Media Engineering and Technology Faculty
German University in Cairo

Scene Recognition From LiDAR Data

Bachelor Thesis

Author: Zeyad Shaban
Supervisors: Dr. Hisham Othman

Submission Date: 19 June, 2018

Media Engineering and Technology Faculty
German University in Cairo

Scene Recognition From LiDAR Data

Bachelor Thesis

Author: Zeyad Shaban
Supervisors: Dr. Hisham Othman

Submission Date: 19 June, 2018

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

---

Zeyad Shaban
19 June, 2018

Acknowledgments

First and foremost, I would like to thank each and everyone of my family for their endless support for me. I am sure I would not be where I am today if it were not for them. They provided me with everything I could ever dream of, the most important of which is love.

I sincerely thank my supervisor, Dr. Hisham Othman, for his guidance and support throughout my bachelor project. I am very grateful for the talk we had during every meeting, for he shared a lot of expertise and knowledge.

I would like to thank Dr. Mohamed Rehan, the CTO of AvidBeam, for his coordination and patience. He let me use AvidBeam's GPUs throughout the whole project. This was of great help during Milestone III.

I would also like to thank Eng. Kamal Hassan and Eng. Eslam Ahmed at AvidBeam for their great assitance. Eng. Eslam taught me a handful of Linux Terminal shortcuts that were of great convenience and also gave me a crash course about Neural Networks and their applications. Eng. Kamal scheduled slots for me to be able to use the machines and also helped me with some technical issues. Their help was of great importance for me.

I would also like to thank everyone at AvidBeam for their great hospitality and welcoming manner.

I also thank Eng. Mohamed El Demerdash at Valeo for the useful information he shared with me during Milestone I. He explained to me a lot about Image Classification and Object Detection.

Last but not least, I would like to thank my best and dearest friend, Mario Naguib, for being there every time I feel up or down. A friend's support is indeed priceless.

Abstract

LiDAR is one of the most exciting emerging technologies in the world of self-driving cars. Mature object detection in urban environments is one of the most complicated problems facing the future of autonomous driving. In this work, system models for extracting data and depth information from LiDAR sensor are investigated to reproduce and compare qualitative results. This was achieved by predicting over 1000 samples chosen from the famous KITTI dataset. Results are reproduced for car, pedestrian and cyclist detection on more than one benchmark. Also, visualizations for 3D bounding boxes surrounding object classes on images are demonstrated.

Contents

Chapter 1

Introduction

1.1 Motivation

Since the late 1960s, computer vision has always allowed everything from factories machinery and farm equipment to autonomous vehicles and drones to run more efficiently, intelligently and safely. With that said, computer visions importance has become even more obvious in a world deluged with digital images. Thanks to the advancements of digital cameras and camera phones, we have been consuming an astonishing amount of visual imagery that is far less useful and usable than it should be. The world wide web has been transformed to a huge pool of unstructured data. Luckily, visual recognition systems have been also seeing huge leaps over the last decade. This is due to the great improvements witnessed in computational power. Super computers can now be used in solving complex computational problems and performing research activities through computer modeling, simulation and analysis. However, visual recognition systems are still rarely employed in robotics applications.

With the increasing efforts of developing the computational performance of machines, a large array of possibilities and theories are taking more steps towards practicality. Visual recognition systems are no exception. Developing autonomous visual recognition systems that are able to assist humans in everyday tasks is one of the great challenges in modern computer science.

According to the human visual system, imagery is a descriptive language that engages some senses. The ultimate goal is trying to let the computer vision system mimic the human understanding of different scenes, patterns and objects. Scene recognition and classification is an active field of research and a lot of proposals are being introduced annually.

Today we are seeing these systems being embedded in a lot of applications. This includes autonomous cars, face detection systems, photo grouping applications, just to

name a few. LiDAR is one of the emerging nontraditional methods to extract data from images. It also has a wide range of applications in geography, geology, meteorology, airborne laser mapping, parks management, river surveying, transport planning, oil mining and solar energy. Our scope, however, is using LiDAR sensors in autonomous vehicles. It can provide accurate and reliable depth information about road features, lane markings, pedestrians, cyclists, stop signs and other obstacles. This information can be used to localize objects and characterize their shapes which helps maintain a wide-scale solution for self-driving cars.

1.2 Project Aim

The goal is trying to investigate the latest approaches that make use of LiDAR point cloud data to recognize and identify objects and also carry out an implementation for scene recognition from this data and provide thorough results.

1.3 Thesis Outline

This section discusses the structure of the thesis.

Chapter 2 is a literature review. It gives a quick introduction about Computer Vision, brief history of how object detection techniques developed over the years, the structure of LiDAR data and the nature of the used dataset. **Chapter 3** discusses the available traditional techniques used in scene recognition and extracting object information from an image. **Chapter 4** explains how a neural network is trained and tested to perform feature extraction and object detection. It also gives a brief explanation on CNNs and Faster R-CNNs. **Chapter 5** gives an introduction about scene recognition using LiDAR data and represents the workflow through out the project as well as the results. **Chapter 6** summarizes this work and suggests future implementations and enhancements.

Chapter 2

Background

2.1 Brief History of Object Detection

The story begins in 2001, the year an efficient algorithm for face detection was invented by two Computer Vision researchers from Cambridge, Paul Viola and Michael Jones [26]. Their demo that showed faces being detected in real time on a web camera was the most remarkable demonstration of computer vision and its potential at the time. Soon, it was implemented in OpenCV and the algorithm was both real time and robust.

In 2005, a group of french researchers created the Histograms of Oriented Gradients (HOG) [15]. HOG is a feature descriptor that counts occurrences of gradient orientation in localized portions of an image. This technique significantly outperformed existing algorithms in pedestrian detection.

Every few years, a new idea comes along that is so effective and powerful forcing researchers to embrace it. Deep Learning is that idea of this decade. Deep Learning algorithms had been around for a long time, but they became mainstream in computer vision with their success in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) of 2012 [20]. In that competition, an algorithm based on Deep Learning was introduced by three of University of Toronto graduates managed to achieve an astounding accuracy of 85%. This was 11% better than the algorithm that won the second place. In ILSVRC 2012, this was the only algorithm based on deep learning. In ILSVRC 2013, all winning algorithms were based on Deep Learning. In 2015 multiple Convolutional Neural Network (CNN) based algorithms surpassed the human recognition rate of 95%.

With such huge success in image recognition, Deep Learning became more popular. Techniques like Faster R-CNN produce surprising results over multiple object classes.

In the following two chapters, we will go through both traditional and modern object recognition and detection techniques.

2.2 Introduction to LiDAR

Light Detection and Ranging (LiDAR) is a remote sensing technology which is used to measure distances. One of LiDARs predecessors is radar, but unlike radar which sends out radio waves, it emits pulses of infrared light to detect objects.

To visualize how LiDAR works, one can imagine that a LiDAR sensor is actively sending light pulses to the ground from an airplane or a helicopter, when these pulses hit the ground and return to the sensor, the time taken for the pulses to travel is used to estimate distances and measurements. Similarly, these pulses can be used to measure distances between objects in multiple urban sceneries. LiDAR sensors are sampling tools. This means that they can send up to 160,000 pulses per second. They create a massive number of points. LiDAR wavelengths often vary to suit the target. It varies from 10 micrometers to approximately 250 nanometers. Suitable combinations of wavelengths can allow for remote mapping of atmospheric content. The best LiDAR sensors can see details of a few centimeters at distances of more than 100 meters.

LiDAR pulses usually can go through what normal light rays can penetrate. This means that a LiDAR unit scanning any environment can hit several objects from almost all angles. A significant amount of the LIDAR energy can penetrate a dense forest vegetation just like sunlight. Typically, a pulse can have more than one return. There can be several middle returns until the pulse finally hits its ground or wall hit and make its last return. If theres nothing in the way, a pulse will just hit the surface. This makes LiDAR data highly valuable for understanding different uniform and non uniform physical structures and geometries.

2.2.1 LiDAR Point Cloud Architecture

Point clouds are a collection of points that represent a 3D shape or feature. Each point has its own set of X, Y and Z coordinates and in some cases additional attributes. It can be thought of as a collection of multiple points, where each point cloud consists of a set of data points in 3D space. These data points cover surfaces of a sensed object. When many points are brought together they start to show some interesting qualities of the feature they represent. Data points in point clouds are always located on the external surfaces of visible objects because these are the spots where light rays from the scanner reflect from the objects. There can be up to millions of points created in one LiDAR point cloud. This helps map physical features with very high resolutions.

Point clouds are most often created by methods used in photogrammetry or remote sensing. Photogrammetry uses photographs to survey and measure the dimensions of objects as well as the areas between them. A combination of photographs taken at many angles can be used to create point clouds. Remote sensing is a way of collecting data of the Earth by use of satellites or aircrafts. On these aerial vehicles, LiDAR sensors can be mounted to collect information about the shape of the Earth and its features.

2.2.2 LiDAR in Autonomous Vehicles

The LiDAR sensor can provide the necessary data for a vehicle software to determine where potential obstacles exist in the environment and where the vehicle is in relation to those obstacles.It uses light waves to measure distances to objects, rendering much more precise distance data.

In transportation systems, to ensure vehicle and passenger safety and to develop electronic systems that deliver driver assistance, understanding the vehicle and its surrounding environment is essential. Lots of electronic systems which add to the driver assistance usually depend on the detection of a vehicle's environment to act autonomously or semi-autonomously. LiDAR mapping and estimation achieves this.

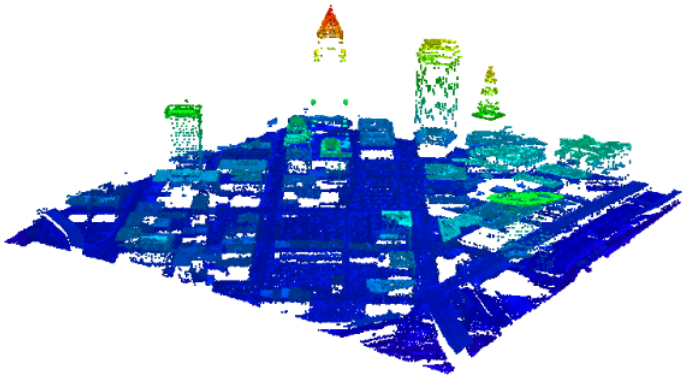


Figure 2.1: Urban scene as seen through LiDAR sensor

A major advantage of using LiDAR is the spatial structures obtained. This data can be fused with other sensors, such as radar, to get a better picture of the vehicle environment in terms of static and dynamic properties of the objects present in the environment. The fusion of LiDAR measurement with different sensors makes the system robust in real-time applications. Moreover, LiDAR sensors have relatively smaller wavelengths which yields much better accuracies. This is extremely useful for autonomous vehicles and similar systems. More about LiDAR is discussed in section 5.1.

