Multi Scale Object Detection Using Convolutional Neural Net

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

September 2018

“ It always seem impossible until it is done”

Nelson Mandela

Abstract

In autonomous vehicle or robots design, the environmental perception and modeling module is responsible for making meaning of the object in the surroundings. This has two inherent challenges: first it is important to identify when an object is present in an image captured by sensors and second is to accurately classify the object into appropriate object class. Added to this, is the problem of classifying multiple objects of different scales and classes captured in a given frame.

Thus, there are a lot of complexities to deal with: both from the algorithmic and modelling perspective. An algorithm that takes too long to converge optimally might never be deployed and a model too clumsy to understand might never be accepted.

The aim of this project is to develop and test algorithmic models that are able o detect object at different scales. This will be achieved by leveraging the effectiveness of convolutional neural nets (conv net). In particular, it is desired to know, how the depth of the conv net affect performance for multi-scaled object detection. This is important so as to be able to obtain which model performs best for a particular object scale. The result of this analysis can be exploited in ensemble model for object detection.

Acknowledgements

Many people have helped shape and train my ideas over the years. Verily, writing all your names out here would fill an entire book in itself. Thus I hope you will all forgive me if I don't: but as always be assured that I love, cherish and admire you all. You have all given me strong motivation and constructive criticism with which my life has thrived.

In particular, I will like to specially 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 working on this project.

Contents

Abstract ii

Acknowledgements iii

List of Figures vi

List of Tables viii

Abbreviations ix

1 Background 1
- 1.1 Introduction . . . 1
- 1.2 The Big picture . . . 2
  - 1.2.1 Autonomous Robots or Vehicles: A General Overview . . . 2
    - 1.2.1.1 Multi-sensor Based Environment perception and Modeling . . . 3
    - 1.2.1.2 Basis Element of Image Device . . . 4
    - 1.2.1.3 Simple Model for Perception: The Pin Hole Camera . . . 5
    - 1.2.1.4 Mathematical Model Of Pin Hole Camera . . . 5
- 1.3 The Kitti Data set . . . 8
- 1.4 What is an Object? . . . 9
- 1.5 Object classification and Object detection . . . 9
  - 1.5.1 Remarks . . . 10
- 1.6 Conclusion . . . 10

2 Image Classification 12
- 2.1 Introduction . . . 12
  - 2.1.1 Simple Image Classification Model . . . 13
- 2.2 Learning Models . . . 14
  - 2.2.1 Nearest Neighbour and K-Nearest Neighbour . . . 14
  - 2.2.2 Linear Models . . . 15
    - 2.2.2.1 Linear Classifier Score Function . . . 15
    - 2.2.2.2 Loss Function for Linear Classifier . . . 15
  - 2.2.3 Convolutional Neural network . . . 17
  - 2.2.4 Convolutional Neural Network . . . 18

2.2.5 ConvNets Building Blocks . . . 19
2.2.5.1 Convolutional Layer . . . 20
2.2.5.2 Pooling Layer . . . 22
2.2.5.3 Fully Connected Layer . . . 24
2.2.5.4 Fully Connected Layer . . . 24
2.3 Conclusion . . . 25

3 Object Detection **26**
3.1 Introduction . . . 26
3.2 Object Localization And Detection . . . 26
3.2.1 Object Localization Problem . . . 27
3.2.1.1 Simple Illustration . . . 29
3.2.2 Sliding Window Algorithm: Sequential Execution . . . 31
3.3 Convolutional Sliding Window: One Forward Pass Execution . . . 32
3.3.1 From Fully Connected Layer to Convolutional Layer . . . 33
3.3.1.1 Evalualting Object Localization: Intersection Over Union 35
3.3.1.2 Non Maximum Suppression . . . 36
3.3.1.3 Anchor boxes . . . 37
3.4 Conclusion . . . 38

4 Theoretical Model Analysis **39**
4.1 Introduction . . . 39
4.1.1 Supervised learning and rate distortion theory . . . 39
4.1.1.1 Rate Distortion for encoder and decoder . . . 40
4.1.1.2 Bayesian learning perspective . . . 41
4.2 Opening the Neural net Black Box . . . 43
4.2.1 Information Bottleneck (IB) . . . 43
4.2.2 Stochastic Gradient Descent and Information Bottleneck . . . 45
4.3 Conclusion . . . 47

5 Experimental Results **48**
5.1 Optimization Technique Used in Implementation . . . 48
5.1.1 Dropout And learning rate . . . 49
5.1.2 Data Augmentation . . . 50
5.1.3 Weight Initialization . . . 50
5.1.4 Mini Batch Stochastic gradient . . . 50
5.1.5 Batch Normalization . . . 51
5.2 Model Results . . . 51
5.2.1 First Model Results . . . 51
5.2.1.1 Single Conv 1 Model Architecture . . . 54
5.2.2 Second Model Results . . . 56

A Appendix : Gentle Introduction to CNN **57**

Bibliography **58**

List of Figures

1.1 framework for autonomous vehicle . . . 2
1.2 Environmental Perception and modelling . . . 4
1.3 Composition of a Simple Vision Device . . . 5
1.4 Kitti Dataset Platform . . . 8
1.5 image and point cloud from Kitti Data set . . . 9
1.6 Object Localization. . . . 10

