive field*. The receptive field is a measure of how much information from the input image is available to the feature vectors of a specific layer in a CNN. If the first component of a CNN is a convolutional layer with kernel width k_1, the receptive field of the first layer is $k_1 \times k_1$ – i.e. each element in the second layer was provided information from the $k_1 \times k_1$ nearest pixels in the input image. Following this pattern, a series of n convolution layers with kernel sizes $k_1, \ldots, k_n$ results in a receptive field of

$$\left(1 + \sum_{i=1}^{n}(k_i - 1)\right) \times \left(1 + \sum_{i=1}^{n}(k_i - 1)\right) \tag{21}$$

Deconvolution

An operation analogous to the convolution operation, but reversed, referred to as deconvolution can be used for upsampling. It is achieved by convolution operation reversed, illustrated in figure 26. Hence, if upsampling by a factor f is desired, it can be formulated as a convolution with a fractional input stride of $\frac{1}{f}$. Such a layer can associate a single input activation to multiple output activations. This "deconvolution" layer has learn-able filter parameters that could correspond to bases for reconstructing shapes of an object [62]. Hence, an end-to-end learning mechanism can be constructed by repeated down-sampling and then upsampling to achieve dense predictions, and this technique has been successfully used for dense pixel-level predictions [38].

Pooling

Pooling layers are non-learnable layers used to reduce the spatial dimensions of the feature maps as they pass through the network. Similar to the convolution layer, they are associated with a kernel of size $k \times k$ and a stride s. There are two commonly used types of pooling layers; the max-pooling layer and the average-pooling layer. The max-pooling layer, illustrated in figure 27, performs a *max* operation with the elements of the feature map at each position of the kernel, thus discarding the

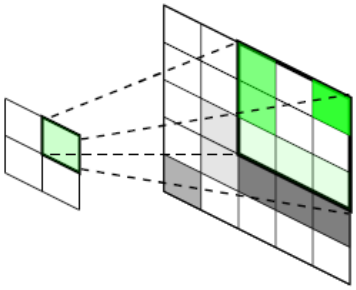


Figure 26.: Deconvolutional Operation Illustration of the deconvolution operation, where a single input activation is associated to multiple output activations. Courtesy [59].

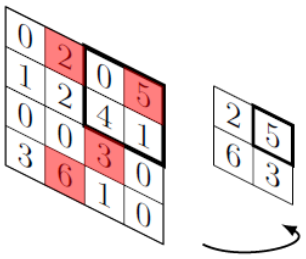


Figure 27.: Max-Pooling Operation Illustration of the max-pooling operation, where the maximum of activations in a window of input activations is recorded and the output activation is simply the maximum value recorded. Courtesy [59].

information of the non-max neurons.

The average-pooling layer performs an average at each position of the kernel, i.e. a normal convolution with the kernel values all set to $\frac{1}{k^2}$. Typically, the stride of the pooling layers is set to $s = k$, thus achieving a dimensional reduction of a factor s without using any zero-padding.

Apart from reducing the spatial dimensions of the feature maps, pooling layers also provide an efficient way of increasing the receptive field in a CNN. A pooling layer with stride s have the effect of increasing the receptive field of a factor s. This effect can also be achieved by including stride in a convolutional layer.

1×1 Convolutional Kernels

Technically, 1×1 convolutional kernels are no different from any $k \times k$ kernel in the way they are applied. However, there is a conceptual difference between them in that $k \times k$ convolutions are usually thought of as edge/feature detectors while 1×1 kernels can only combine activations of each feature vector. The interpretation of such an operation is that it is simulating the effect of a fully connected layer, applied to each feature vector [38]. Since the convolution operation is not dependent on the spatial size of its input, transforming fully connected layers to 1×1 convolutions is a useful way of generalizing the network for different input sizes.

Batch Normalization

Training deep neural networks can be quite tricky in practice due to the fact that the distribution of each intermediate layer's inputs changes during training. Such a change is caused by the changes in the parameters of the previous layers. This problem is referred to as internal covariate shift. When the input distribution changes, the activations tend to move into the saturated regimes, and this effect is amplified as the network depth increases, thus also causing the vanishing gradient problem. To tackle this, a normalization technique in [63] was introduced, where normalization of the inputs to the activation layers are done over the mini-batch.

The batch normalizing transform algorithm is explained in detail in [63]. Briefly, consider a mini-batch of n inputs in a mini-batch $B = x_1, x_2 \ldots, x_n$. Let the normalized values be $\hat{x_1}, \hat{x_2} \ldots, \hat{x_n}$ and the resulting linear transformation of the batch normalization can be represented by $y_1, y_2 \ldots, y_n$. Then, the batch normalizing transform given by

$$BN_{\gamma,\beta} : x_1, x_2 \ldots, x_n \rightarrow y_1, y_2 \ldots, y_n \tag{22}$$

where γ and β are learn-able parameters that correspond to the scaling and shifting in the transformation. Hence the mini-batch mean and variance which are given by

$$\mu_B \leftarrow \frac{1}{n} \sum_{i=1}^{n} x_i \tag{23}$$

$$\sigma_B^2 \leftarrow \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_B)^2 \tag{24}$$

are used to achieve the normalization. Thus, the normalization looks like

$$\hat{x_i} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{25}$$

and the resulting transformation can be given by

$$y_i \leftarrow \gamma \hat{x_i} + \beta \tag{26}$$

Dropouts

Dropout is a regularization technique for neural network models proposed by Srivastava et al. [64]. In a dropout layer, randomly selected neurons are ignored during training, meaning that their contribution to the activation is removed in the forward pass and any weight updates are not applied to these neuron during the backward pass. Dropouts are most common regularization technique used for training deep neural networks.

As an ANN learns, the weights of the neurons are tuned for specific features providing some specialization. Neighboring neurons rely on this specialization, which can result in a model that overfits to the training data. One can imagine that if neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network. The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.