Pedestrian Detection Based on Deep Neural Network for Autonomous Vehicle Using Multimodal Information

by

Olagoke Lukman Olabisi

A **Project** submitted

in the
School of Electrical Electronics Engineering
Laboratory of Signal Processing 5

Supervised by: Prof Jean-Philippe Thiran
Assisted by: Damien Matti

December 2017

For now that you have made that decision,
to spread your wings
then soar the air,
taking off against the turbulent current,
away from comfort and the safe zones,
against that others call common sense,
Soonest, your doubts, fears, insecurities, kick in,
lining up in a firing squad fashion,
ready to shoot you out of the sky,
so just make sure this is what you want,
For this is just the beginning
and this can only get harder,
Alas, there is no free lunch here,
But there is hardwork, faith and progress!

Lukman's Library
Adapted from [1]

Abstract

Autonomous driving represent a highly disruptive innovation for the future of road transport capable of influencing aspects as fundamental as road safety and mobility itself [2]. Accordingly for acceptability into roadways, autonomous vehicles would be required to improve the extant traffic safety - for example by reducing the traffic collisions. However, since pedestrians represent one of the most unpredictable actors in a scene with road traffic [2] there is need for robust method for pedestrian detection. This project describes a multi sensor-based system to detect pedestrians in an autonomous vehicle.

In particular, this project aims to develop multi sensor-based system for pedestrian detection using convolutional neural network (CNN). Earlier research work have used techniques such as region proposal networks (RPN) to speed-up the image detection. However, this project exploits the use of feature extraction on additional information (in the form of proposal from another sensor) in order to create better *descriptor*. This descriptor can then provide auxiliary information for CNN-based approaches. .

Acknowledgements

Many people have helped shaped my ideas and from whom I have learnt a lot. I specially thank anyone whose name would not be mentioned here but whose life or knowledge has influence me in any way.

I will like to thank Professor Jean-Philippe Thiran of the Laboratory of Signal Processing 5, EPFL, for offering me the opportunity and support to work on this project. A lot of thanks also goes to the assistant supervisor, Mr. Damien Matti (Ph.D Supervisor at EPFL) for his constant support and mentoring while I work on the project.

My special thanks also goes to Mr. and Mrs Adebayo for their support over the years. In addition, I thank my siblings; I must say I love you all.

Contents

List of Figures

List of Tables

Abbreviations

Acronym	What (it) Stands For.
LAS	**LAS**er File Format
LIDAR	**L**ight **D**etection **A**nd **R**anging
RPN	**R**egion **P**roposal **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
AQI	**A**ir **Q**uality **I**ndex
ppm ppb ppt	**P**arts **P**er **M**illion / **B**illion / **T**rillion
PM	**P**articulate **M**atter
VIM	**V**isualisation and **I**mputation of **M**issing values
ACF	**A**uto **C**orrelation **F**unction
PACF	**Partial A**uto **C**orrelation **F**unction

Dedicated to my parents and Mr. Ashade.

Chapter 1

The Simple Vision Model

In order to be able to transfer specific capabilities of human visual perception to the autonomous vehicle, we need to gather information about the real world using different sensors types (for example: LiDAR,Camera). The different sensor types establishes a mapping from scene onto data [3]. Accordingly, the different sensor types would result data in in the form of 2D imagery and 3D point cloud. The measured data from different sensor types contain information that might help us reduce the uncertainty about the task at hand: Pedestrian recognition.

This chapter introduces the concept of autonomous vehicle system and goes on to show where the vision system fits in the overall architecture. Furthermore basic motivation for vision system using the simple pinhole camera is developed. This simple model is then extended to show how 3D point cloud is generated. This 3D point cloud together with the accompanying 2D images will be used in the the later chapter for classification and identification purposes

1.1 Autonomous Vehicle: A General Overview

Autonomous Vehicle is a vehicle that is capable of sensing its environment and navigating without human input [4]. Autonomous vehicles promise to be a game-changer in both personal and industrial transport. Generally, traffic safety would be expected to improve with increase in autonomous vehicles on roadways. This is because autonomous systems, in contrast to human drivers, have faster reaction times and are fatigue-proof in their functioning [2]. Thus, the number of traffic collisions, dead and injured passengers, and damage to the environment will be reduced[2]. New business models, such as mobility as a service, which aims to be cheaper than owning a car, would

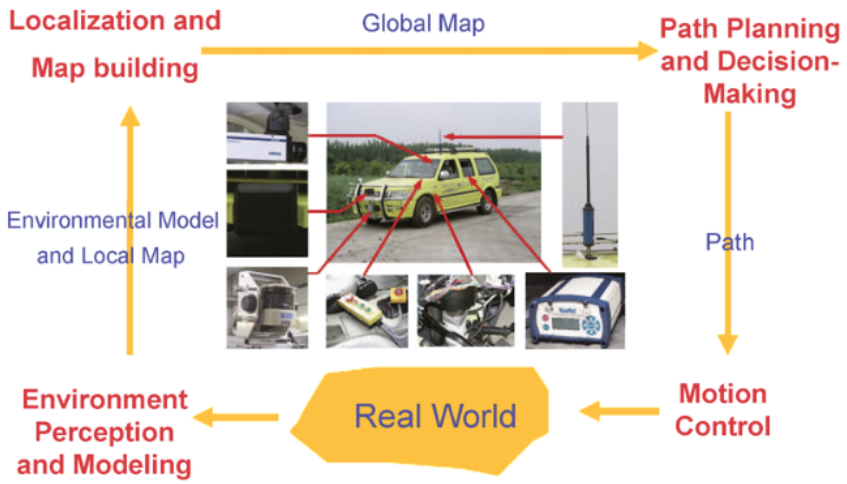
be available[2]. Since no driver is required, the labour cost associated with transporting people and goods is reduced. Finally, with ubiquitous internet connection, it would be possible for vehicles and roadside units to share information in peer-to-peer networks[2]. These networks would reduce traffic congestion in cities and roadways, improve safety, and even decrease the need for vehicle insurance [2].

1.1.1 Autonomous Vehicle Modules

Autonomous vehicle basically consists of four fundamental technologies (Fig 1.1) [5]:

- environment perception and modeling
- localization and map building
- path planning and decision-making
- motion control.

Figure 1.1: framework for autonomous vehicle



[Image Source [5]]

The environment perception and modeling module is responsible for sensing environment structures in a multi-sensor way and providing a model of the surrounding environment[5]. The environmental model typically includes a list of objects in the environment - moving objects, that of static obstacles, vehicle position relative to the current road, the road shape, pedestrians etc. This module typically passes the environment model and the local map to the localization and map building module after processing the original data.

The goal of the second module, vehicle localization and map building, is to use geometric feature location estimate in the map to determine the vehicles position, and to interpret the sensor information to estimate the locations of geometric features in a global map . Thus, the second module yields a global map based on the environment model and a local map [5].

The path planning and decision-making module is to assist in ensuring that the vehicle is operated in accordance with the rules of the ground, safety, comfortability, vehicle dynamics, and environment contexts. Hence, this module aims to improve mission efficiency and generate the desired path [5].

The final module, motion control, is to execute the commands necessary to achieve the planned paths, thus yielding interaction between the vehicle and its surrounding environment[5].

This project will focus on the Environmental perception module and modelling aspect. In particular the aim is to design an architecture capable of detecting pedestrian in an accurate and timely fashion.