2.1 Simple Cat Classification . . . 13
2.2 L1 distance Illustration . . . 14
2.3 linear score function illustrated . . . 16
2.4 Neural net Versus Conv Net . . . 18
2.5 Illustration of Conv . . . 19
2.6 Pooling as Downsampling . . . 23
2.7 Max Pool using 2×2 filter . . . 23
2.8 Typical conv net Architecture pipeline . . . 24
2.9 Components, techniques and application of conv net . . . 24

3.1 image of cat . . . 27
3.2 Localization problem . . . 27
3.3 Detection problem . . . 27
3.4 Classification problem . . . 28
3.5 Add localization module . . . 28
3.6 Bounding box parameters . . . 29
3.7 Input : Object class car . . . 30
3.8 Input : Object class background . . . 30
3.9 Closely cropped image for use in sliding window algorithm . . . 31
3.10 Sliding window technique Illustration . . . 32
3.11 Conversion of Fully connected layer to Full convolutional layer . . . 33
3.12 Implementation of sliding window algorithm with conv net . . . 34
3.13 Image shift positions in sequential execution Model . . . 34
3.14 Window position superimposed: Optimal execution . . . 35
3.15 Intersection of union . . . 35
3.16 Non maximum suppression . . . 36
3.17 Default Bounding Box Anchors . . . 37
3.18 Default Bounding Box After non maximum suppression . . . 37

4.1 rate distortion encoder and decoder versus supervised learning . . . 40
4.2 Information curve for Deep Neural net . . . 44

4.3 Markov View of DNN layers 45
4.4 Graph of Stochastic gradient descent phases across epoch 46

5.1 Input 49
5.2 Model Finetuning 50
5.3 Network Architecture 51
5.4 Cross entropy loss for small pedestrians 52
5.5 Cross entropy loss for Big pedestrians 52
5.6 Cross entropy loss for small pedestrians 53
5.7 Cross entropy loss for cyclists 53
5.8 Single Conv architecture 54
5.9 Training Loss plot for New Conv 1 Architecture 55
5.10 New Conv1 versus Conv4: Training loss 56
5.11 New Conv1 vs Full network: Training loss 56

List of Tables

Abbreviations

Acronym	**W**hat (it) **S**tands **F**or.
IoU	**I**ntersection **O**ver **U**nion
LIDAR	**L**ight **D**etection **A**nd **R**anging
Conv Net	**C**oncolutional **N**eural **N**etwork
RPN	**R**egion **P**roposal **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
HOG	**H**istogram of **O**riented **G**radient
FC	**F**ully **C**onnected Layer
SIFT	**S**cale **I**nvariant **F**eature **T**ransform

Dedicated to my ***mum*** *(who passed away during my Masters studies)*

and

Mr. Ashade (for his constant mentoring)

Chapter 1

Background

1.1 Introduction

In order to be able to transfer specific capabilities of human visual perception to autonomous vehicle or robots , 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 [1]. 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: object detection.

This chapter starts with a discussion of how object detection fits into the autonomous system design architecture. Next, a simple perception model - using simple the pin hole camera - is described to provide a general abstraction of what a "hypothetical" image capture or sensor device should do. From this simple abstraction, we can infer what parameters or problems are inherent for such devices. The remainder of the chapter then shifts to the main task of this project : the detection of objects captured by such devices - since the images have already been captured (by these devices) and were acquired online from kitti Vision Bench mark suite `http://www.cvlibs.net/datasets/kitti/`.

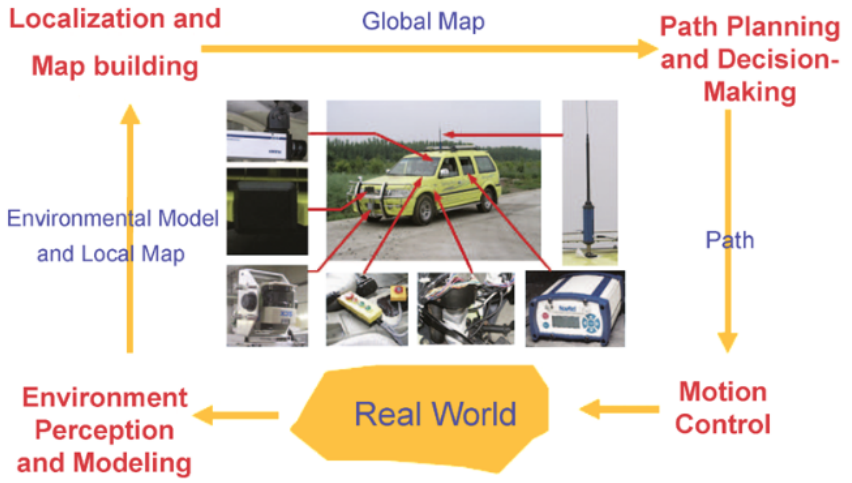
1.2 The Big picture

1.2.1 Autonomous Robots or Vehicles: A General Overview

Autonomous vehicle [1] is a vehicle that is capable of sensing its environment and navigating without human input [2]. It basically consists of four fundamental technologies (Fig 1.1):

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

Figure 1.1: framework for autonomous vehicle



[Image Source [3]]

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[3]. 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.