2.3 KITTI Dataset

The KITTI Vision Benchmark Suite [17] presents a novel dataset captured from a station vehicle for use in mobile robotics and autonomous driving research [16]. In total, they recorded 6 hours of traffic scenarios at 10-100 Hz using a variety of sensor modalities. The dataset is captured by driving around in a mid-sized city with up to 15 cars and 50 pedestrians visible per image. The scenarios are diverse, capturing real-world traffic situations and range from freeways over rural areas to inner city scenes with many static

and dynamic objects. Their data is calibrated, synchronized and timestamped. They also provide raw image sequences as well as object labels in the form of 3D tracklets. They provide several online benchmarks among which is the 3D object detection benchmark which is our scope of interest.

The 3D Object Tracking Benchmark focuses on computer vision algorithms for object detection and 3D orientation estimation. The dataset associated with this benchmark provides accurate 3D bounding boxes for object classes such as cars, vans, trucks, pedestrians, cyclists and trams. This results in classifying the image into accurate 3D poses, which can be used to evaluate the performance of algorithms for 3D orientation estimation and 3D tracking.

For collecting data used to evaluate the benchmark, the following equipment was used: two color video cameras, two grayscale video cameras, a Velodyne LiDAR sensor, a GPS/IMU localization unit and a computer running a real-time database.

The setup was done over two units. Each unit contains one of the two types of video cameras mentioned above. One unit was fixed on the right side (on the top of the car) and the other was fixed on the left side. The camera setup is chosen such that to obtain a baseline of roughly 54 cm between the same type of cameras and that the distance between color and grayscale cameras is minimized (6 cm). This is a good setup since color images are very useful for tasks such as segmentation and object detection, but provide lower contrast and sensitivity compared to their grayscale counterparts, which is of high importance in stereo matching and optical flow estimation. The position information from the GPS/IMU system is used to compensate egomotion in the 3D laser measurements. All the equipment is manually calibrated.

To generate a 3D object ground-truth, a set of annotators were hired. They were asked to assign tracklets in the form of 3D bounding boxes to objects such as cars, vans, trucks, trams, pedestrians and cyclists. A special purpose labeling tool was created to display 3D laser points as well as the camera images to increase the quality of the annotations. Then the annotators were asked to additionally mark each bounding box as either visible, semi-occluded, fully occluded or truncated. Overall, three terabytes of data were collected. But only a small subset of this data is used to evaluate benchmarks. This is the dataset that we will be using.

Chapter 3

Classic 2D Object Detection

3.1 Image Classification

An image recognition algorithm (image classifier) takes an image or a patch of an image as input and outputs what the image contains. The output is a class label defining the main component in the image.

In figure 3.1, an image classifier takes a single image and assigns probabilities to four labels. From the computer perspective, an image is represented as one large 3-dimensional array of numbers. In this example, the cat image is 248 pixels wide, 400 pixels tall, and has three color channels: Red, green and blue (RGB). Therefore, the image consists of 248 x 400 x 3 numbers, or a total of 297,600 numbers. Each number is an integer that ranges from 0 (black) to 255 (white). The task is to turn these numbers into a single label, such as cat. In other words, we need to predict a label for the given image. That is the label whose class achieves the highest score among all other classes.

3.1.1 Anatomy of Image Classifier

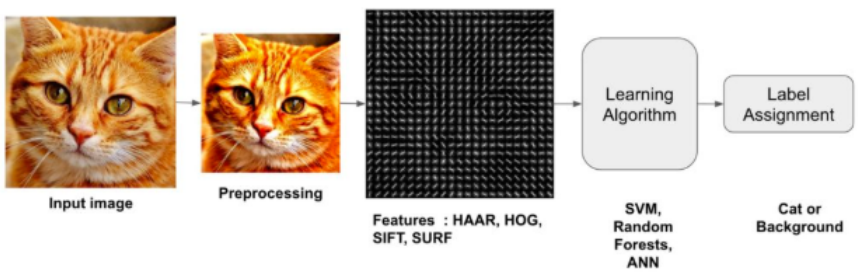


Figure 3.1: Image Classification Pipeline

The diagram in figure 3.1 illustrates the procedures involved in a traditional image classifier. Many traditional Computer Vision image classification algorithms follow this pipeline, while Deep Learning based algorithms bypass the feature extraction step completely. We will go through the stages of the pipeline in this chapter. However, we will discuss how Deep Learning algorithms work in the following chapter.

First of all, often an input image is pre-processed to normalize contrast and brightness effects. A very common pre-processing step is to subtract the mean of image intensities and divide by the standard deviation. In other cases, when dealing with color images, a color space transformation may help get better results. There are no prescribed steps for pre-processing. Better results are always reasonably achieved by trial and error. Also, as part of pre-processing, the input image or the input patch is cropped and resized to a fixed size. This is essential because the next step, feature extraction, is performed on a fixed-size image.

Input images have too much extra information that is not necessary for classification. Therefore, the first step in image classification is to simplify the image by extracting the important information contained in the image and leaving out the rest. For example, if we want to find shirt buttons in images, we will notice a significant variation in RGB pixel values. However, by running an edge detector on the image we can simplify it. We can easily recognize the circular shape of the buttons in the image and so we can conclude that edge detection retains the essential information while throwing away non-essential information. This step is called feature extraction. In traditional computer vision approaches, designing these features are crucial to the performance of the algorithm. However, we can do much better than simple edge detection and find features that are much more reliable. In our example of shirt buttons, a good feature extractor will not only capture the circular shape of the buttons but also information about how buttons are different from other circular objects like car tires.

In the previous two steps, the image is converted to a feature vector. However, before a classification algorithm can be used, we need to train it by showing thousands of examples of foregrounds and backgrounds. Different learning algorithms learn differently, but the general principle is that learning algorithms treat feature vectors as points in higher dimensional space, and try to find surfaces that partition the higher dimensional space in such a way that all examples belonging to the same class are on one side of the surface. This way, the classification algorithm takes the feature vector as input and outputs a class label.

There can exist, however, several challenges while achieving the task of recognizing objects that are worth considering. Some of these challenges are occlusion, illumination conditions, background clutter, intra-class variation, object deformation, viewpoint variation, etc. A good image classifier must be invariant to the cross product of all these variations.

3.1.2 Object Localization

In general, if we want to classify an image into a certain category, we should use image classification. On the other hand, if we aim to identify the location of objects in the image, or count the number of instances of an object, we should use object detection. There is however some overlap between these two scenarios. If we want to classify an image into a certain category, it could happen that the object or the characteristics that are required to perform categorization are too small with respect to the full image. In that case, we would achieve better performance with object detection instead of image classification even if we are not interested in the exact location or count of the object.

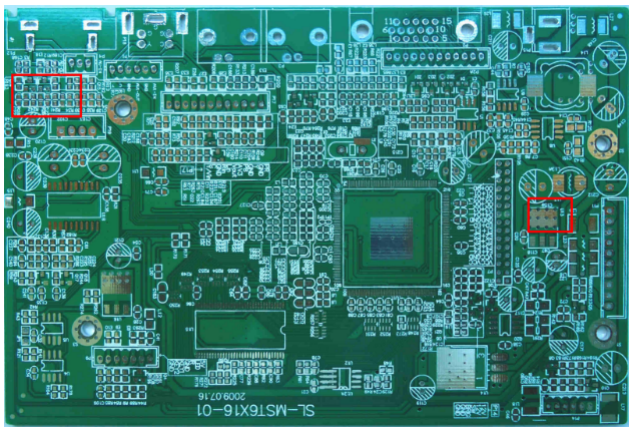


Figure 3.2: Object localization on an electric circuit

For example, suppose we need to check circuit boards and classifying them as either defect or correct. While it is essentially a classification problem, the defects might be too small to be noticeable with a classification model as in figure 3.2.

With an image classification model, we generate image features of the full image. These features are aggregates of the image. With object detection, we perform this on a finer regional level of the image. Constructing an object detector will cost more time, yet it will result in a better model and has a bigger chance to achieve better results.

3.2 Introduction to Object Detection

Object Detection is a basic visual perception task and one of the key areas of applications of Computer Vision. It essentially deals with finding and locating specific objects within an image. With the recent advancements in computational power, object detection applications are easier to develop than ever before. In addition, with current approaches leveraging full end-to-end pipelines and massive datasets, performance has also improved significantly, enabling real-time use cases.

An object recognition algorithm identifies which objects are present in an image. It takes the entire image as an input and outputs class labels and class probabilities of objects present in that image. For example, a class label could be a cat and the associated class probability could be 97%. On the other hand, an object detection algorithm not only tells you which objects are present in the image, it also outputs bounding boxes to indicate the location of the objects inside the image.

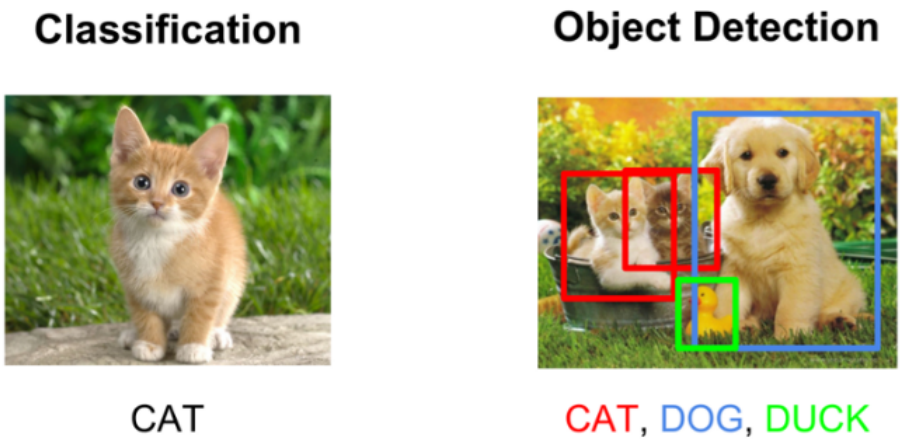


Figure 3.3: The difference between classification and detection

Thus, at the heart of every object detection algorithm is an object recognition algorithm. Suppose there is a trained object recognition model which identifies dogs in image patches. This model will tell whether an image contains a cat or not. It does not tell where the object is located. To localize the object, sub-regions (patches) of the image have to be selected and then applying the object recognition algorithm to these image patches. The location of the objects is given by the location of the image patches where the class probability returned by the object recognition algorithm is high or above a certain threshold.

There are many ways to generate small sub-regions. The most important two of which are the Sliding Window algorithm and the Region Proposal algorithm.

3.2.1 Sliding Window Algorithm

In the sliding window approach, we slide a box or window over an image to select a patch and classify each image patch covered by the window using the object recognition model. It is an exhaustive search for objects over the entire image. Not only do we need to search

all possible locations in the image, we have to search at different scales. This results into classifying tens of thousands of image patches. Moreover, the Sliding Window approach is good for fixed aspect ratio objects such as faces or pedestrians. The sliding window approach is computationally very expensive when we search for multiple aspect ratios.