1.1.2 Multi-sensor Based Environment perception and Modeling

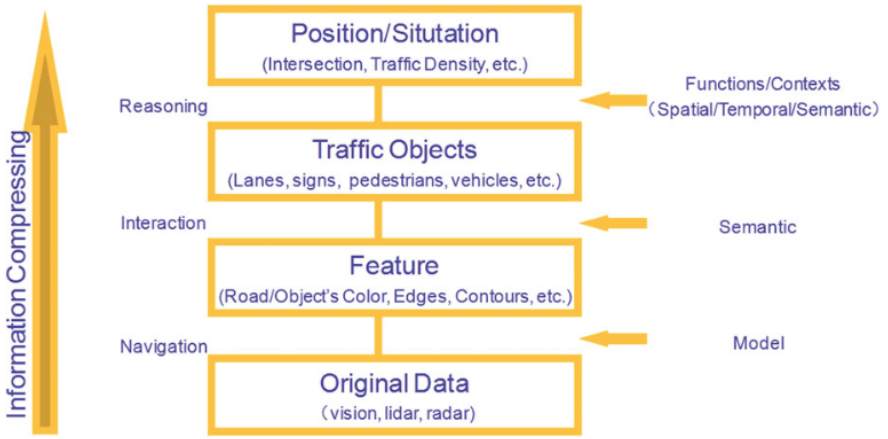
Figure 1.2 illustrates a general environment perception and modeling framework. The frame is explained below

1. The original data are collected by various sensors
2. Various features are extracted from the original data, such as road (object) colors, lane edges, building contours.
3. Semantic objects are recognized using classifiers, and consist of lanes, signs, vehicles, pedestrians
4. We can deduce driving contexts, and vehicle positions

- Multi-sensor Fusion

  Multi-sensor Fusion is the basic framework of intelligent vehicles for better sensing surrounding environment structures, and detecting objects/obstacles [5]. Sensors for detecting surrounding environment perception can either be active or passive. Active sensors include Lidar, Radar, ultrasonic and radio, while the commonly-used passive sensors are infrared and visual cameras. These various sensors are capable of providing different detection precision and range, and yielding different

Figure 1.2: Environmental Perception and modelling



[Illustration of basic component of imaging device. Source [5]]

effects on environment. In particular, combining various sensors could cover not only short-range but also long-range objects/obstacles, and also work in various weather conditions.

In general, in a driving environment, we are interested in static/dynamic obstacles, lane markings, traffic signs, vehicles, and pedestrians. Correspondingly, object detection and tracking are the key parts of environment perception and modeling. This project focuses on the pedestrian detection task.

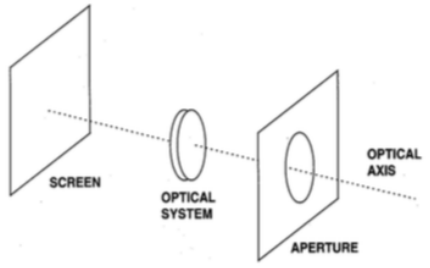
1.2 Simple Model for Perception: The Pin Hole Camera

To get a good start on how perception mechanism actually works, we start by building an intuition using a very simple model: the pin hole camera. Firstly, we note that the core focus of any imaging system are *digital images*. We draw a division between intensity images and range images [6]. *Intensity images* are the familiar, photograph-like images encoding light intensities acquired by cameras for example. Often, they measure the amount of light impinging on photosensitive device. *Range images* encode shape and distance between a known reference frame and a visible point in the scene. They are often acquired by special sensors like LIDAR, RADAR and SONARS. We emphasize that any digital image irrespective of type consist of a 2-D array of numbers. The numbers might represent light intensities, distances or other physical quantities, depending on the nature of actual image[6].

1.2.1 Basis Element of Image Device

The process for image formation in the simple model for perception system begins with light rays which enter the camera through an *angular aperture or pupil*, and hit a screen or *image plane*, which is the vision's system photosensitive device which register's image intensities, see Fig 1.3. We note that most of these rays resulted from reflections of the rays emitted from a light sources and hitting the object surfaces [6].

Figure 1.3: Composition of a Simple Vision Device



[Illustration of basic component of imaging device. Source [6]]

Any single point of a scene reflects light coming from possible all directions. Thus it is possible that many rays reflected from same point enters the imaging device eg (camera). However to obtain a sharp image, all rays coming from the same point should converge onto a single point on the image plane. When this happens we say the image is properly focused. Focusing can be achieved broadly in 2 ways [7]:

- Reducing the camera aperture to a point. This gives the *pin-hole camera.*
- using an optical system such as lenses, and other elements designed to make all rays from the same 3D point converge to a point.

1.2.2 Mathematical Model Of Pin Hole Camera

Figure below shows a pinhole camera. The operation of the pinhole camera consist of a small hole, *pinhole or optical centre* through which rays from object in real world passes through to form an inverted image on the back of the image box, called image plane. For convenience we consider a virtual image generated by placing the image plane in front of the pin hole camera.

Points in 3D world are represented as, $\mathbf{w} = \begin{bmatrix} u, & v, & w \end{bmatrix}^T$, and we assume that the optical centre is at the centre of this 3D world. The virtual image is formed on the image plane, which itself is slightly displaced from the optical centre along the *w-axis.* The *principal* point defines the point where the optical axis strikes the image plane.

The point where the optical point strikes the image plane is called *focal length*. The task consist in establishing correspondence between the point $\mathbf{x} = \begin{bmatrix} x, & y \end{bmatrix}^T$ in the image plane, and the the corresponding 3D point $\mathbf{w} = \begin{bmatrix} u, & v, & w \end{bmatrix}^T$ of the real world image. The image position $\mathbf{x}$ can be found by observing where the rays from the 3D real image strikes the image plane. This process is called *perspective projection*. In a *Normalized camera*, we have the focal length to be 1 and assume that the origin of the 2D coordinate system of the image plane, is centred at the principal point. A 3D point $\mathbf{w} = \begin{bmatrix} u, & v, & w \end{bmatrix}^T$ is then projected onto the image plane, $\mathbf{x} = \begin{bmatrix} x, & y \end{bmatrix}^T$, by using using similar triangle relationship [7]. This gives:

$$x = \frac{u}{w} \qquad \text{and} \qquad y = \frac{v}{w}$$

.

To make the model more realistic we will correct the assumptions that the focal length is one. Further to this, the final position in the image plane is measured in pixels and not distance so that we have to factor in photo-receptor spacing. To correct both of these assumptions, we add a scaling factor along each axis of image plane [7]. This gives:

$$x = \frac{\phi_x u}{w} \qquad \text{and} \qquad y = \frac{\phi_y v}{w}$$

These parameters are called the *focal length parameters*.

We are not yet done; we know that the pixel position $\mathbf{x} = \begin{bmatrix} 0, & 0 \end{bmatrix}^T$ is at the top left for most images rather than at the centre in which the principal point is positioned. To correct this we add *the offset parameters* [7]. This gives:

$$x = \frac{\phi_x u}{w} + \delta_x \qquad \text{and} \qquad y = \frac{\phi_y v}{w} + \delta_y$$

We also add the *skew parameter* γ that moderates the projected position x as a function of the height v in the real world:

$$x = \frac{\phi_x u + \gamma v}{w} + \delta_x \qquad \text{and} \qquad y = \frac{\phi_y v}{w} + \delta_y$$

Finally we account for the fact that the camera is not usually aligned at the origin of the world coordinate system - in which we assume the optical axis is aligned with w-axis. Thus, we need a coordinate transformation from the world points w to the camera coordinate system [7]. This is achieved with the following transformation:

$$\mathbf{w}' = \mathbf{\Omega}\mathbf{w} + \boldsymbol{\tau}$$

Or expressed in full matrix form as:

$$\mathbf{w}' = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{21} & \omega_{22} & \omega_{23} \\ \omega_{31} & \omega_{32} & \omega_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$$

Incorporating this into the equation for x and y we have:

$$x = \frac{\phi_x(\omega_{11}u + \omega_{12}v + \omega_{13}w + \tau_x) + \gamma(\omega_{21}u + \omega_{22}v + \omega_{23}w + \tau_y)}{\omega_{31}u + \omega_{32}v + \omega_{33}w} + \delta_x$$

$$y = \frac{(\omega_{21}u + \omega_{22}v + \omega_{23}w + \tau_y)}{\omega_{31}u + \omega_{32}v + \omega_{33}w} + \delta_y$$

The parameters $(\phi_x, \phi_y, \gamma, \delta_x, \delta_y)$ are the *intrinsic parameters* because they describe the camera itself. On the other hand, $(\mathbf{\Omega}, \boldsymbol{\tau})$ are the *extrinsic parameters* since they describe the position and orientation of the camera in the world. However the main point of going through these details is to bring out the 3 fundamental geometric problems: to which we now turn to [7].