[1] We assume autonomous vehicle is an instance of autonomous robot. So that the concepts discussed here can be extended to any generalized autonomous robots

The goal of the second module, localization and map building, is to use geometric feature location estimate in the map to determine the vehicles (or robott's) 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 [3].

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 [3].

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[3].

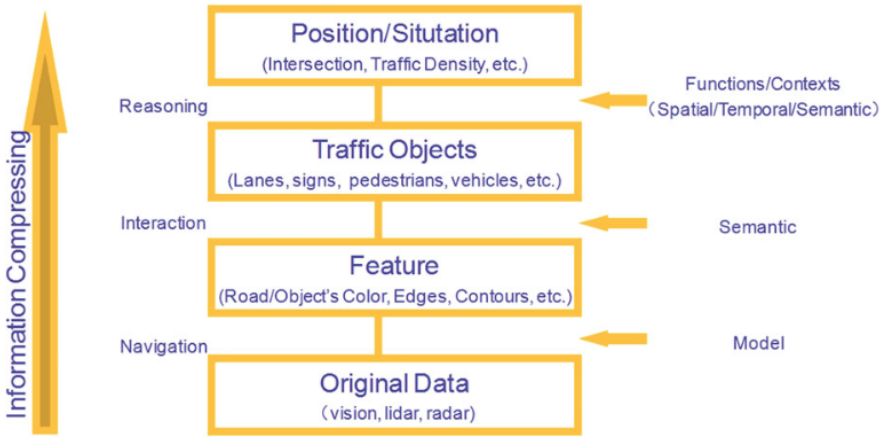
This project will focus on the Environmental perception module and modelling aspect.

1.2.1.1 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

Figure 1.2: Environmental Perception and modelling



[Illustration of basic component of imaging device. Source [3]]

- Multi-sensor Fusion

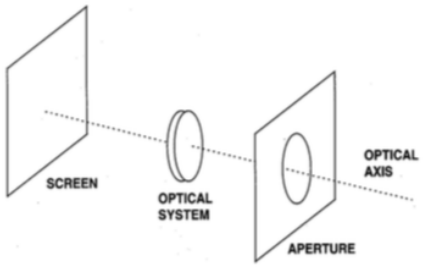
  Multi-sensor Fusion is the basic framework of intelligent vehicles for better sensing surrounding environment structures, and detecting objects/obstacles [3]. Sensors for detecting surrounding environment perception can either be active or passive. Active sensors include Lidar, Radar, ultrasonic and 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 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. This project focuses on developing algorithmic models for detecting such objects.

1.2.1.2 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 [4].

Figure 1.3: Composition of a Simple Vision Device



[Illustration of basic component of imaging device. Source [4]]

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 [5]:

- 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.

1.2.1.3 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 [4]. *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[4].

1.2.1.4 Mathematical Model Of Pin Hole Camera

The Figure above (fig 1.3) 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 [5]. 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 [5]. 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* [5]. 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 [5]. 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. From the analysis above, we identify the 3 fundamental geometric[5] problems:

- 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)

This project won't discuss further the first two problems- since the data set we used, the Kitti data set has taken care of this. [6] - see next subsection.

The last problem is being studied in a parallel project to this one. It is hoped that data from point cloud can processed or trained to provide cue for the model described in this project. This will definitely lead to increase in accuracy of the model described in this project.

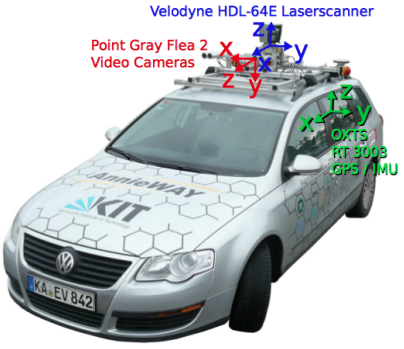
1.3 The Kitti Data set

The section introduces 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 1.4: Kitti Dataset Platform

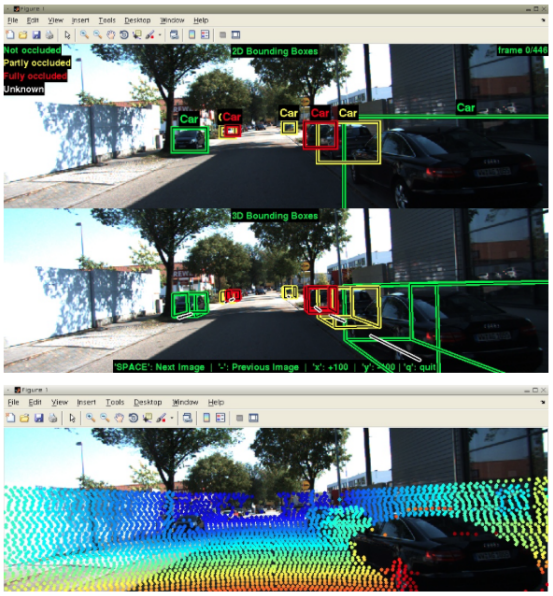


[VW Passat station wagon equipped with four video cameras (two color and two grayscale cameras), a rotating 3D laser scanner and a combined GPS/IMU inertial navigation system, source [6].]

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 1.5: image and point cloud from Kitti Data set



[Image Source [6].]

1.4 What is an Object?

In this project, objects are stand-alone things with well-defined regions and centre[7] (such as cars and bicycles) as opposed to amorphous background stuff such as sky , grass , and ground.