3.2.2 Region Proposal Algorithms

The problems with the Sliding Window approach can be solved using the region proposal algorithm. This method takes an image as an input and outputs bounding boxes corresponding to all patches in the image that are most likely to be objects. These region proposals can be noisy, overlapping and may not contain the object perfectly, but among these region proposals, there will be a proposal which will be very close to the actual object in the image. These proposals can then be classified using the object recognition (classification) model. The region proposals with high probability scores are locations of objects of interest.

Region proposal algorithm identifies prospective objects in images using segmentation. In segmentation, adjacent regions which are similar to each other are grouped together. Similarities between regions are based on some criteria such as color, texture, shape, size, etc. Unlike the Sliding Window approach, the Region Proposal algorithm works on grouping pixels into a smaller number of segments. So the final number of proposals generated are many times less than that of the Sliding Window approach. This reduces the number of image patches that have to be classified and makes region proposals applicable to different scales and aspect ratios.

Generally, Region Proposal algorithm provides a very high recall. This is because the regions that contain the objects of interest have a very high chance to be among the list of region proposals offered by the algorithm. Although the list may end up containing a lot of regions that do not contain any object, but it is acceptable for the region proposal algorithm to produce a lot of false positives so long as it catches all the true ones. Most of these false positives will be rejected by applying the object recognition algorithm. When there are more false positives, the time it takes to do the detection goes up and the accuracy is affected slightly. But having a high recall is still good because missing regions containing actual objects severely impacts the detection rate.

Selective Search is one of the most popular Region Proposal algorithms used in object detection. It is designed to be fast with a very high recall. It is based on computing hierarchical grouping of similar regions based on color, texture, size and shape compatibility.

Selective Search starts by over-segmenting the image based on the intensity of the pixels using the graph-based segmentation method on [3]. However, these segmented regions can not be used as region proposals. That is because most objects contain more than one segmented reason and also region proposals for occluded objects can not be generated using this method. Segmentation is not perfect, but the goal is to just predict many

region proposals such that some of them should have very high overlap with actual objects.

Selective Search takes these segmented parts and then adds all bounding boxes corresponding to those parts to the list of region proposals and group adjacent segments together based on the similarities (color, texture, size and shape) between them. It does this multiple times. At each iteration, larger segments are formed and added to the list of region proposals. Hence, multiple region proposals from smaller segments to larger and more complex segments are created. This process is called **hierarchical segmentation**. Figure 3.4 [25] shows the initial, middle and last step of the hierarchical segmentation process. Selective Search typically gives thousands of region proposals.

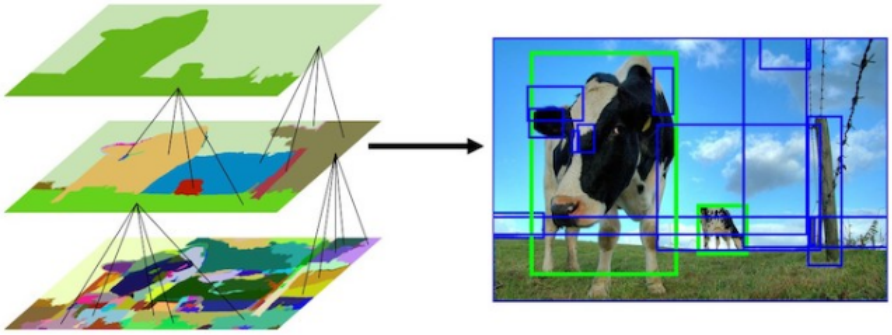


Figure 3.4: Hierarchical Segmentation

Chapter 4

2D Object Detection with Deep Learning

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images, cluster them by similarity, and perform object recognition within multiple scenes. They can identify faces, individuals, street signs, tumors and many other aspects of visual imagery. The efficiency of CNNs in image recognition is powering major advances in Computer Vision. That is one of the main reasons why Deep Learning became more popular recently.

2012 was the first year that neural networks became of great significance as Alex Krizhevsky, from University of Toronto, used them to win the ILSVRC. This resulted in dropping the classification error record from 26% to 15%, an astounding improvement at the time. Alex and his two colleagues published a paper explaining how CNNs work [20]. Ever since then, a lot of technology companies have been using Deep Learning at the core of their services.

In this chapter, we will discuss regular neural networks, convolutional neural networks and the role of Deep Learning in object recognition and detection.

4.1 Traditional Neural Networks

The simplest definition of a neural network is that it is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs [14].

Generally, neural networks receive an input, and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons within a single layer function completely independently and do not share any connections. The last fully-connected layer is called the output layer and in classification settings, it represents the

class scores.

A neural network can be perceived as a black box. For example, it can have one input and three outputs. The input is an image of any size. The three outputs are numbers between 0 and 1. The outputs are labeled Cat, Dog, and Other. The three numbers always add up to 1.

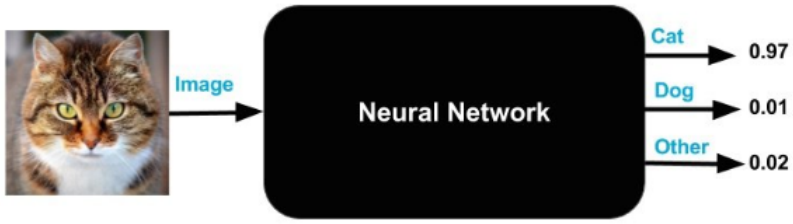


Figure 4.1: Neural network as a black box

A perfect neural network would output (1, 0, 0) for a cat, (0, 1, 0) for a dog and (0, 0, 1) for anything that is not a cat or a dog. In reality, though, even a well trained neural network will not give such accurate results. The outputs can be interpreted as probabilities. This specific output means that the black box thinks there is a 97% chance that the input image is an image of a cat and a small chance that it is either a dog or something else.

For an image of size 256*256*3, the input is actually 196,608 numbers.

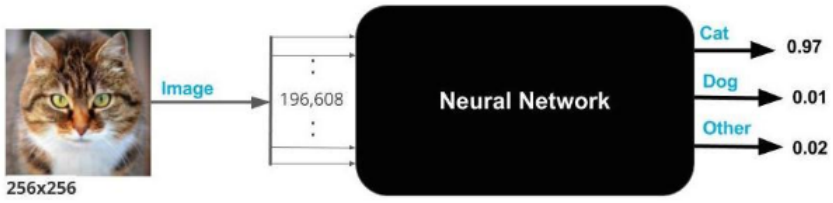


Figure 4.2: Input image as a 2D matrix

4.1.1 Training Neural Networks

The black box often has weights that can be used to tune it. When the weights are in the right position, the neural network can give the right output more often for different

inputs. This black box is useless if the right weight settings are not applied. Thus, the neural network must be trained. Training the neural network simply means finding the right weight settings.

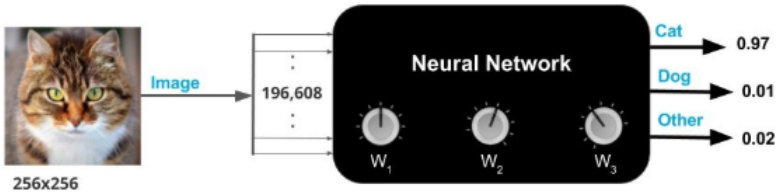


Figure 4.3: Network Weights

Training a neural network is very similar to training a little child. When we show a child a ball, tell him that it is a ball and repeat that many times with different kinds of balls, the child figures out that it is the shape of the ball that makes it a ball and not the color, texture or size.

To train a neural network, we show it several thousand examples of the classes we want it to learn about. This kind of training is called Supervised Learning because we are providing the neural network an image of a class and explicitly telling it that it is an image from that class.

To train a neural network, two main things are needed. First, the **labeled training data**, which is thousands of images of each class with their corresponding bounding box coordinates (for example, the bottom left and top right coordinates) and labels. Second, the **cost function**. This is because we need to know if the current setting is better than the previous setting or not. A cost function sums up the errors made by the neural network over all images in the training set. For example, a common cost function is called sum of squared errors (SSE). If the expected output for an image is a cat , or (1, 0, 0), and the neural network outputs (0.37, 0.5, 0.13), the squared error made by the neural network on this particular image is: $(1 - 0.37)^2 + (0 - 0.5)^2 + (0 - 0.13)^2 = 0.6638$. The total cost over all images is simply the sum of squared errors of all images. The goal of training is to find the right weight settings that will minimize the cost function.

As shown in figure 4.4, if our cost function is shaped like a bowl, we could find the slope of the cost function and move a step closer to the optimum weight setting. This procedure is called Gradient Descent because we are moving down (descending) the curve is based on the slope (gradient). When we reach the bottom of the bowl, the gradient or slope goes close to zero and that completes our training.

Gradient Descent works similarly when there are multiple weights. For example, when

there are two weights, the cost function is a bowl in 3D. If we place a ball on any part of this bowl, it will roll down to the bottom following the path of the maximum downward slope. This is exactly how gradient descent works. Also, if we let the ball roll down at full velocity, it will overshoot the bottom and it will take much more time to settle down at the bottom compared to a ball that is rolled down slowly in a more controlled manner. Similarly, while training a neural network, we use a parameter called the learning rate to control convergence of cost to its minimum.

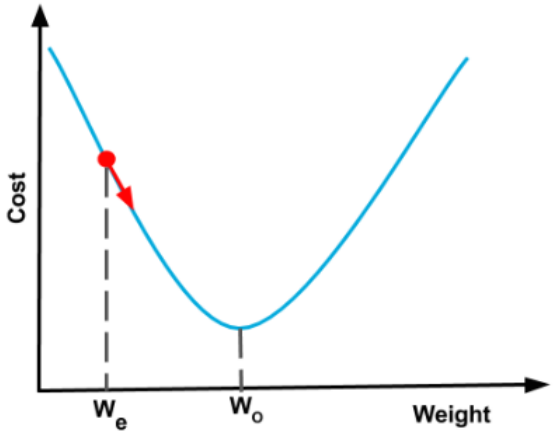


Figure 4.4: Gradient Descent

When we have millions of weights, the shape of the cost function is a bowl in this higher dimensional space. Even though such a bowl is impossible to visualize, the concept of slope and Gradient Descent works just as well. Therefore, Gradient Descent allows us to converge to a solution, thus making the problem controllable.

Furthermore, since the cost function depends on the difference between true output and the current output for all images in the training set, every image in the training set contributes to the final gradient calculation based on how badly the neural network performs on those images.

However, regular neural networks do not scale well to full images. For example, an image of size 256x256x3 would lead to neurons that have 196,608 weights. Moreover, we would want to have several neurons. So this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional neural networks, however, take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular neural network, the layers of a CNN have neurons arranged in 3 dimen-

sions, which are width, height and depth. For example, for an image of size 32x32x3, the final output layer would be of size 1x1x10, because the size of the full image will be reduced into a single vector of class scores, arranged along the depth dimension. Unlike regular neural networks, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner.

4.2 Convolutional Neural Networks

Convolutional neural networks do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by an experiment by Hubel and Wiesel, two Canadian neurophysiologists, in 1962. They showed that some individual neuronal cells in the brain responded only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons together were able to produce visual perception [13]. This idea of specialized components inside of a system having specific tasks, such as looking for specific characteristics, is one that machines can use as well, and is the basis for CNNs.