1.3 Three Geometric Problems

We itemize the 3 fundamental geometric problems below [7]:

- Perspective-n-Point problem: the goal here is to recover the position and orientation of the camera relative to a known scene.
- Calibration: the goal here is to estimate the intrinsic parameter
- Inferring 3D world points (Multi view reconstruction)

By far we are interested here in the third problem which is inferring 3D world points. The goal here is to estimate the 3D coordinate of a point $\mathbf{w}$ in the scene, given its position in $\{x\}_{j=1}^{J}$ in J $\geq$ 2 cameras. When J=2, this is called calibrated stereo reconstruction and when J > 2, it is called multiview reconstruction [7]. If we repeat this process for many points, the result would be a 3D point cloud which could be used to help an autonomous navigate an environment, or to generate images of the scene from a new view point [7]. In this project the point cloud will be acquired using LiDAR

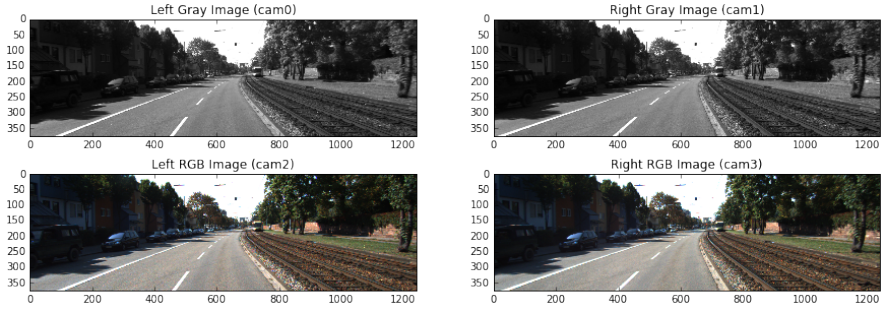
1.4 3D Point Cloud Acquisition With LiDAR

The term point in geometry refers to a unique location in a specific space. In particular, considering the euclidean space $\mathcal{R}^D$, we obtain a 2D image when $D = 2$ and obtain scanned 3D points (or reconstructed 3D point) for $D = 3$ [3]. It should be noted that a point has no dimensions such as length, volume or area[3].

Point cloud would formally refer to a set of data points in a given space. The focus is mainly on 3D point cloud representing the measured or generated counterpart of the physical scenes. The 3D point clouds are characterized by spatial location XYZ-coordinates and may optional be assigned additional attributes such as intensity or any abstract information.

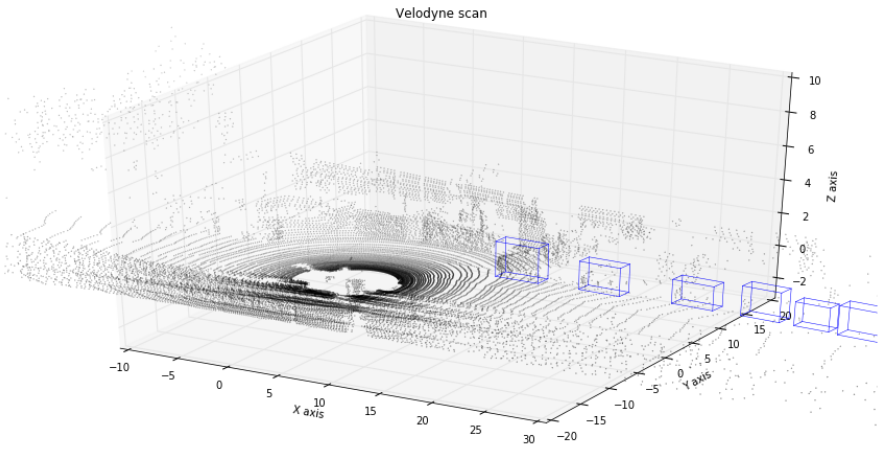
The point cloud are acquired using LiDAR. This point cloud together with the accompanying images are already provided in the kitti data set [8]. Below is what a typical image and point cloud look like in Kitti Data set.

Figure 1.4: Typical camera Image in Kiti Data set



[Camera Image, preprocessed from [8].]

Figure 1.5: Point Cloud corresponding to Camera Image Above from Kitti Data set



[Adapted from [8]. The blue bounding box correspond to position of the cars. The Kitti Dataset uses Velodyne LiDAR]

1.4.1 LiDAR SYSTEM

This would not be a place to discuss extensively about LiDAR. However, a brief information about it, necessary for following the discussion in this project is provided.

LiDAR (Light Detectiona and Ranging) refers to a remote sensing technology that emits intense, focused beams of light and measures the time it takes for the reflections to be detected by the sensor. This information is used to compute ranges, or distances, to objects. In this manner, LiDAR is analogous to radar (radio detecting and ranging), except that it is based on discrete pulses of laser light [9].

A typical LiDAR system would include [10]:

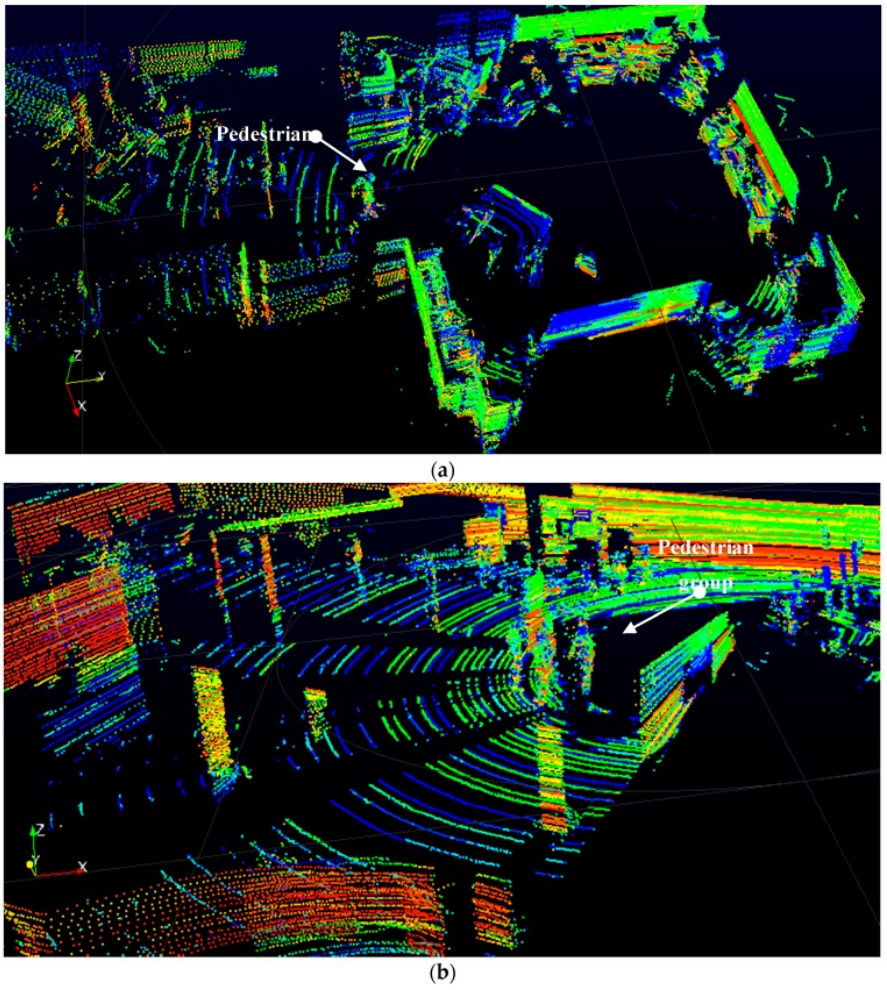
- Laser scanner
- High-precision clock
- GPS
- IMU or Inertial navigation measurement unit
- Data storage and management systems

Typically the LiDAR system scans an area by sending out a pulse). Knowing the speed of light and obtaining the time for the return(reflected pulse) from precision clock then it is a matter of simple calculation to compute the distance to the target point.