We desire that an object:

- has well-defined closed boundary in space[7]
- sometimes it is unique within the image and stands out as salient.[7]
- Object forming background will be labeled as neutral and won't correspond to any object class

1.5 Object classification and Object detection

The goal in this project is to train a model that performs Object detection. Objection detection is quite more challenging [8] than object classification since it requires more complex methods[8] to solve.

Complexity arises because detection requires localization of each object in a frame and subsequent categorization or classification of such object. Localization itself is has a little overhead::

- First, there are numerous candidate locations - set of all proposed window positions for probable object location [8] called proposals must be pre-processed[8]
- Second, these candidates proposals provide rough localization that must be further refined to provide more accurate localization. [8]

Figure 1.6: Object Localization.

[Usually green rectangle represent the real object location (ground truth). The red/blue rectangle represent the proposed locations. The algorithm should learn the objectness of the red rectangle window (given he ground truth) using metrics such as Intersection of union. This is followed by classification of the object. Image Source [7].]

1.5.1 Remarks

In general, it is important to measure and score how a window cover each object in a frame then develop model to classify each of them. This is what makes image recognition different from object detection. In particular, this means that it is required to measure or score the "objectness of a window". The objective of this objectness score is to localize objects in a window before classification.

It is desired that high score should be assigned to windows fitting an object tight - and low score otherwise. How this objectness and classification is done is the central aim of this project.

1.6 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 image capture is been simply

realized. In particular, we assume the images of the objects have been acquired and stored publicly online `http://www.cvlibs.net/datasets/kitti/`. Our aim is to develop algorithm that can help us detect object and classify it.The next chapter discusses how we carry out image classification .

Chapter 2

Image Classification

2.1 Introduction

This chapter discusses the intricacies of image classification. It is assumed that the data set are available and what is left is to develop a model or algorithm to do the classification. For simplicity we assume for the moment that there is one object class per image frame. Though there might be multiple such objects to classify. All that is required is that each object class is located in separate frame.

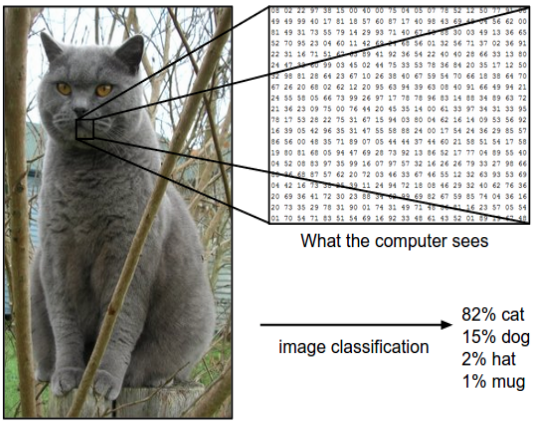
An obvious approach would be to write an algorithm that recognizes each distinct instant class of object. Such algorithmic representation would be required to be invariant to viewpoint variation, scale variation, deformation, occlusion, illumination conditions, background clutter, intra-class variations [9]. 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 same object class in different scenario [9].

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 objects in the scenes. This approach is the data-driven approach, since it relies on first accumulating a training data-set of labeled object images from the sensors. This makes it obvious the reason a standard vision benchmark suite such as kitti[6] is needed.

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 those "dumb numbers" for decision making: object classification.

Figure 2.1: Simple Cat Classification



[Image from Stanford Lecture note on: Convolution neural network for Visual recognition [9].]

A simplified pipeline for object classification using data diven apptoach is outlined 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*[9].
- 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 [9]).

2.2 Learning Models

2.2.1 Nearest Neighbour and K-Nearest Neighbour

As usual, it is good with a simplified model: the nearest neighbor classifier. The nearest neighbor classifier takes a test scene or image, compare it to every single one of the training set, and try predict if belongs to an object class or not. The aim would be to learn the pixel values that encapsulates the object 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 , [9], the L1 distance is computed as so:

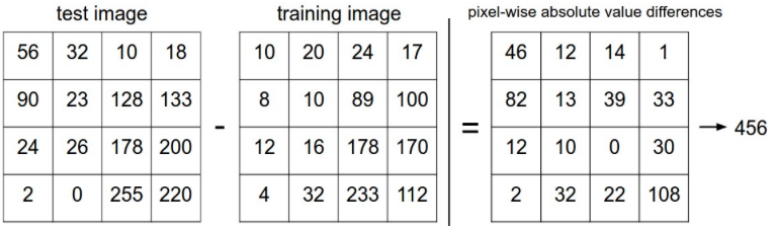
$$d_1(I_1, I_2) = \Sigma_p |I_1^p - I_2^p|$$

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 2.2: L1 distance Illustration



[Image from online source, [9].]

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 to be classified)[9]. The cost function defines the agreement between the predicted scores and the ground truth[9].