Last chapter, we discussed how image classification generally works and how the computer is able to look for low-level features such as edges and curves. Here, we will discuss how CNNs manage to detect features and recognize objects based on the same principle, but by building up to more abstract concepts through a series of middle layers. A more detailed overview of how a CNN works would be that it takes an image, passes it through a series of convolutional, nonlinear, and fully connected layers, and produces an output. We will discuss some of these layers in the following subsections.

4.2.1 Convolutional Layer

The first layer in a CNN is always a convolutional layer. The best way to explain a convolutional layer is to imagine a flashlight that is shining over a portion of an image. Let us suppose that we have an image of size 32 x 32 x 3 and that this flashlight covers a 5 x 5 area. This flashlight is called the filter, or the kernel and the region that it is shining over is called the receptive field. The depth of this filter has to be the same as the depth of the input, so the dimensions of this filter is 5 x 5 x 3. It starts at the top left 5 x 5 pixels of the image, and it slides across all areas of the input image. The number of steps that the filter slides to go to the next receptive field is called the stride. For a stride of 1, the filter slides by moving 1 pixel to the right. As it is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image. These multiplications are all summed up. So now we have a dot product for every 5 x 5 window in the original image. Every unique location on the input volume corresponds to a number. After sliding the filter over all the locations, we are left

with a 28 x 28 x 1 array of dot products, which is called an activation map, or feature map.

The reason we get a 28 x 28 array is that there are 784 different locations that a 5 x 5 filter can fit on a 32 x 32 input image. These 784 numbers are mapped to a 28 x 28 array. The width of the activation map is equal to the total number of steps the filter takes to slide across one row of the underlying image. The size of the matrices that convolutional networks process and produce at each layer is directly proportional to how computationally expensive the networks are and how much time they take to train. A larger stride means less time and computational power but also a smaller activation map.

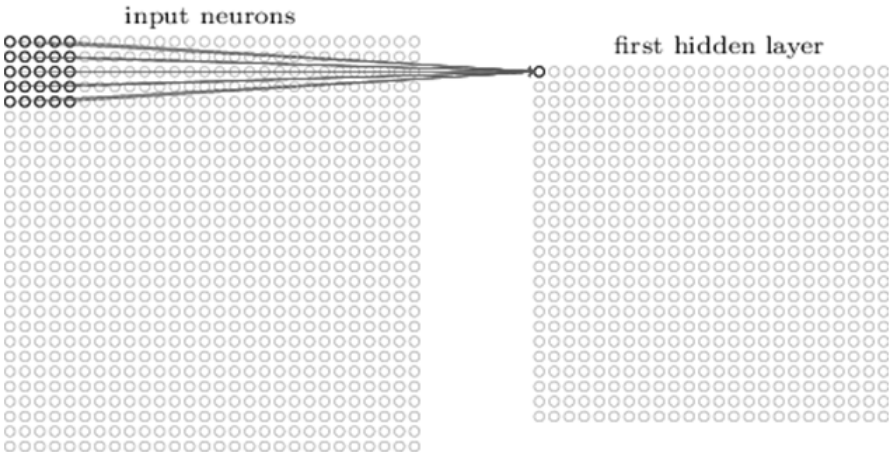


Figure 4.5: Constructing Activation Maps

If we had used two 5 x 5 x 3 filters instead of one, then our output volume would have been 28 x 28 x 2, which are two activation maps on top of one another. Mathematically, this is how a convolutional layer works.

Each of these filters can be thought of as a **feature identifier**. Generally, features in images represent straight edges, simple colors, curves, etc. These are some of the simplest characteristics that all images have in common with each other. Suppose our first filter is a curve detector and of size 7 x 7 x 3. As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve.

Furthermore, suppose that we will pass our curve detector over the image in figure 4.6. As long as the receptive field represents a curve, the corresponding value in the feature map will have to be a large number. As the filter moves along the image, convolving a receptive field that does not represent a curve results in a much lower value. This is because there was not anything in this portion of the image that responded to the curve detector filter.

At the end, and after the filter had passed over all possible receptive fields in the image, the resulted activation map will show the areas which most likely contain curves in the

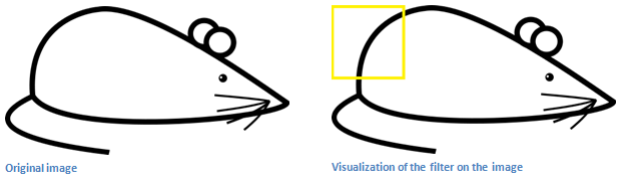


Figure 4.6: Input Image

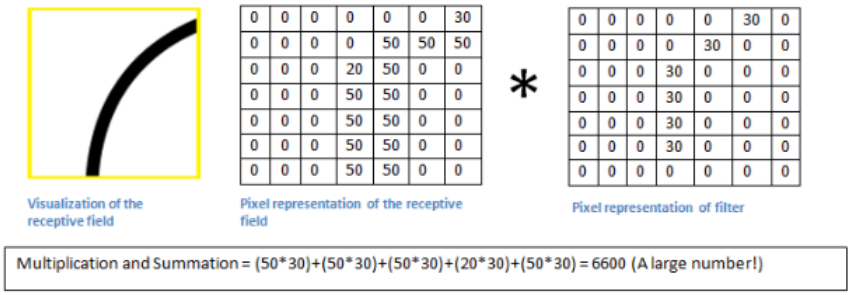


Figure 4.7: The receptive field contains a curve

image. In this example, the top left value of our 26 x 26 x 1 activation map will be 6600. This high value means that it is likely that there is a curve in the input volume that caused the filter to activate. Similarly, the top right value in our activation map will be 0 because there was not anything in the input volume that caused the filter to activate. We can have other filters for lines, curves to the left or even straight edges. The more the filters, the greater the depth of the activation map, and the more information we have about the input volume.

4.2.2 Middle Layers

After passing through the first convolution layer, the output is an activation map that represents the availability of low level features. Going through another convolutional layer, the output of the last layer becomes the input of this layer. When we apply a set of new filters in the second layer on top of our old activation map, the output will be another activation map that represents higher level features. Types of these features can be semi-circles, squares, or any shape that is a result of a combination of several low level features such as straight lines or simple curves.

As we go through the network, and pass through more convolutional layers, we get activation maps that represent more complex features and the patterns processed by the convolutional network becomes more abstract. Also, the filters begin to get more responsive to larger pixel spaces which results in them being able to convolve with larger receptive fields and consider information from larger areas of the original input volume.

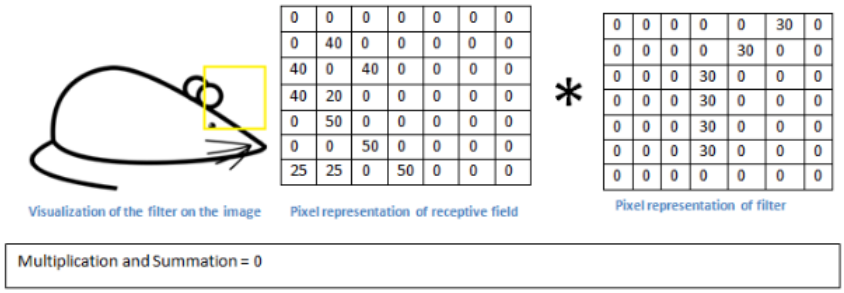


Figure 4.8: The receptive field does not contain a curve

In a traditional CNN architecture, there are other layers between the convolutional layers. They provide nonlinearities and preservation of dimension that can help improve the robustness of the network and control overfitting.

4.2.3 Downsampling Layer

One of the main problems with images is that they are high-dimensional, which means they cost a lot of time and computing power to process. Convolutional networks are designed to reduce the dimensionality of images in a variety of ways. Increasing filter stride is one way to reduce dimensionality. Another way is through downsampling.

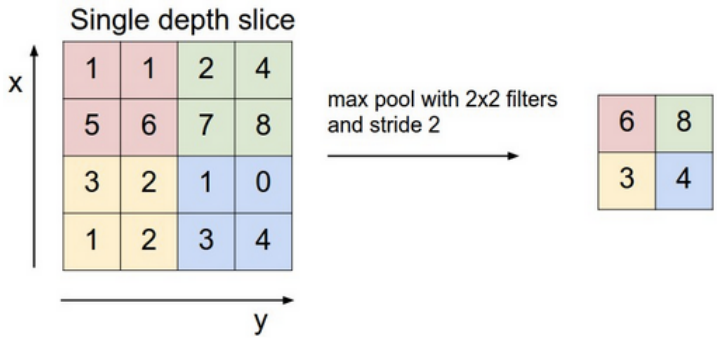


Figure 4.9: Maxpooling (Downsampling)

The sampling layer showed in figure 4.9 [?] has three names: max pooling, downsampling and subsampling. Generally, the activation maps of convolutional layers are passed through this downsampling layer one patch at a time. In this case, max pooling simply chooses the largest value from every patch of the image, places it in a new matrix next to max values from all other patches, and discards the rest of the information contained in the activation maps. This way, only the locations on the image that showed the strongest correlation to each feature are preserved, and those maximum values combine to form a lower-dimensional space. This, however, reduces the amount of storage and processing required.

4.2.4 Fully Connected Layer

After detecting all of those high level features, a fully connected layer is the layer at the end of the network that takes an input volume, that is the output of the preceding layer, and outputs an N dimensional vector, where N is the number of classes that the image classifier can choose from. Each value in the vector represents the probability of a certain class.

The way this fully connected layer works is that it uses the output of the preceding layer to determine which features most correlate to a particular class. For example, if the classifier is predicting a dog, there must be high values in the activation maps that represent high level features like paws, tails, four legs, etc. Similarly, if the classifier is predicting a bird, there must be high values in the activation maps that represent high level features like wings, beaks, etc.

Full CNN Example

The image in figure 4.10 shows an example of a typical convolutional network.

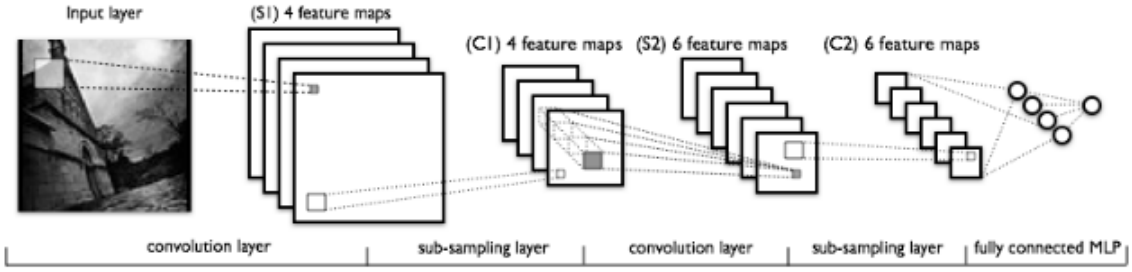


Figure 4.10: Overview of a typical CNN

We will go through the layers of the network from left to right:

- First: an actual input image. The light rectangle is the filter that slides across it.
- Second: a convolutional layer that produces one or more activation maps, one for each filter, stacked on top of one another.
- Third: a downsampling layer that produces the condensed activation maps.
- Fourth: a new convolutional layer and a new set of activation maps created by passing filters over the downsampled stack of activation maps.
- Fifth: a second downsampling layer, which condenses the second set of activation maps.
- Finally: a fully connected layer that classifies the output with one label per node.