However, to get more accurate results, the information from GPS and IMU are taken into considerations and the final coordinate is appropriately calculated and corrected. A collection of these returns would form the LiDAR point cloud.

The Figure below shows LIDAR data points and the detection of pedestrian:

Figure 1.6: LiDAR point cloud and Pedestrian Detection



[Taken from [2]. The arrow points to position of pedestrian in point cloud]

1.5 Conclusion

In this chapter a general overview of what the environmental perception and modelling aspect of an autonomous vehicle is all about was discussed. In addition a simple model of pin hole camera was used to show how point clouds could be generated in addition to the Camera images. The main objective of this project would be to show the usefulness of LiDAR data as an additional source of *priori* information for reducing the search space during object classification for pedestrian identification.

The next chapter discusses how we carry out image classification. This follows intuitively from the fact that, having acquired the data using various sensors, our aim would be to do feature extraction and recognize semantic object (i.e pedestrian)

Chapter 2

Image Classification

2.1 Introduction

Having gathered information about the real world using different sensors types, the ultimate aim would be to be able to identify various objects in the scenes. This would be the topic of this chapter.

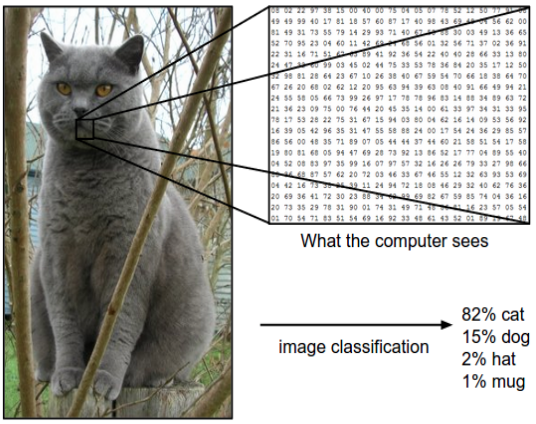
An obvious approach would be to write an algorithm that recognise each distinct instant of pedestrian in a scene. Such algorithmic representation would be required to be invariant to Viewpoint variation, Scale variation, deformation, occlusion, illumination conditions, background clutter, intra-class variations [11]. It turns out that such approach would not be robust and has yet it is not obvious how one might write an algorithm for identifying pedestrians in different scenerio.

It turns out that the data driven approach would be a viable option. In this approach, we a provide the computer with the scenes we have acquired from various sensors and then develop learning algorithms that look at these examples and learn about the visual appearance pedestrian in the scenes. This approach is referred to as a data-driven approach, since it relies on first accumulating a training data-set of labeled pedestrian images from the sensors.

2.1.1 Simple Image Classification Model

As noted in the previous chapter, to a computer an image is represented as one large 3-dimensional array of numbers as shown in the image, fig 2.1. The task is to intelligently use this "dumb numbers" for decision making: that is recognising the cat.

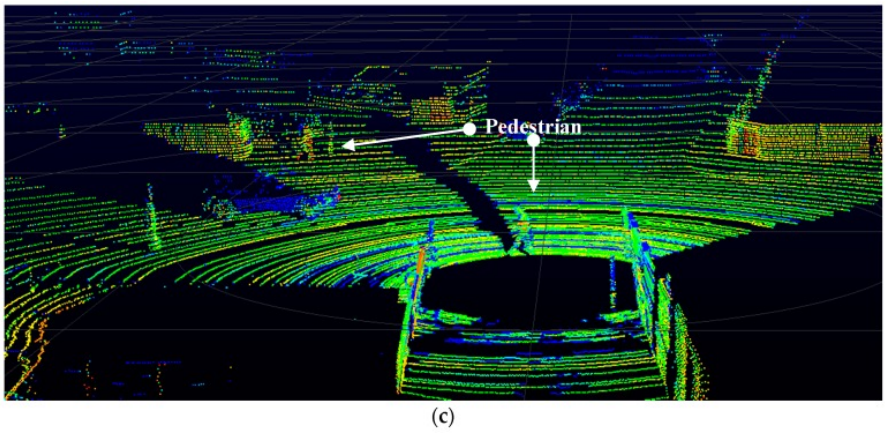
Figure 2.1: Simple Cat Classification



[Image from Stanford Lecture note on: Convolution neural network for Visual recognition [11].]

In this project, the aim is to recognise the pedestrians in a scene captured by sensors, fig 2.2:

Figure 2.2: Pedestrian recognition Illustrated



[Image from online source, [2].]

A simplified pipeline for onject recognition is provided below:

- Input: the input consists camera captured images of a scene. We refer to this data as the training set. The data from the 3D LiDAR will also be used to augment the input data set. Both the image and point cloud are in the Kitti data set.
- Learning: The aim is to use the training set to learn to recognise pedestrians in the scene. This step is called Learning a model or *training a classifier*[11].
- Evaluation: In the end, it is necessary to evaluate the quality of the classifier by presenting to it a new set of scenes test set, and ask it to detect pedestrians. A

score will then be assigned base on how well pedestrians in a scene are detected. Intuitively, we are hoping that a lot of the predictions match up with the true answers (which we call the ground truth [11]).

2.2 Learning Models

2.2.1 Nearest Neighbour and K-Nearest Neighbour

Though it is rarely use, the nearest neighbour classifier is a good starting point. The nearest neighbor classifier takes a test scene or image, compare it to every single one of the training set, and try predict if it is a pedestrian or not. The aim would be to learn the pixel values that encapsulates the pedestrian position in the scene or image. To do the comparison, the L1 or L2 distance will be employed. Given 2 images say I_1 and I_2 , [11], the L1 distance is computed as so:

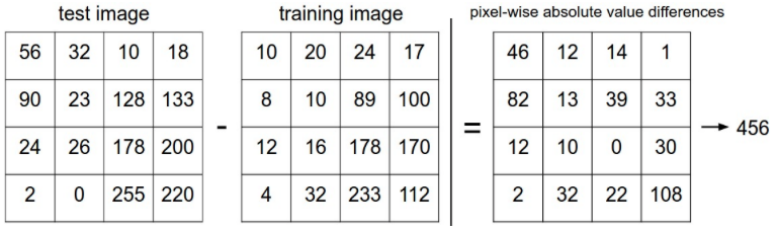
$$d_1(I_1, I_2) = \Sigma_p |I_1^p - I_2^p|$$

and the L2 distance :

$$d_2(I_1, I_2) = \sqrt{\Sigma_p (I_1^p - I_2^p)^2}$$

To put this into context, consider the simple illustration below fig 2.3. Assume the test image correspond to 2D image and we give it to nearest neighbour classifier to check if it is pedestrian or not. To do this it computes the L1 distance between the test image and training image: if the distance is small or zero then the image corresponds likely to a pedestrian.

Figure 2.3: L1 distance Illustration



[Image from online source, [11].]

Suppose there are different labels that an image might correspond to in the training set- for example a proposed image might be pedestrian, car, traffic etc. The idea of the k-nearest neighbour would be to find the closest k images to a given test

image; and then vote to pick the correct one. When k=1, we have the nearest neighbour classifier.