2.2.2.1 Linear Classifier Score Function

To each $x_i \in R^D$ of the training set 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[9]. For the linear classifier, a simple yet useful score function is the linear mapping below :

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

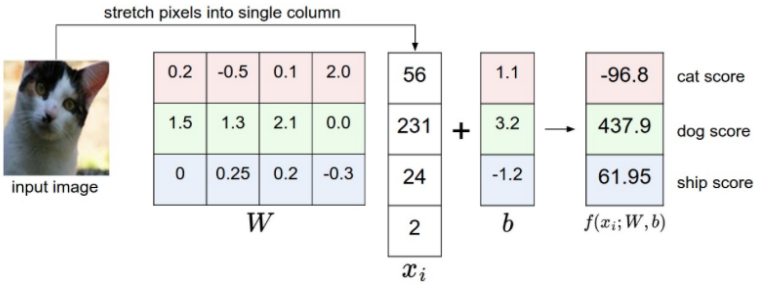
It is assumed that the image x_i has all of its pixels flattened out to a single column vector of shape [D x 1]. To see this in context see the image below:

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

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

Figure 2.3: linear score function illustrated



[Image from online source, [9]. 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]

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[9]:

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

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 [9] 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 [9]:

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

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 [9]. 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.

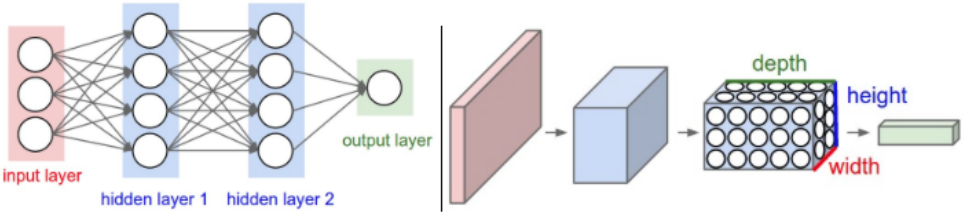
2.2.3 Convolutional Neural network

Conv net is deep-learning architecture which takes its inspiration from the perception mechanism of living organisms. It has recently been very effective in visual pattern pattern recognition and representation.

Like typical neural network it has multiple weighted linear units (discussed earlier) endowed with an activation function which together act on data fed into it from the input node and is usually trained using the back-propagation[10] algorithm. But,

unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth [9]. 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 2.4: 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, [9].]

2.2.4 Convolutional Neural Network

The convolution of x and w written as $x * w$ is defined as [11] :

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

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

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 [11]:

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

The figure below shows convolution on 2D tensor:

Figure 2.5: Illustration of Conv

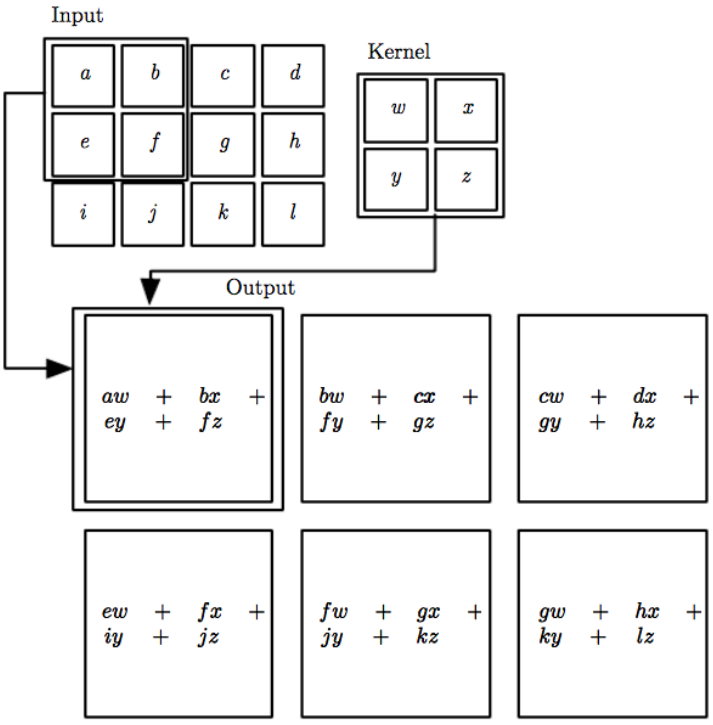


Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor 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, [11].]

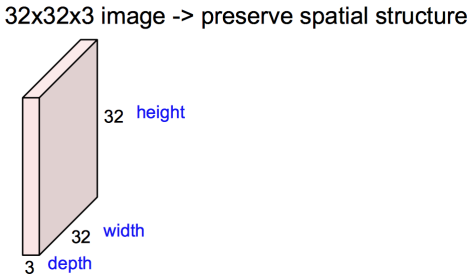
2.2.5 ConvNets Building Blocks

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

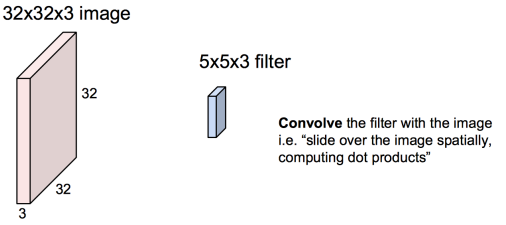
2.2.5.1 Convolutional Layer

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[9].