4.3 Faster R-CNN

Faster R-CNN was developed by researchers at Microsoft [24]. It is based on R-CNN [19]. Instead of using default bounding boxes, Faster R-CNN, at its core, has two networks: a Region Proposal Network (RPN) for generating a fixed set of region proposals and a network to detect objects. The RPN uses the convolutional features from the the image classification network, enabling nearly cost-free region proposals. The RPN is implemented as a fully convolutional network that predicts object bounds and object class scores at each position. The main difference with Fast R-CNN [18] is that the later uses Selective Search (discussed in section 3.2.2) to generate region proposals.

The RPN network works with sliding windows across the feature maps. At each sliding-window location or anchor, a set of proposals are computed with various scales and aspect ratios. The outcome of the RPN is adjusted bounding boxes. Another interpretation of the RPN is that it guides the network attention to the interesting regions that are most likely containing objects.

After using the RPN, proposed regions with different sizes are created. Different sized regions means different sized CNN feature maps. It is not always easy to make an efficient structure work on feature maps with different sizes. Thus, the Region of Interest Pooling step can further simplify the problem by reducing the feature maps to make them exactly the same size. In this step, ROI Pooling splits the input feature map into a fixed number of equal regions, and then apply Max Pooling on each of those regions. Therefore, the output of ROI Pooling is independent of the size of the input [9].

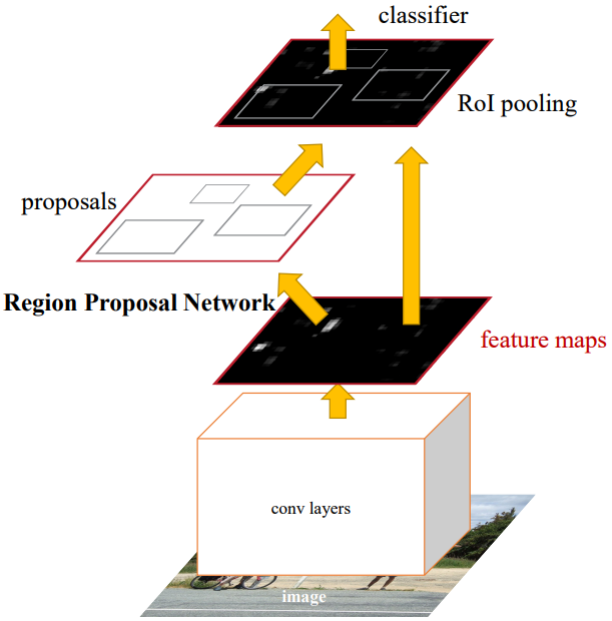


Figure 4.11: Overview of Faster R-CNN

4.4 Evaluation Metrics

4.4.1 Average Precision

The most common evaluation metric that is used in object recognition tasks is Average Precision (AP). It is a number from 0 to 100 that represents the ratio of true object detections to the total number of objects that the classifier predicted. The higher the value, the better the performance.

Each bounding box typically has a score associated with it that represents the probability of the box containing an object. Based on the predictions, a precision-recall curve is computed for each class by varying the score threshold that determines what is counted as a true positive detection for this class. The Average Precision is the area under this curve. First the AP is computed for each class, and then averaged over all classes.

4.4.2 Intersection over Union

Intersection over Union (IoU) is is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. It represents a value between 0 and 1 that corresponds to the overlapping area between the predicted bounding box and the ground-truth bounding box. The higher the IoU, the better the predicted location of the box for a given object. All bounding boxes with an IoU greater than a set threshold are considered true positives.

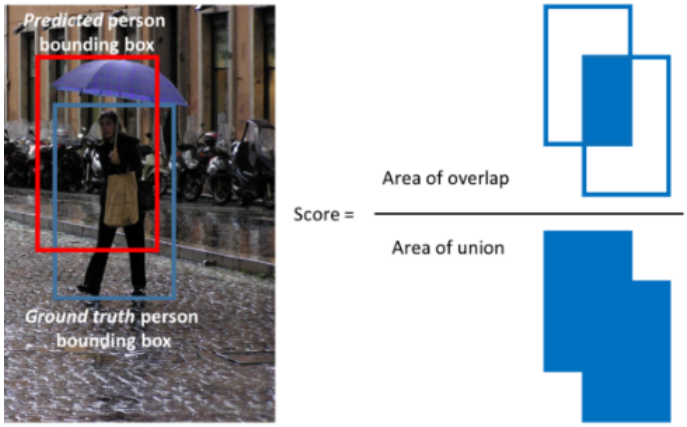


Figure 4.12: Intersection over Union

Chapter 5

Object Detection from LiDAR Data

This chapter discusses the workflow of the project. The project is divided into three main milestones. The first milestone was concerned with the research phase: identifying the problem and studying state-of-the-art related work. The second one was concerned with preparing data as well as everything about the setup needed in order to reproduce results. The last milestone dealt with achieving results on extracting information from LiDAR sensor data and dumping visualizations.

5.1 Milestone I: Research

The first milestone was concerned with research. Researching was done simultaneously on multiple fields in order to have a better understanding of the problem and take positive steps towards its solution. First of all, a detailed research was done on exploring the state-of-the-art approaches on KITTI 3D Object Detection and Bird's-Eye View Object Detection Benchmark Leaderboards [6, 7]. Second was researching the traditional and modern machine learning algorithms and Deep Learning techniques. Third was researching and understanding the structure and format of the KITTI raw dataset [16].

The first milestone required studying some sections in **CS231n** and **CS229** courses offered online by Stanford University [1, 2]. This helped grasp decent knowledge of best practices used in designing, implementing and evaluating object detection models based on machine learning.

At the end of this milestone, two state-of-the-art techniques were chosen to implement and compare results: VoxelNet [27] and Frustum PointNets [22]. Further details about the two approaches are explained in the following subsections.

5.1.1 Introduction to Object Detection with LiDAR

LiDAR works by spinning continuously in 360 degrees, firing high-frequency beams of concentrated light, and then measuring how long it takes for these beams to return to

the sensor. The results are then compiled into a point cloud, which can be defined as a set of data points in space. This happens millions of times per minute, which means millions of data points and calculations are taking place at an incredible speed. The result is a constantly-updated map of the world around the sensor. This map is so detailed that it can be used to simultaneously track objects, which makes it extremely useful for safely guiding autonomous vehicles on the road. By using a LiDAR to map and navigate an environment, an autonomous car knows ahead of time the boundaries of surrounding objects. This advanced predictability allows for better adjustment and more safety. That is because a self-driving car needs to safely make decisions every second while constantly re-adjusting for new data coming in from the environment.

The remarkable thing about LiDAR is that it functions almost instantaneously for both stationary and moving objects. In other words, it detects moving pedestrians, cyclists and other vehicles in traffic. In this work, we will demonstrate how detection is actively performed on static as well as moving objects.

Compared to most of radars, LiDAR is far more target selective and more precise. Furthermore, LiDAR significantly reduces blind spots and wide-angle hazards because of its 360 degrees view and extended sensor range. Also, LiDAR technology is effective in low-visibility situations such as night time, because it emits its own source of light.On top of this, LiDAR provides computer-friendly data in the form of exact measurements.

However, LiDAR is considered to be an immature emerging technology, which means that there is still a lot that needs to be figured out to make LiDAR a viable wide-scale product for the autonomous vehicle industry.

5.1.2 Frustum PointNets

Instead of solely relying on 3D proposals, Frustum PointNets leverages both mature 2D object detectors and advanced 3D deep learning for object localization. This method is more 3D-centric as it lifts depth maps to 3D point clouds and process them using 3D tools.

The best thing about Frustum PointNets is that it does not convert point cloud data to volumetric grids, but operates on raw point clouds by learning directly from them. That is because this transformation may obscure the 3D patterns of the original data.

Frustum PointNets reduce the search space by making use of precomputed 2D object detections. It extracts 3D bounding frustums of objects by extruding 2D bounding boxes from these detections as illustrated in figure 5.1. Then, within this new 3D search space, it consecutively perform 3D object instance segmentation and amodal 3D bounding box regression using PointNet [23]. The segmentation network predicts the 3D mask of the object of interest and the regression network estimates the amodal 3D bounding box covering the entire object even if part of it is occluded or truncated.

Frustum Proposals

Frustum proposals are generated based on 2D bounding boxes, which are lifted to a frustum that defines a 3D search space for the object. All points within the frustum are then collected to form a frustum point cloud. Frustums, however, may orient towards many different directions, which can result in large variation in the placement of point clouds. Therefore, the frustums are normalized by rotating them towards a center view such that the center axis of the frustum is orthogonal to the image plane. 2D object detections were performed according to [21].

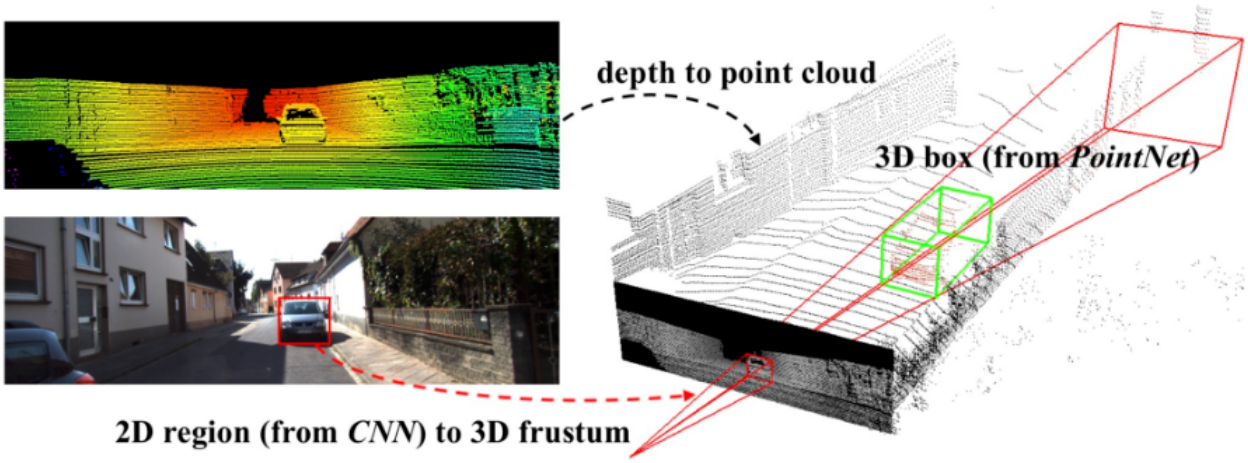


Figure 5.1: How frustums are generated

3D Instance Segmentation

Given the 3D frustums, instance segmentation is performed on 3D point clouds using [23]. The network takes a point cloud in frustum and predicts a probability score for each point that indicates how likely the point belongs to the object of interest. Each frustum contains exactly one object of interest. After 3D instance segmentation, points that are found to belong to the object of interest are extracted.