In practice, it is required to Split the training set into training set and a validation set. The validation set is used to tune all hyperparameters. Hyperparameters are parameters that (for example the L1 and L2) influence the accuracy of our model during training. The test set is required to be a fresh set on which we can evaluate the model.

The nearest neighbour is relatively easy to train but not robust. However, it serves as a useful simple illustration. We now turn to Linear models.

2.2.2 Linear Models

For this section, we define the score function and the loss function. The *score function* defines a mapping from the raw data (of the 2D image) to class scores (where the class represents the possible set of objects that are present in the scene)[11]. The cost function defines the agreement between the predicted scores and the ground truth[11].

2.2.2.1 Linear Classifier Score Function

To each $x_i \in R^D$ of the training set acquired from our camera for example we associate a label y_i. Accordingly, $i = 1, 2 \ldots N$ represent the total number of images in the set (where each image has dimensionality D) and $y_i \in 1 \ldots K$ represents the distinct categories of objects (pedestrians,traffic light,houses etc).

The score function is then defined as

$$f : R^D \rightarrow R^K$$

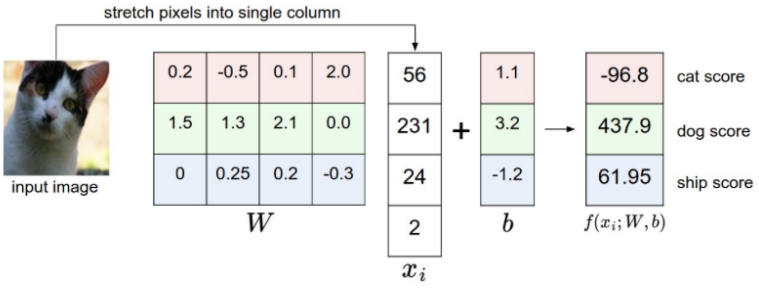
and maps the raw image pixels to class scores[11]. For the linear classifier, a simple yet useful score function is the linear mapping below :

$$f(x_i, W, b) = Wx_i + b$$

It is assumed that the image x_i has all of its pixels flattened out to a single column vector of shape [D x 1]. To see this in context see the image below:

The matrix W (of size [K x D]) represents the weight, and the vector b (of size [K x 1]) represent the bias.

Figure 2.4: linear score function illustrated



[Image from online source, [11]. Each row of W can be assumed to represent the necessary "weight values" needed to compute the score of a given class label. For any test image, we can compute the corresponding score all class labels. In the example, the classes are cat, ship and dog. Cat however has the highest score has expected]

2.2.2.2 Loss Function for Linear Classifier

Having computed the score for each class. It is appropriate to define a loss function which helps us evaluate the accuracy of the predictions based on the score. A high loss means the model is not doing well in making correct predictions and a low loss means the predictions of the model are good.

The Multiclass support vector Machine is a simple example of loss function that is widely used. This is given as[11]:

$$\boldsymbol{L_i} = \sum_{\boldsymbol{j \neq i}} \max(0, s_j - s_{y_i} - \Delta)$$

where s_j is the score of the class j and s_{y_i} is the score of the true class, Δ is an hyperparameter. To put this in context, take the output of the cat classification above:

$\boldsymbol{s} = [-96.8, 437.9, 61.95]$. Then to compute the SVM loss function assuming $\Delta = 20$:

$$L_i = max(0, -96 - 437.9 + 20) + max(0, 61.95 - 437.9 + 20)$$

$$L_i = max(0, -) + max(0, -)$$

In general, the SVM loss function wants to ensure the score of the correct class y_i larger than the incorrect class scores by at least by Δ. If the score is negative then it is thresholded to zero as in the formula shows. In the example above the loss value is zero.

It is also possible to add a regularisation parameter to the loss function computation. This is important because if W is the weight we acquired from training. Then any λW would also be a valid weight.Therefore, to encode some preference for a certain set of weights W over others to remove this ambiguity, the regularisation parameter is

needed. The regularisation parameter helps compute the regularisation loss [11] and is given below:

$$R(W) = \sum_{n} \sum_{m} W_{n,m}^2$$

Adding the SVM loss and regularisation loss (weighted by hyperparameter λ) we have the following objective function [11]:

$$L = \frac{1}{N} \sum_{i} L_i + \lambda R(W)$$

or in full as

$$L = \frac{1}{N} \sum_{i} \sum_{j \neq y_i} [\max(0, f(x_i; W, b)_j - f(x_i; W, b)_{y_i} + \Delta] + \lambda \sum_{m} \sum_{n} W_{n,m}^2$$

where N is the total number of training example and λ an hyperparameter.

In practice it turns out that Δ can be set to 1 (assuming we *normalize* the data set before training). Together both Δ and λ control tradeoff between the data loss and the regularization loss in the objective [11]. The magnitude of the weights has direct effect of the computed scores, therefore the Δ and λ help prevent the model to genralize (prevent *overfitting*, or say invariant to view point variation).

Another popular choice of loss function is the soft Max defined as:

$$L_i = -\log(\frac{\exp(f_{y_i})}{\sum_j \exp(f_j)})$$

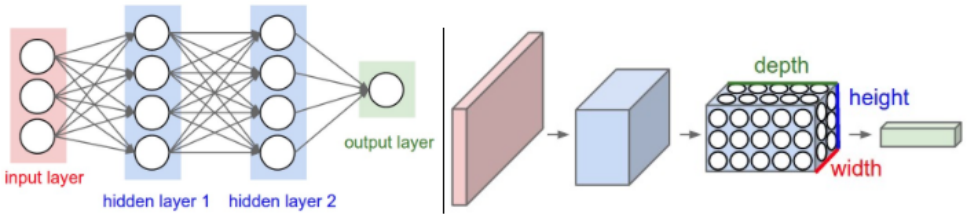
The ouput of the softmax loss function is a normalized class probabilities. This means it assigns probability value to each class, and the most likely class gets the highest probability value.

2.2.3 Convolutional Neural network

Convolutional networks or as convolutional neural networks (CNN or ConvNets for short), are a specialized kind of neural network used in image classification . A neural network typically has numerous weighted linear units (discussed earlier) endowed with an activation function as its computational unit and act on data fed into it from the

input node. But, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth [11]. As the name suggest CNN makes use of the usual convolution operation in place of the usual matrix operation of a typical neural net.

Figure 2.5: Neural net Versus Conv Net



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

[Image from, [11].]

2.2.4 Convolutional Neural Network

The convolution of x and w written as $x * w$ is defined as [12] :

$$s(t) = (x * w)(t) \tag{2.1}$$

$$= \int x(a)w(t-a)da \tag{2.2}$$

If we consider discrete convolution, the equations above becomes:

$$s(t) = (x * w)(t) \tag{2.3}$$

$$= \sum_{a=-\infty} x(a)w(t-a) \tag{2.4}$$

In ConvNets, x would typical be the input to the CNN and w the kernel or Filter. The output is usually called Feature Map. Since the input to the ConvNets are multidimensional array, 2D image for example, then we have to use 2D Kernel. Let I be the 2D image and K the kernel then the 2D convolution is defined as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-m)$$

Since, convolution operation is commutative, the equation above can be rewritten as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(i,j)$$

Most libraries implements cross correlation which is same as convolution but without flipping the the kernel [12]:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(i,j)$$

The figure below shows convolution on 2D tensor:

Figure 2.6: Illustration of Conv

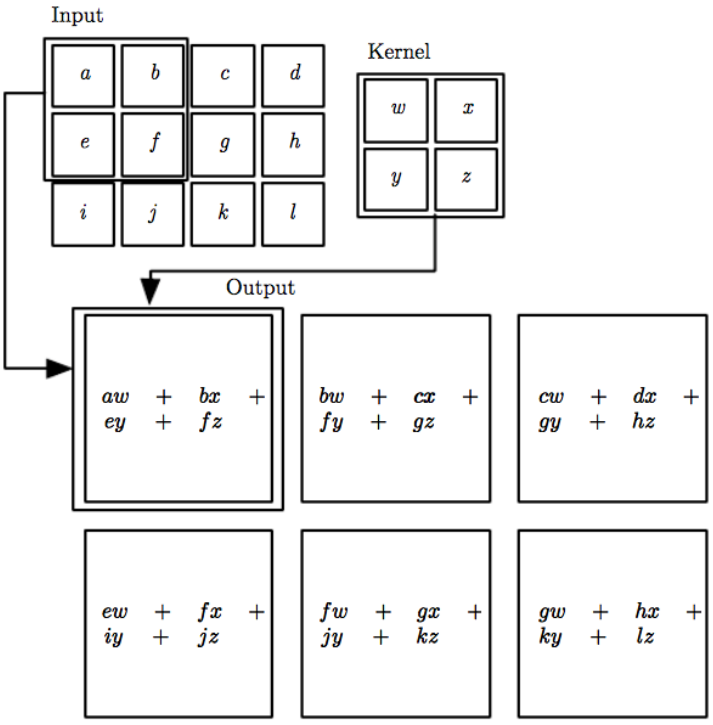


Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

[Illustration from the deep learning book by Ian Goodfellow, [12].]

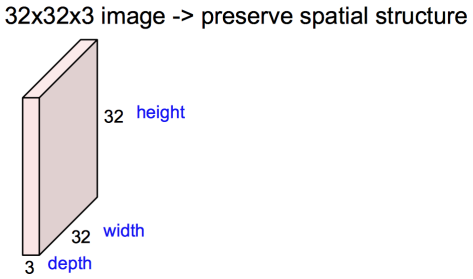
2.2.5 ConvNets Building Blocks

Three main types of layers are used to build ConvNet architectures: Convolutional Layer,Pooling Layer, and Fully-Connected Layer.

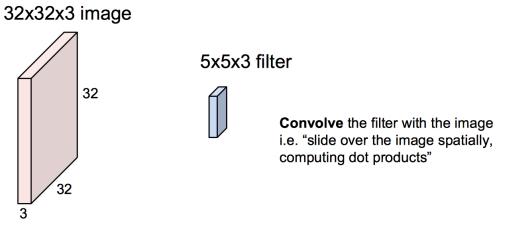
2.2.5.1 Convolutional Layer

The Conv layer is the main computational unit of the network. The CONV layers parameters consist of a set of learnable filters. Each filter is small spatially (along width and height), however it extends through the full depth of the input volume. **Local Connectivity:** Typically, it is impractical to connect neurons to all neurons in the previous volume. Each neuron is connected to only a *local region* of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume[11].

There is no way to visualise the convolution layer than with an example:

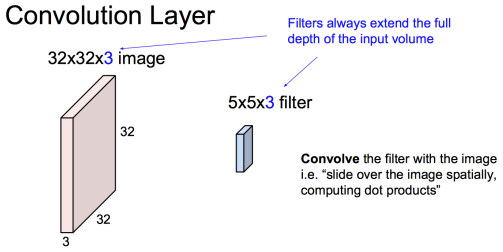


(a) Let's say this is the input to the network

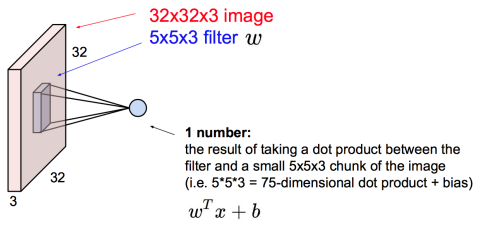


(b) The input and the Filter

[Image Source [11]]

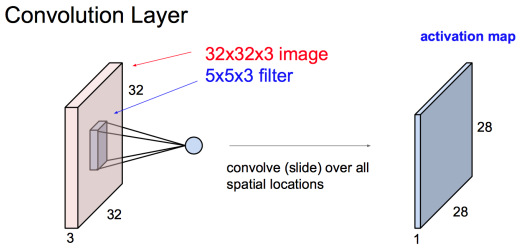


(c) Filter should extend the depth

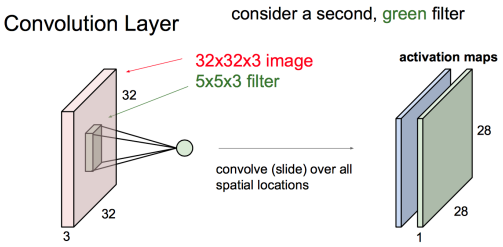


(d) The output: Note use of Linear Function locally, not on entire image

[Image Source [11]]

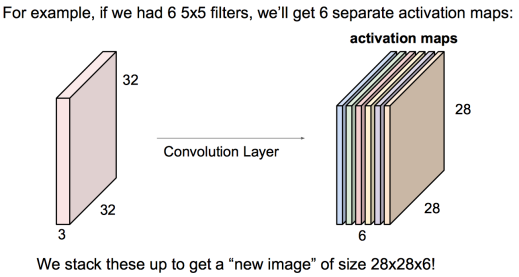


(e) After convolution over the entire image , the output is a Feature map or activation map

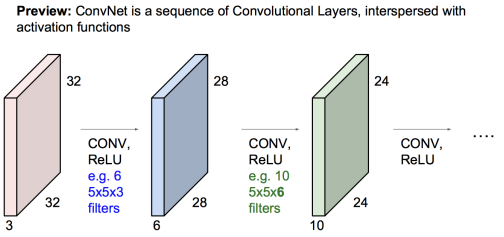


(f) Typically, more than one filter is used

[Image Source [11]]



(g) Each Activation Map correspond to a filter



(h) CNN is sequence of these layers with activation function (ReLu)

[Image Source [11]]

It is important to note that:

- The depth of the output volume is an hyperparameter and reflects the number of filter used in the network [11]
- The stride determines how we slide the filter[11]
- The input volume is often zero padded. This invovlves padding the input volume with zeros around the border.
- Parameter sharing refers to using the same parameter for more than one function in a model[12]. It is used in convnet to control the number of parameters[11]. In a convolutional neural net, each member of the kernel is used at every position of the input The parameter sharing used by the convolution operation means that rather than learning a separate set of parameter for every location, we learn only one set [12].

Example 2.1. *Consider that we have receptive of size* $55 \times 55 \times 96 = 290,400$ *(width,height and depth) in the first Conv Layer of a network, and each was connected to region of* $11 \times 11 \times 3 = 363$ *size in the input volume. Thus, this gives a total*

363 *weights and 1 bias. Together, this adds up to* $290400 \times (363+1) = 105,705,600$ *parameters on the first layer of the ConvNet alone! Using Parameter sharing and denoting a single 2-dimensional slice of depth as a* ***depth slice*** *(=96), we are going to constrain the neurons in each depth slice to use the same weights and bias. This is equivalent to all* 55×55 *neurons in each depth slice*(96) *using the same parameters - 96 unique set of weights (one for each depth slice). This gives* $96 \times 11 \times 11 \times 3 = 34,848$ *unique weights, or 34,944 parameters (+96 biases).*

- RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero.