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

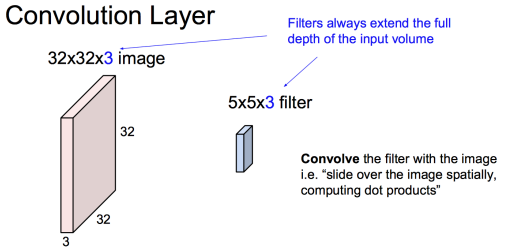


(a) Let's say this is the input to the network

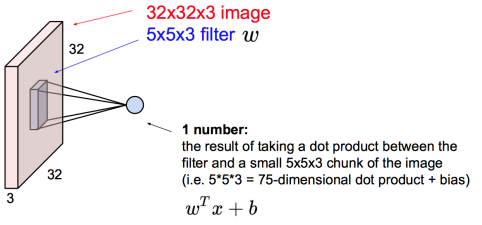


(b) The input and the Filter

[Image Source [9]]

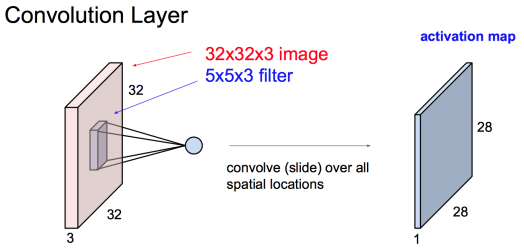


(c) Filter should extend the depth

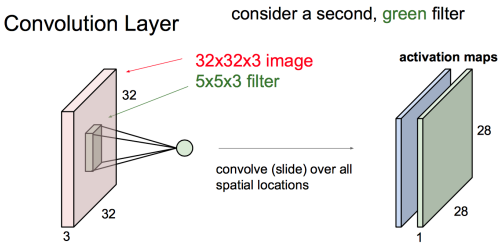


(d) The output: Note use of Linear Function locally, not on entire image

[Image Source [9]]

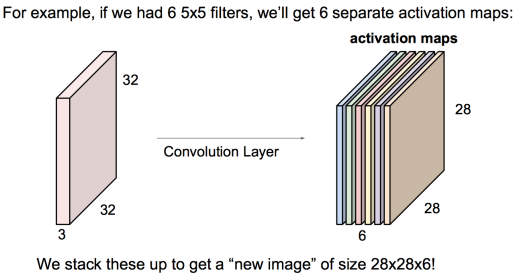


(e) After convolution over the entire image , the output is a Feature map or activation map

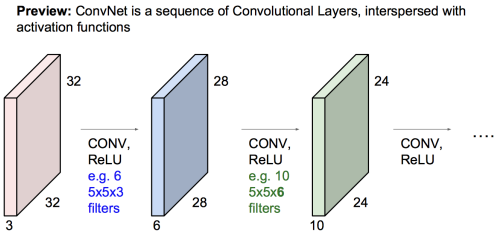


(f) Typically, more than one filter is used

[Image Source [9]]



(g) Each Activation Map correspond to a filter



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

[Image Source [9]]

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 [9]
- The stride determines how we slide the filter[9]
- 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[11]. It is used in convnet to control the number of parameters[9]. 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 [11].

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).[11]*

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

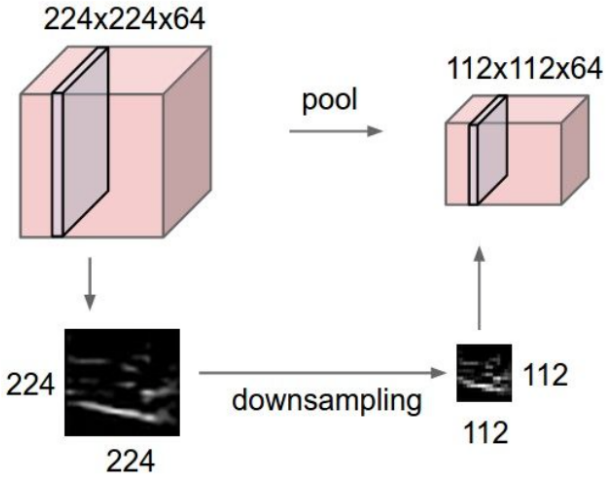
Conv Layer Summary [9]

- 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

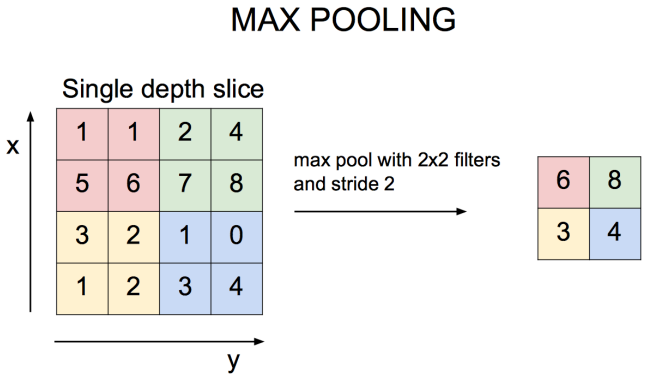
Its function is to progressively reduce the spatial size of the representation thereby reducing the amount of parameters and computation in the network- hence reduce overfitting. It operates on every depth slice of the input and resizes it spatially, using the "max" operation. Thus is illustrated below:

Figure 2.6: Pooling as Downsampling



[Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume [9].]

Figure 2.7: 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, [9].]