Amodal 3D Box Estimation

Given the segmented object points, this module estimates the objects amodal oriented 3D bounding box by using [23] and a processing transformer network. The output of this step is not an object class score, but parameters for a 3D bounding box.

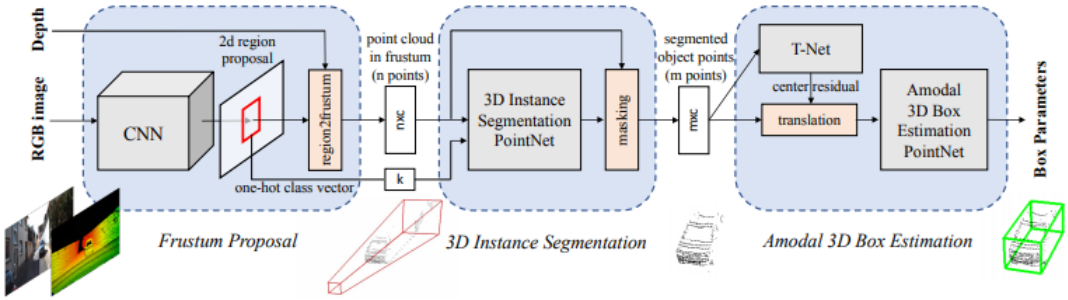


Figure 5.2: Frustum PointNets Network Architecture

5.1.3 VoxelNet

The best thing about VoxelNet is that it unifies feature extraction and bounding box prediction into a single phase. This is different from other recent approaches for dealing with point cloud data where pre-processing is done to convert it first to Bird's-Eye View or Front View before using an image-based CNN object detection model such as [24].

Knowing that one point cloud is just a group of points in the 3D space around the sensor mapping the environment, VoxelNet divides this point cloud into equally spaced 3D voxels. Now, every voxel contains a set of points. After dividing the point cloud into voxels, each voxel is encoded to each of the stacked layers implemented in the network. These middle layers are called voxel feature encoding (VFE) layers. Stacking multiple VFE layers allows learning very complex features for characterizing local 3D shape information. The VFE layers are great to avoid large 3D convolutions and handle the sparsity of point cloud data. After passing through all the VFE layers, several 3D convolution layers further aggregate local voxel features. During this step, the network transforms every group of points within each voxel into one feature representation. This step yields transforming every point cloud into a higher-dimensional volumetric representation. After that, every processed point cloud is passed to an RPN to perform detections.
Figure 5.3 illustrates the architecture of VoxelNet. In the following subsections, we will discuss the main components of VoxelNet.

Feature Learning Network

Given a point cloud, its 3D space is divided into equally spaced voxels. Each voxel has a depth, height and width. Then, every set of points inside these newly-created voxels is grouped together. Voxels may have different number of points. After that, a fixed number of points is sampled from those voxels. This is because one point cloud typically contain approximately 100K points. This would increase computing complexity and memory usage. This helps also decrease the imbalance of points between voxels.

The next step is that every voxel is passed through a fully connected network (FCN). Only non-empty voxels are processed because more than 90% of voxels are typically empty. Thus, This reduces memory usage and computation complexity. FCN transforms the voxel to into a feature space where the shape of the surface contained within the voxel can be encoded. This FCN is composed of a linear layer, a batch normalization layer (BN), and a rectified linear unit layer (ReLU). After that, a maxpooling layer gets the locally aggregated features. Finally, an output feature set is obtained.

Convolutional Middle Layers

Each convolutional middle layer applies 3D convolution, BN layer, and ReLU layer sequentially. The convolutional middle layers aggregate voxel-wise features within a progressively expanding receptive field, adding more context to the shape description.

Region Proposal Network

VoxelNet makes use of the RPN architecture proposed in [24] to perform object detection. The input to the RPN is the feature map provided by the convolutional middle layers discussed above. The RPN has three blocks of fully convolutional layers. The first layer of each block downsamples the feature map by half via a convolution with a stride size of 2, followed by a sequence of convolutions of stride 1. After each convolution layer, BN and ReLU operations are applied. The output of every block is then upsampled to a fixed size and concatenated to construct a high resolution feature map. Finally, this feature map is mapped to the desired learning targets: a class probability score map and a regression map.

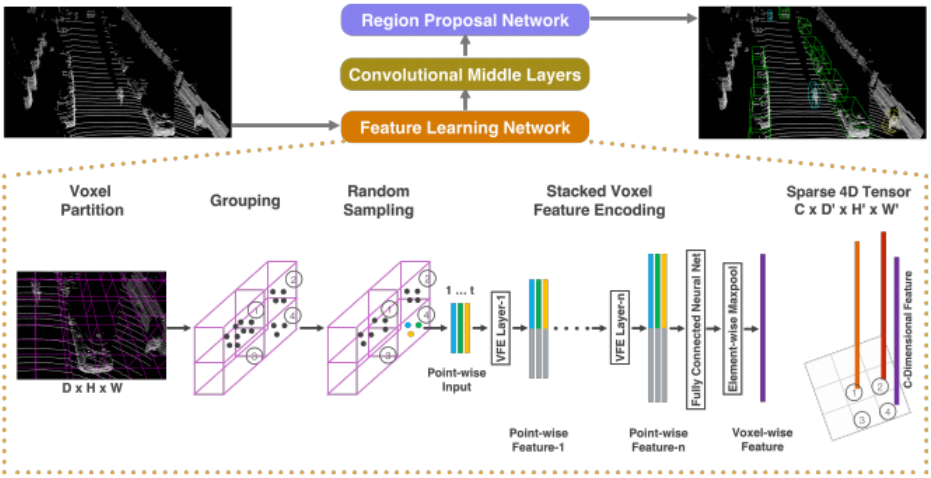


Figure 5.3: VoxelNet Network Architecture

Network Training for Car Detection

First of all, all point clouds that are outside the boundaries of the corresponding 2D image are removed. This results in cropping all point clouds before training. Voxel size was chosen to be as follows: Depth = 10, Height = 400 and Width = 352. The maximum number of randomly sampled points in each non-empty voxel is set to be: T = 35.

After that, two VFE layers were used: VFE-1(7, 32) and VFE-2(32, 128) and three convolutional middle layers were used: Conv3D(128, 64, 3, (2,1,1), (1,1,1)), Conv3D(64, 64, 3, (1,1,1), (0,1,1)), and Conv3D(64, 64, 3, (2,1,1), (1,1,1)). After reshaping, the input to the RPN is a feature map of size 128 400 352, where the dimensions correspond to channel, height, and width of the 3D tensor.

A bounding box is considered as positive if it has the highest IoU with a ground truth or its IoU with the ground truth is above 0.6 (in birds eye view). A bounding box is considered as negative if the IoU between it and all the ground truth boxes is less than 0.45. Bounding boxes are treated as do not care if they have 0.45 IoU 0.6 with any ground truth.

5.2 Milestone II: Setup

The second milestone was concerned with setting up the development environment. We will discuss the two environments used in details in the following subsections.

5.2.1 Setting Up Linux

With the start of this milestone, there was a lot of experimentation done to find the most suitable Linux distribution and memory configuration settings in order to proceed. Then, Linux Ubuntu 16.04 LTS was downloaded and installed in order to use Debian packages for development. After that, the next step was to learn how to efficiently use Linux Terminal and understand the Linux file management system.

5.2.2 Preparing Data

In this phase, 1000 images (800 MB) were chosen from the KITTI dataset [16] for predicting on both VoxelNet [27] and Frustum PointNets [22]. The images had their corresponding labels (for evaluation), camera calibration matrices and point cloud (depth information) files (1.4 GB). After that, the data was organized as in figure 5.4.

```
└── DATA_DIR
    ├── training   <-- training data
    |   ├── image_2
    |   ├── label_2
    |   └── velodyne
    └── validation  <-- evaluation data
    |   ├── image_2
    |   ├── label_2
    |   └── velodyne
```

Figure 5.4: Data Preparation

5.2.3 Setting Up Frustum PointNets Environment

This setup was done according to the official Github repository associated with the project [4]. The CPU used for prediction was the Dual-Core Intel Fifth Generation i5-5200U with a 2.20 GHz clock speed and 8 GB of RAM.

First, a Python 2.7 virtual environment was installed. Python virtual environments

are isolated from other Python development environments, thus they are incapable of interfering with or being affected by other Python programs on the same machine. This provides a safe and reliable way for installing and running desired packages and is useful for avoiding conflicts.

The dependencies installed on top the virtual environment were:

- **Tensorflow 1.4**: Deep Learning framework to deal with Neural Networks.
- **OpenCV**: Computer Vision software library that takes advantage of multi-core processing.
- **Mayavi**: Library used for 3D point cloud visualization.

Second, all the scripts were modified to support testing on our 1000 images and dumping results. During this process, frustum point clouds were extracted from the 1000 images along with ground truth labels from the original KITTI data, based on ready precomputed 2D object detections.

Third, the script in [8] was compiled to be able to evaluate 3D Object Detection and Bird's-Eye View based on their average precision (AP) after predicting on the 1000 samples was done. The evaluation script required the dependency of C++ Boost Libraries.

5.2.4 Setting Up VoxelNet Environment

This setup was done according to the unofficial Github repository [11] which is a tensorflow implementation of VoxelNet. The GPU used for prediction this time was the NVIDIA TITAN Xp with a memory speed of 11.4 Gbps, 12 GB of RAM, 3840 CUDA cores and compute capability of 6.1.

CUDA Toolkit 8.0 and **cudNN 6.0** were used for testing on top of the latest Nvidia driver version (R390). **CUDA** is a parallel computing platform and API model created by NVIDIA. It enables GPU cores to collectively run thousands of computing threads and, thus, achieves huge increases in computing performance by harnessing the GPU power. **cudNN**, on the other side, is a GPU-accelerated library of primitives for deep neural networks. It provides functionality for common operations on deep neural networks.

First, a Python 3.5 virtual environment was installed, upon which were installed the following dependencies:

- **Tensorflow 1.4**
- **OpenCV**

- **Numba**: Python function compiler. It gives the power to speed up applications with high performance functions written in Python.
- **Shapely**: Python package for manipulation and analysis of geometric objects in the Cartesian plane.
- **Easydict**: Library that enables access to dict values as attributes.

Second, the Velodyne point cloud files associated with the 1000 images were cropped to remove point clouds that exist outside the corresponding image coordinates.

5.3 Milestone III: Tests and Results

Results reproduced from predicting on the 1000 samples and visualizations dumped from testing are both demonstrated in this section. To produce fair results, the 1000 samples used for predicting and the evaluation script used to evaluate predictions were the same for both VoxelNet and Frustum PointNets.

The pre-trained model used for VoxelNet was the one provided on [12]. While that used for Frustum PointNets was provided on [4]. Car Detection was done on both VoxelNet and Frustum PointNets, but Pedestrian and Cyclist Detection was done only on Frustum PointNets.