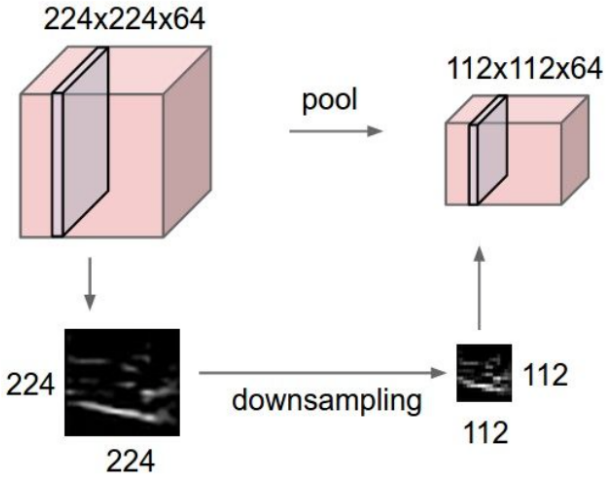
Conv Layer Summary [11]

- Input: $W_1 \times H_1 \times D_1$ as input volume.
- Parameters required:
  - Number of Filters, **K**
  - spatial extent, **F** (Filter size)
  - Stride, **S**
  - Amount of Zero padding, **P**
- Output Volume: $W_2 \times H_2 \times D_2$ where:
  - $\mathbf{W_2} = \frac{\mathbf{W_1 - F + 2P}}{\mathbf{S+1}}$
  - $\mathbf{H_2} = \frac{\mathbf{(H_1 - F + P)}}{\mathbf{S+1}}$
  - $\mathbf{D_2} = \mathbf{K}$
- $\mathbf{F} \times \mathbf{F}$ weights per filter. Total of $(\mathbf{F} \times \mathbf{F} \times \mathbf{D}).\mathbf{K}$ weights and **K** biases.

2.2.5.2 Pooling Layer

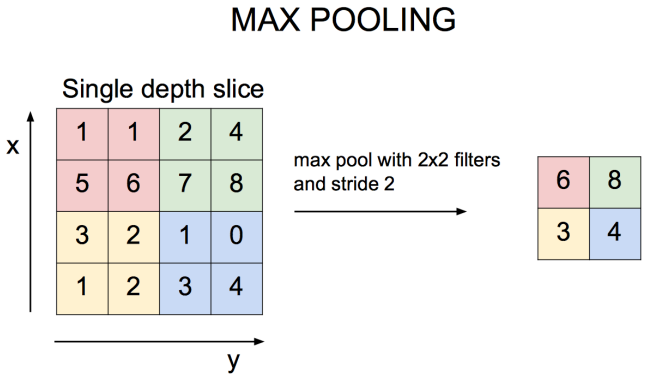
Its function is to progressively reduce the spatial size of the representation thereby reducing the amount of parameters and computation in the network- hence reduce over-fitting. It operates on every depth slice of the input and resizes it spatially, using the MAX operation.

Figure 2.7: Pooling as Downsampling



[Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume [11].]

Figure 2.8: Max Pool using 2×2 filter



[The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2×2 square). Source, [11].]

Pooling Layer Summary[11]

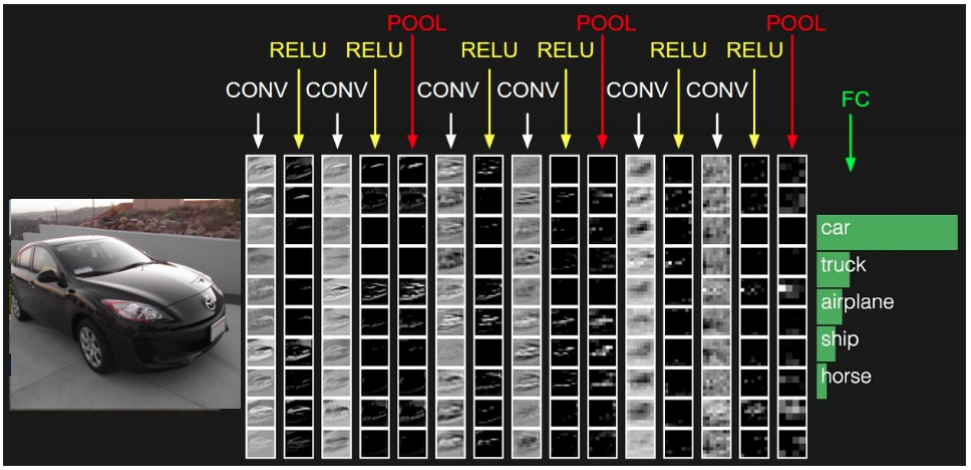
- Input: $W_1 \times H_1 \times D_1$ as input volume.
- Parameters required:
  - spatial extent, **F** (Filter size)
  - Stride, **S**
- Output Volume: $W_2 \times H_2 \times D_2$ where:
  - $\mathbf{W_2} = \frac{\mathbf{W_1 - F}}{\mathbf{S+1}}$
  - $\mathbf{H_2} = \frac{\mathbf{(H_1 - F)}}{\mathbf{S+1}}$
  - $\mathbf{D_2} = \mathbf{D_1}$
- Introduces zero parameters. Usually no zero padding.

2.2.5.3 Fully Connected Layer

Contains neurons that connect to the entire input volume, as in ordinary Neural Networks [11]. It is in this Layer that the class scores for the network is computed.

In general a typical CNN archtitecture pipeline would involve **(INPUT - CONV LAYER - RELU - POOL - FULLY CONNECTED LAYER)**[11]. This is illustrated in the figure below:

Figure 2.9: Typical CNN Architecture pipeline



[Taken from [11].]

2.3 Kitti Data Set

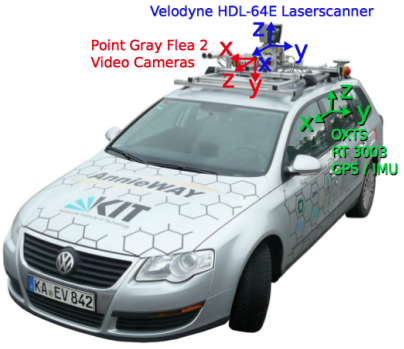
Since the project takes a data driven approach, it is important to provide summary facts about the data sets used in the project. This section does that and also relevant summaries about the Kitti data set.

The Kitti dataset was captured from a VW station wagon moving around Karlsruhe (Germany) and has been used extensively in mobile robotics and autonomous driving research. In total, it consist 6 hours of diverse real driving scenerio recorded at 10-100Hz using a veriety of sensor highlightted. The sensor setup is listed below:

- 2 × PointGray Flea2 grayscale cameras (FL2-14S3M-C), 1.4 Megapixels,1/2″ Sony ICX267 CCD, global shutter
- 2 × PointGray Flea2 color cameras (FL2-14S3C-C), 1.4 . Megapixels, 1/2″ Sony ICX267 CCD, global shutter.

- 4 × Edmund Optics lenses, 4mm, opening angle 90° , vertical opening angle of region of interest (ROI) 35°
- 1 × Velodyne HDL-64E rotating 3D laser scanner, 10 Hz, 64 beams, 0.09 degree angular resolution, 2cm distance accuracy, collecting 1.3 million points/second, field of view: 360° horizontal, 26.8° vertical, range: 120 m.