Pooling Layer Summary[9]

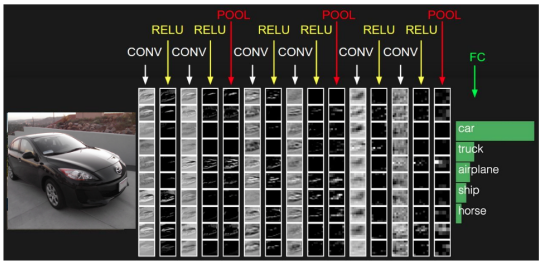
- Input: $W_1 \times H_1 \times D_1$ as input volume.
- Parameters required:
  - spatial extent, $\mathbf{F}$ (Filter size)
  - Stride, $\mathbf{S}$
- Output Volume: $W_2 \times H_2 \times D_2$ where:
  - $\mathbf{W_2} = \frac{\mathbf{W_1 - F}}{\mathbf{S+1}}$
  - $\mathbf{H_2} = \frac{\mathbf{(H_1 - F)}}{\mathbf{S+1}}$
  - $\mathbf{D_2} = \mathbf{D_1}$
- 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 [9]. It is in this Layer that the class scores for the network is computed.

In general a typical CNN architecture pipeline would involve *(input - conv layer - relu - pool - fully connected layer)*[9]. This is illustrated in the figure below:

Figure 2.8: Typical conv net Architecture pipeline

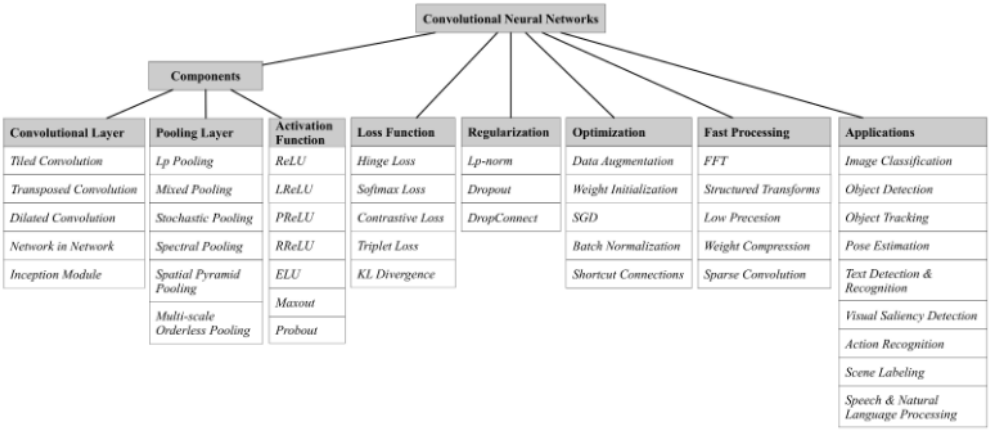


[Image source [9].]

2.2.5.4 Fully Connected Layer

The figure below shows the possible components, techniques and application of conv net:

Figure 2.9: Components, techniques and application of conv net



[Image source [9].]

2.3 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 utilize 3D point cloud to reduce uncertainty in the prediction.

Chapter 3

Object Detection

3.1 Introduction

In the last chapter , image classification was extensively discussed. This chapter extends the last chapter further to include image detection. This is a domain of computer vision that has been around in the last few decades. Previously, progress in object detection was based exclusively on the use of HOG (Histogram of gradient) and SIFT (scale Invariant Feature Transform) algorithms. However, in recent terms CNN have shown to outperform these other methods[12]. The task ahead then becomes how to adapt CNN for object detection task. Image detection, differs from image detection because it involves object localization.

This chapter starts with a brief discussion on object localization. Furthermore, the technique and metrics for object localization and detection in CNN will be analyzed.[1]

3.2 Object Localization And Detection

In the last chapter, image classification problem was discussed. This can be formulated like so: given the image, the algorithm (CNN) identifies the object (for example cat in the image below):

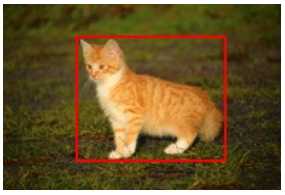
[1]This chapter is based on the Deep learning specialization course I completed on Coursera [13] taught by Prof Andrew Ng

Figure 3.1: image of cat

[Image source [14].]

In the localization problem not only is the algorithm to label or identify the object but it has also to put a bounding box around the position of the object:

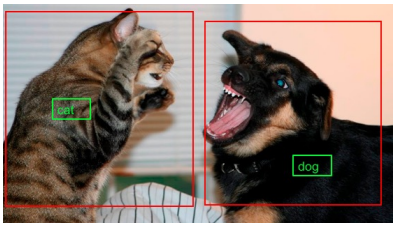
Figure 3.2: Localization problem



[Image source [14].]

In the detection problem, there might be multiple of such objects of different classes at multi-scales in the image, and the algorithm has to localize and label them:

Figure 3.3: Detection problem



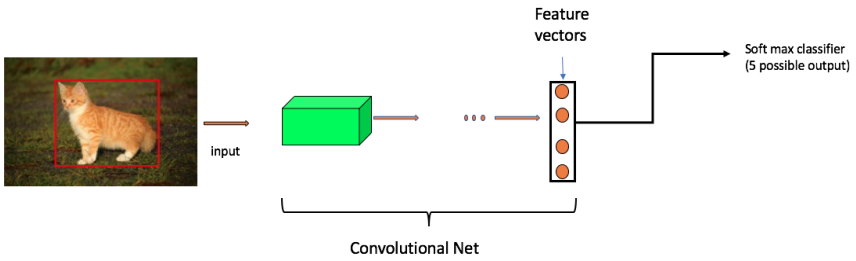
[Image source [15].]