The evaluation script [8] computes precision-recall curves and average precision for fully visible, partly occluded and heavily occluded objects. Reproduced results are then compared with those reported on [10] for VoxelNet and [5] for Frustum PointNets. Detected objects fall accordingly under one of the three states according to their degree of occlusion:

- **Easy**:
  - Minimum bounding box height: 40 px
  - Maximum occlusion level: Fully visible
  - Maximum truncation: 15%
- **Medium**:
  - Minimum bounding box height: 25 px
  - Maximum occlusion level: Partly occluded
  - Maximum truncation: 30%
- **Hard**:
  - Minimum bounding box height: 25 px
  - Maximum occlusion level: Heavily occluded
  - Maximum truncation: 50%

5.3.1 Car Detection

Bird's-Eye View Car Detection

The Bird's-Eye View is an elevated view of an object from above, with a perspective as though the observer were a bird or an airplane. Figures 5.5 and 5.6 represent Bird's-Eye View Car Detection AP results and precision-recall curves respectively.

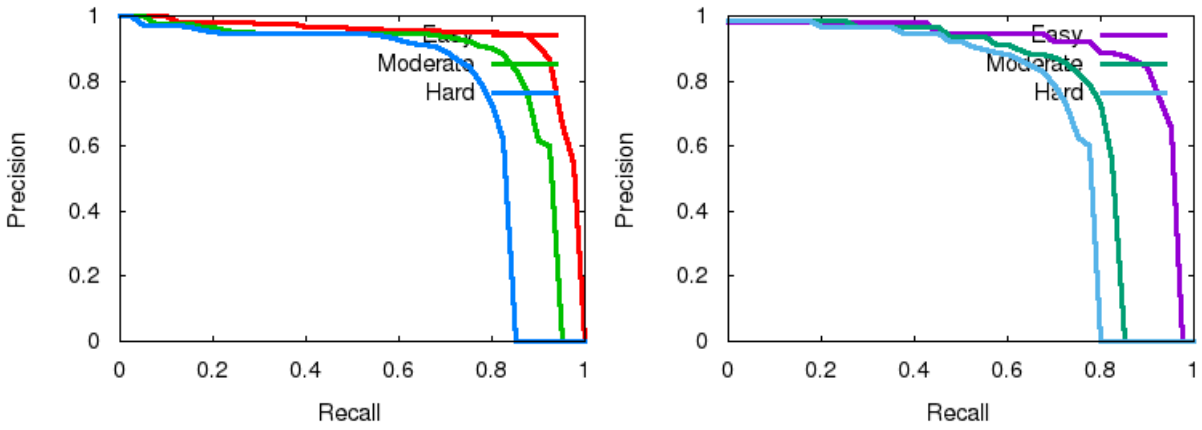
	Easy	Medium	Hard
Reported	88.70	84.00	75.33
Reproduced	87.70	83.45	75.51

(a) Frustum PointNets

	Easy	Medium	Hard
Reported	89.35	79.26	77.39
Reproduced	85.74	75.69	67.64

(b) VoxelNet

Figure 5.5: Bird's-Eye View Car Detection AP Results



(a) Frustum PointNets (b) VoxelNet

Figure 5.6: Bird's-Eye View Car Detection Precision-Recall Graphs

3D Car Detection

3D detections are represented by 3D bounding boxes that surround the objects. Figures 5.7 and 5.8 represent 3D Car Detection AP results and precision-recall curves respectively.

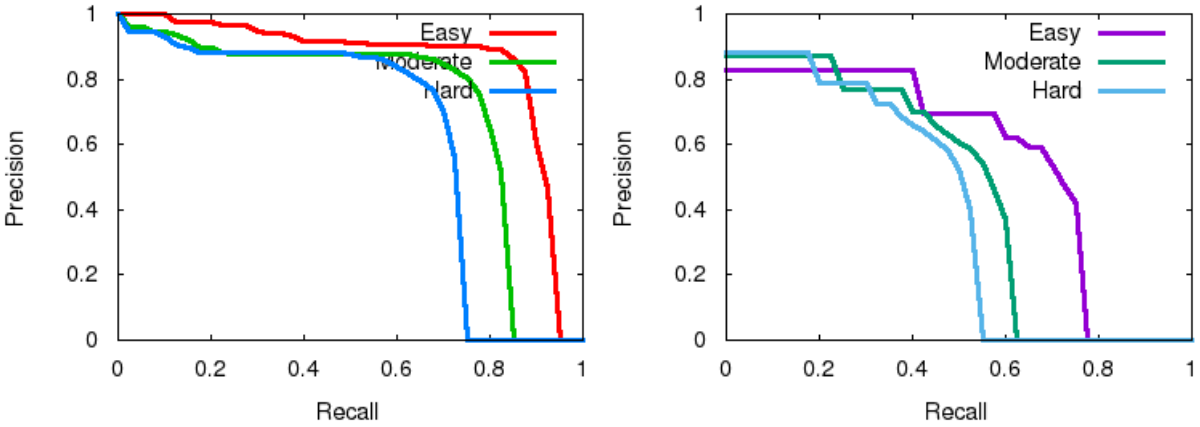
	Easy	Medium	Hard
Reported	81.20	70.39	62.19
Reproduced	82.27	71.36	63.60

(a) Frustum PointNets

	Easy	Medium	Hard
Reported	77.47	65.11	57.73
Reproduced	54.37	46.02	41.01

(b) VoxelNet

Figure 5.7: 3D Car Detection AP Results



(a) Frustum PointNets (b) VoxelNet

Figure 5.8: 3D Car Detection Precision-Recall Graphs

5.3.2 Pedestrian Detection

Figures 5.9 and 5.10 represent Pedestrian Detection AP results and precision-recall curves respectively.

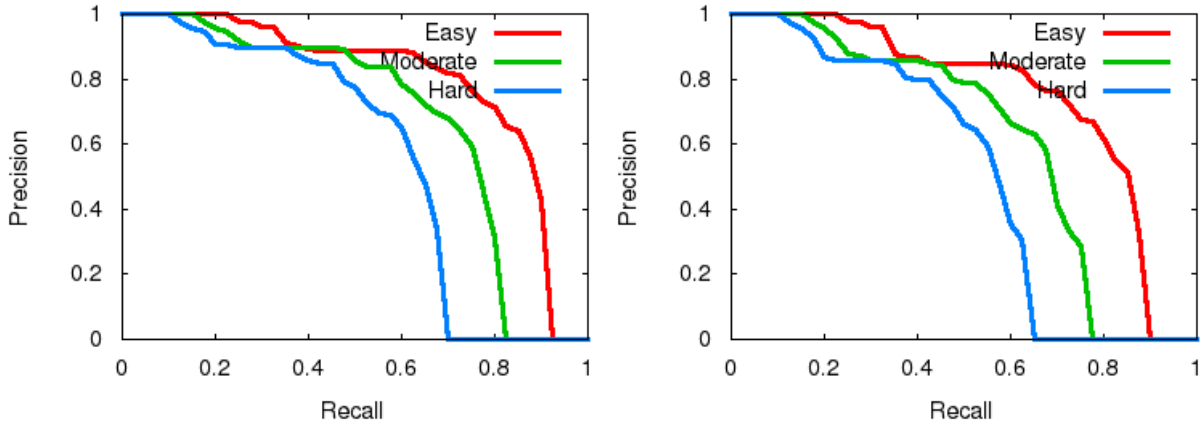
	Easy	Medium	Hard
Reported	58.09	50.22	47.20
Reproduced	78.05	67.11	55.31

(a) Bird's-Eye View Pedestrian Detection

	Easy	Medium	Hard
Reported	51.21	44.89	40.23
Reproduced	71.74	59.44	50.30

(b) 3D Pedestrian Detection

Figure 5.9: Pedestrian Detection AP Results



(a) Bird's-Eye View Pedestrian Detection (b) 3D Pedestrian Detection

Figure 5.10: Pedestrian Detection Precision-Recall Graphs

5.3.3 Cyclist Detection

Figures 5.11 and 5.12 represent Cyclist Detection AP results and precision-recall curves respectively.

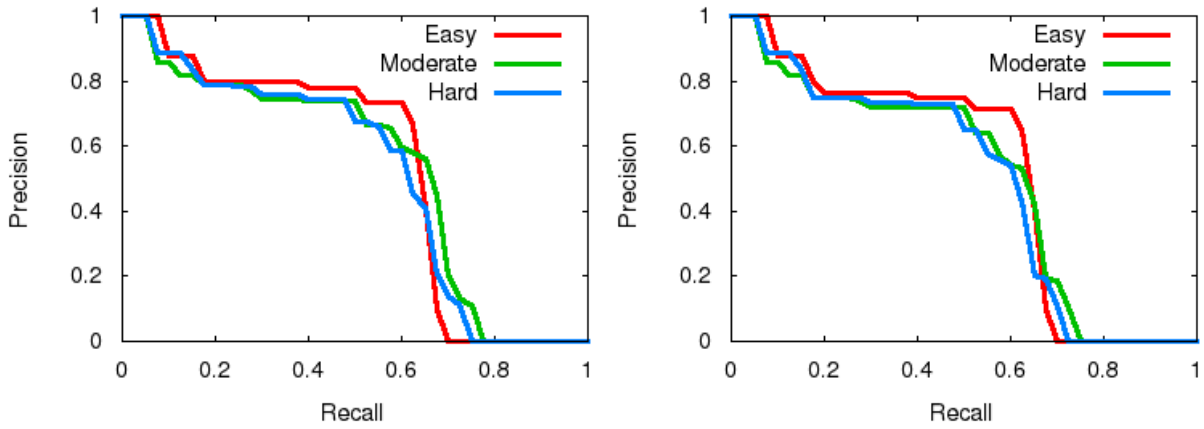
	Easy	Medium	Hard
Reported	75.38	61.96	54.68
Reproduced	52.42	51.49	50.73

(a) Bird's-Eye View Cyclist Detection

	Easy	Medium	Hard
Reported	71.96	56.77	50.39
Reproduced	51.03	50.00	49.06

(b) 3D Cyclist Detection

Figure 5.11: Cyclist Detection AP Results



(a) Bird's-Eye View Cyclist Detection (b) 3D Cyclist Detection

Figure 5.12: Cyclist Detection Precision-Recall Graphs

5.3.4 Visualzation

This subsection demonstrates how object detection through LiDAR is visualized. Four sample images were chosen to achieve this. The four images were tested on VoxelNet's pre-trained model available on [12]. The first image (figure 5.13) contains some cars moving on a highway that goes both directions.