Figure 2.10: Kitti Dataset Platform

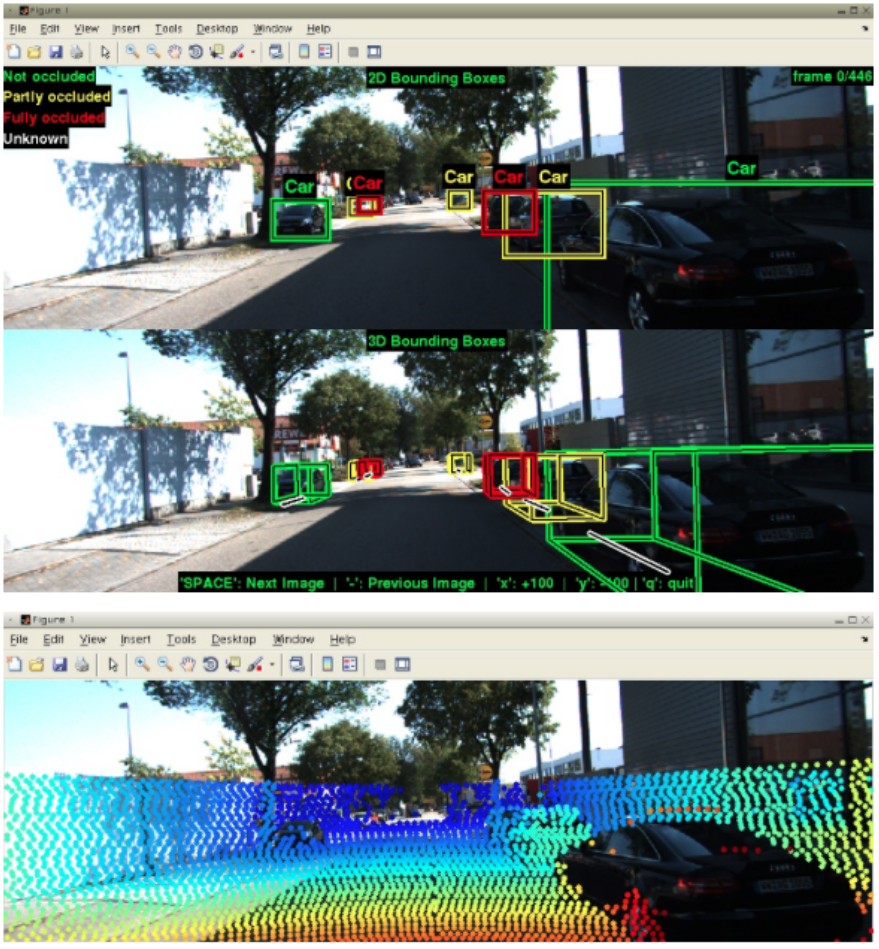


[VW Passat station wagon equipped with four video cameras (two color and two grayscale cameras), a rotating 3D laser scanner and a combined GPS/IMU inertial navigation system, source [8].]

The kitti raw data set is divided into categories: Road, City, Residential, Campus and Person. This project is focused on the Person category. For each category, both gray and color images are stored with loss-less compression using 8-bit PNG files. The engine hood and sky region have been cropped from the images.

The Velodyne scans are stored as floating point binaries. Each point is stored with its (x, y, z) coordinate and an additional reflectance value (r).Per frame sensor readings and corresponding timestamps are provided for each category.

Figure 2.11: image and point cloud from Kitti Data set

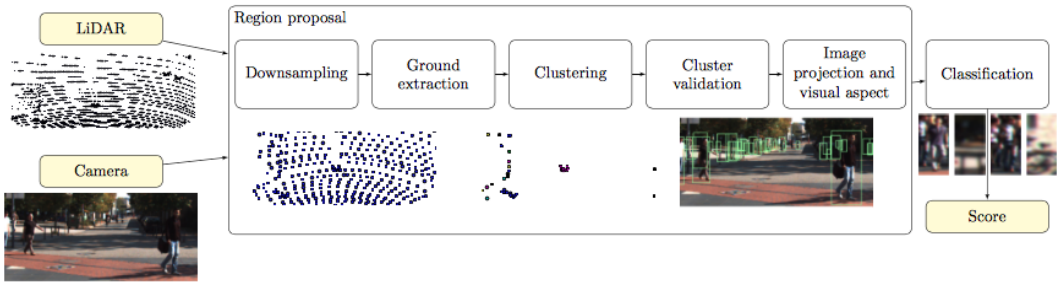


[Image and bounding box (top) Velodyne point clouds (bottom) and their projections onto the image plane. Source [8].]

2.4 Pedestrian detection framework

Combining this point cloud and camera images, The Pedestrian detection framework pipeline proposed in this project is depicted below:

Figure 2.12: Pedestrian Detection Framework



[Pedestrian Detection Framework pipeline proposed in this project. Source from LTS4 Lab EPFL.]

2.5 Conclusion

This chapter lays the foundation of image recognition. In addition the typical CNN architecture was discussed. In the next chapter we build on this and try to utilise 3D point cloud to reduce uncertainty in the prediction.

Chapter 3

Unknown

3.1 Unknown

Chapter 4

Unknown

4.1 Unknown

Chapter 5

Unknown

Appendix A

Appendix : Gentle Introduction to CNN

Appendix B

Activation Functions

In this project and especially in our discussion of LSTM we made rference to activation function. The activation function of a node defines the output of that node given an input or set of inputs. As an analogy, a standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on the input.

Figure B.1: Various Activation Functions

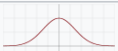
Name	Plot	Equation	Derivative (with respect to x)	Range	Order of continuity	Monotonic	Derivative Monotonic
Adaptive piecewise linear (APL) [15]		$f(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$	$f'(x) = H(x) - \sum_{s=1}^{S} a_i^s H(-x + b_i^s)$ [2]	$(-\infty, \infty)$	C^0	No	No
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞	Yes	No
Bent identity		$f(x) = \frac{\sqrt{x^2+1}-1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2+1}} + 1$	$(-\infty, \infty)$	C^∞	Yes	Yes
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}	Yes	No
Exponential linear unit (ELU)[13]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\alpha, \infty)$	C^1 when $\alpha = 1$, otherwise C^0	Yes iff $\alpha \geq 0$	Yes iff $0 \leq \alpha \leq 1$
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$	$(0, 1]$	C^∞	No	No
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	C^∞	Yes	Yes
Leaky rectified linear unit (Leaky ReLU)[10]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	C^∞	Yes	No
Parameteric rectified linear unit (PReLU)[11]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes iff $\alpha \geq 0$	Yes

Image from `https://en.wikipedia.org/wiki/Activation_function`

Bibliography

[1] Rise and Shine. Motivational talk. https://www.youtube.com/watch?v=SuPLxQD4akQ, 2012. Accessed Dec 5, 2017.

[2] Pedro J. Navarro, Carlos Fernndez, Ral Borraz, and Diego Alonso. A machine learning approach to pedestrian detection for autonomous vehicles using high-definition 3d range data. *Sensors*, 17(1), 2017. ISSN 1424-8220. doi: 10.3390/s17010018. URL http://www.mdpi.com/1424-8220/17/1/18.

[3] Martin Weinmann. *Reconstruction and Analysis of 3D Scenes: From Irregularly Distributed 3D Points to Object Classes*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319292447, 9783319292441.

[4] Wikipedia. Autonomous car — wikipedia, the free encyclopedia, 2017. URL https://en.wikipedia.org/w/index.php?title=Autonomous_car&oldid=796221467. [Online; accessed 19-August-2017].

[5] Hong Cheng. *Autonomous Intelligent Vehicles - Theory, Algorithms, and Implementation*. Advances in Computer Vision and Pattern Recognition. Springer, 2011. ISBN 978-1-447-12279-1. doi: 10.1007/978-1-4471-2280-7. URL https://doi.org/10.1007/978-1-4471-2280-7.

[6] Emanuele Trucco , Alessandri Verri. *Introductory techniques for 3D vision*. Prentice-Hall, New Jersey, April 1998.

[7] Simon J. D. Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, 2012. doi: 10.1017/CBO9780511996504.

[8] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[9] Jamie Cartar et Al. Lidar 101: An introduction to lidar technology, data, and applications. https://coast.noaa.gov/data/digitalcoast/pdf/lidar-101.pdf, Nov 2012. [Online; accessed on 27-Aug-2017].

[10] PortLand State University. Lidar. `http://web.pdx.edu/~jduh/courses/geog493f12/Week04.pdf`, Nov 2012. [Online; accessed on 27-Aug-2017].

[11] Convolutional neural net for vision recognition, April 2017. URL `http://cs231n.github.io/classification/`. [Online: accessed on 22 August 2017, CS231N, Stanford University].

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.