Figure 5.13: Sample Image I

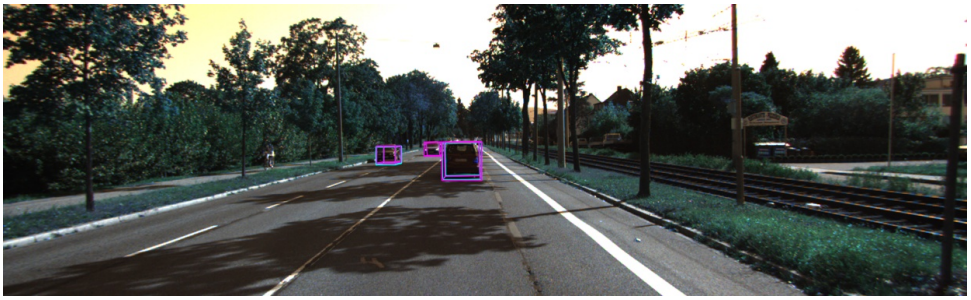


Figure 5.14: 3D Car Detection on Sample Image I

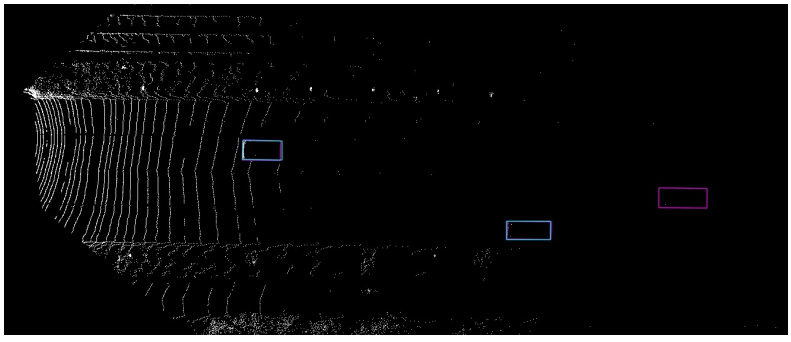


Figure 5.15: Bird's-Eye View Car Detection on Sample Image I

The second image (figure 5.16) contains some cars that are parked on both sides of the road. Some cars are semi-occluded.

Figure 5.16: Sample Image II

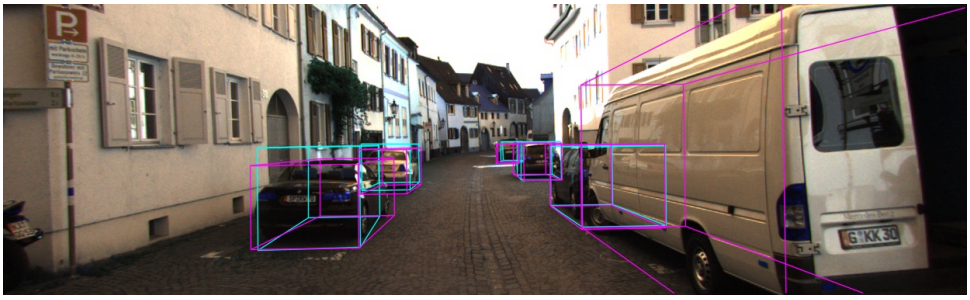


Figure 5.17: 3D Car Detection on Sample Image II

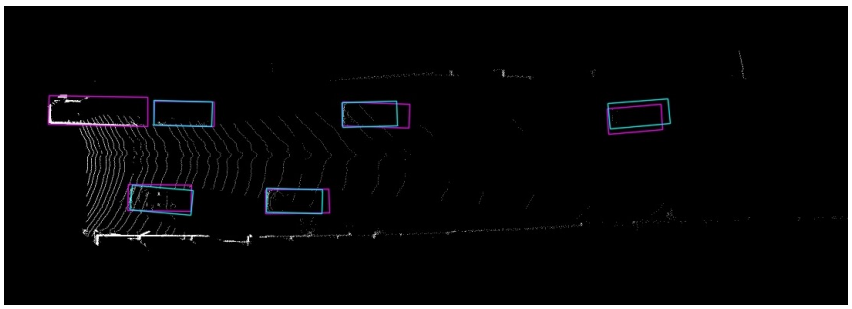


Figure 5.18: Bird's-Eye View Car Detection on Sample Image II

The third image (figure 5.19) is of a mixed environment that contains a number of moving as well as parking cars. Most of the cars in this image are far from the sensor.

Figure 5.19: Sample Image III

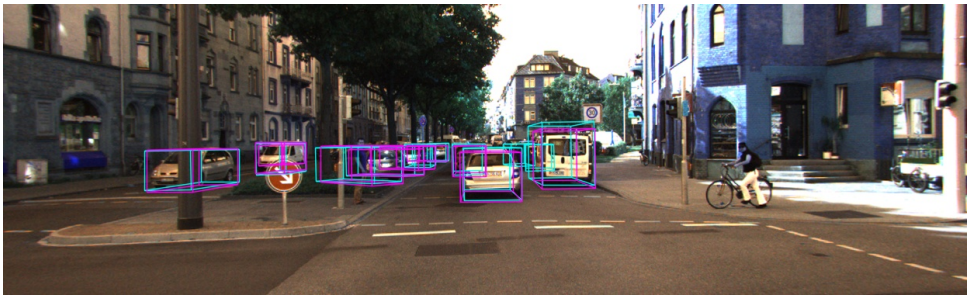


Figure 5.20: 3D Car Detection on Sample Image III

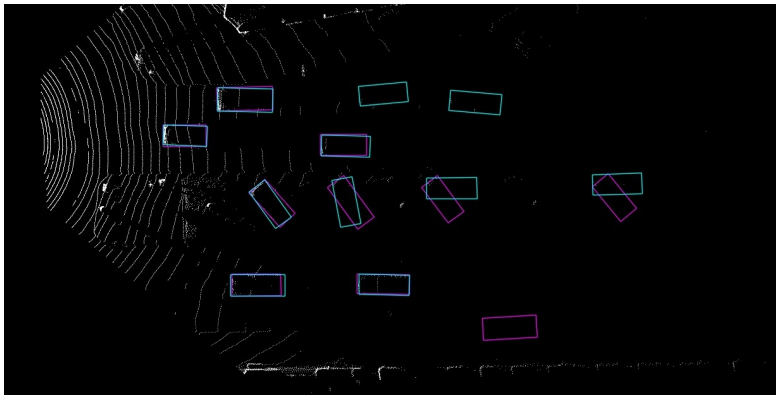


Figure 5.21: Bird's-Eye View Car Detection on Sample Image III

The last image (figure 5.22) contains no cars at all. The network did not detect any objects and therefore no boxes were drawn in figures 5.23 and 5.24.

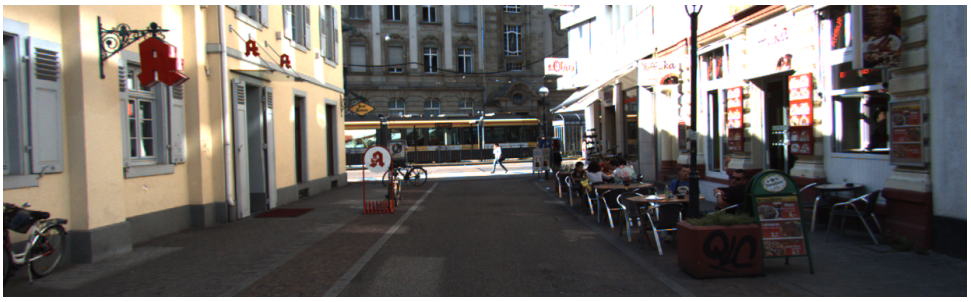


Figure 5.22: Sample Image IV

Figure 5.23: 3D Car Detection on Sample Image IV

Figure 5.24: Bird's-Eye View Car Detection on Sample Image IV

Chapter 6

Conclusion and Future Work

In this thesis, results for object detection from using both RGB images and their corresponding LiDAR point clouds were reproduced using two different methods on different benchmarks. The comparison between the reported and reproduced accuracies shows that the prediction results were at about the same range. Car Detection is more mature on both methods, but Pedestrian and Cyclist Detection classes still need a lot of training to improve results. Visualizations of 3D bounding boxes shows that it is of great importance for a self-driving car to predict more true positives to be able to take the right decision at the right time and avoid any losses. The future of autonomous driving is getting better as LiDAR sensors are getting lower in cost and higher in measuring range.

Future work includes producing results on Pedestrian and Cyclist Detection using VoxelNet as well as working on increasing the accuracy for both classes. Also, additional classes may be added such as: street signs, billboards and pavements.

Appendix

Appendix A

Lists

Nomenclature

AOS Average Orientation Similarity

AP Average Precision

BN Batch Normalization

CNN Convolutional Neural Network

CUDA Compute Unified Device Architecture

cuDNN CUDA Deep Neural Network Library

FCN Fully Connected Network

HOG Histogram of Gradients

ILSVRC ImageNet Large Scale Visual Recognition Challenge

IoU Intersection over Union

KITTI Karlsruhe Institute of Technology and Toyota Technological Institute

LiDAR Light Detection and Ranging

ReLU Rectified Linear Unit

ROI Region of Interest

RPN Region Proposal Network

VFE Voxel Feature Encoding

List of Figures

Bibliography

[1] CS229: Machine Learning. http://cs229.stanford.edu/.

[2] CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.stanford.edu/2017/.

[3] Efficient Graph-Based Image Segmentation. http://cs.brown.edu/people/pfelzens/segment/.

[4] Frustum PointNets for 3D Object Detection from RGB-D Data Official Implementation. https://github.com/charlesq34/frustum-pointnets.

[5] Frustum PointNets Official Detailed Results. https://goo.gl/Jvod7F.

[6] KITTI 3D Object Detection Benchmark Leaderboard. https://goo.gl/puC2Lk.

[7] KITTI Bird Eye View Object Detection Benchmark Leaderboard. https://goo.gl/6996XA.

[8] KITTI Object Detection Evaluation Script. https://github.com/prclibo/kitti_eval.

[9] Region of interest pooling explained. https://goo.gl/MJL9f1.

[10] VoxelNet Official Detailed Results. https://goo.gl/Z4LBRm.

[11] VoxelNet Tensorflow Implementation by @jeasinema. https://github.com/jeasinema/VoxelNet-tensorflow.

[12] VoxelNet Tensorflow Implementation by @qianguih. https://github.com/qianguih/voxelnet.

[13] Hubel & Wiesel Neural Basis of Visual Perception Experiment. https://goo.gl/vZYsWx. 2014.

[14] Maureen Caudill. Neural Network Primer: Part I. 1987.

[15] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detections. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[16] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[17] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[18] Ross Girshick. Fast R-CNN. *arXiv preprint arXiv:1504.08083*, 2015.

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1311.2524*, 2013.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. 2012.

[21] Tsung-Yi Lin, Piotr Dollr, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. *arXiv preprint arXiv:1612.03144*, 2016.

[22] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. http://stanford.edu/~rqi/frustum-pointnets/. *arXiv preprint arXiv:1711.08488*, 2017.

[23] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv preprint arXiv:1612.00593v2*, 2017.

[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv preprint arXiv:1506.01497*, 2015.

[25] Uijlings, J. R. R. and van de Sande, K. E. A. and Gevers, T. and Smeulders, A. W. M. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

[26] Paul Viola and Michael Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[27] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. *arXiv preprint arXiv:1711.06396*, 2017.