In the detection problem , multiple object of different classes and at different depth can be in the image. In the localization (and by extension classification problem), there is usually single object located at the centre of the image. This is a constraint on the object class in an image and not on the number of object classes in the training set.

3.2.1 Object Localization Problem

The figure below shows the pipeline for classical object identification.

Figure 3.4: Classification problem



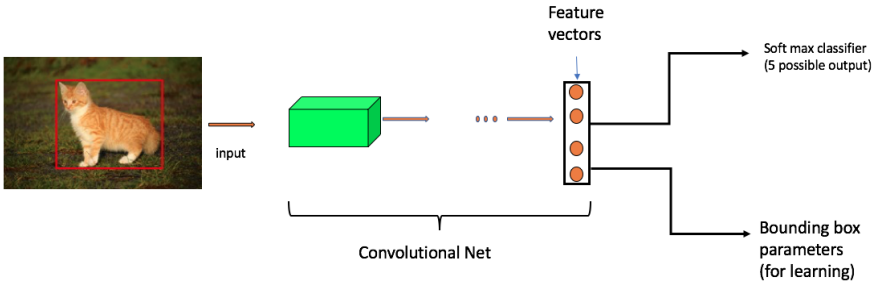
[This pipeline use the conv net discussed in last chapter.]

The output of this pipeline in a feature set. The feature set is input into a softmax -as discussed in the last chapter- to compute the predictive classes. The predictive classes in this project consist of 5 object categories - there are 5 possible outputs from softmax, this corresponds to the number of object classes in the input labels , the classes considered are[2] -:

- Cars
- Bicycles
- Big Pedestrians
- Small Pedestrians
- Background (None of the above)

To upgrade this into a localization framework, simply add bounding box as output in the learning pipeline as below:

Figure 3.5: Add localization module



[This pipeline use the conv net discussed in last chapter. In localization we assume there is one object class per image though there might be multiple object in a single image]

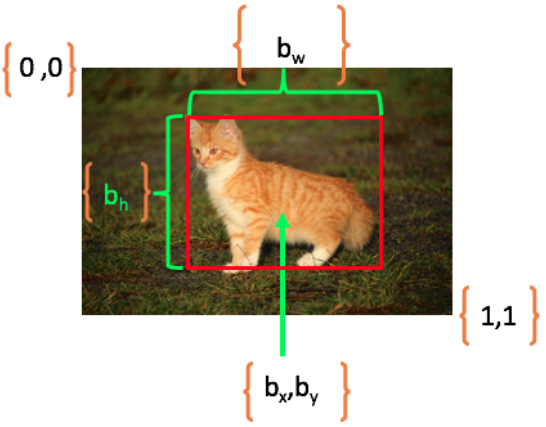
[2]cat class object is not part of the data set. It is used in the example for pedagogical illustration

The output now carries the bounding box information. But how is the bounding box specified? The bounding box is described by 4 variables:

$$[b_x, b_y, b_h, b_w]$$

The b_x, b_y specify the centre of the image, while b_h, b_w specify the height and width of the image. In addition, let's assume the convention that the upper left of the image is co-ordinate [0,0] and lower-right is co-ordinate [1,1]. This is illustrated below:

Figure 3.6: Bounding box parameters



[The bounding box parameters would also be learned. This would be illustrated later]

3.2.1.1 Simple Illustration

Let the input, x, fed into the conv net be any object class defined in the set below:

{x : x ∈ { Small pedestrian, Big Pedestrian, Bicycles , Cars, Background}}

The target label is given as:

$$y = \begin{bmatrix} O_p \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

The component O_p represent "object present". It serves to detect when an object class is in the image. Mathematically, it represent the probability that an object is not "background". Then there are the parameters that define the bounding box for the object detected. Lastly there is object class parameters that signify which object class is present. Thus c_1 represents (probability) class Small pedestrians, c_2 represents (probability) class big pedestrians, c_3 (probability) represent class bicycles, and c_4 (probability) represent class cars.

Example 3.2.1. Let x = car.

Figure 3.7: Input : Object class car

[input to the neural net. Image source [16]]

Then: $y = [1, b_x, b_y, b_h, b_w, 0, 0, 0, 1]^\top$

Thus y implies that an object is present and the object class is car.

Example 3.2.2. Let x = Background (no object class).

Figure 3.8: Input : Object class background

[input to the neural net. Image source [17]]

Then: $y = [1, ?, ?, ?, ?, ?, ?, ?, ?, ?]^\top$

Thus y implies that no object is present. The other parameters are thus not needed: hence the "?" - meaning "I don't care".

This section ends with a discussion of what loss function should be used? For the object classes, cross entropy loss will be appropriate. For the bounding box parameters L2 loss will be appropriate. For the O_p a logistic loss or L2 loss will do the job.

Having discussed object localization in depth, it is now right to extend this model for detection.

3.2.2 Sliding Window Algorithm: Sequential Execution

In the sliding detection algorithm, a closely cropped image of the object classes are designed and input into the conv net. This is illustrated below.

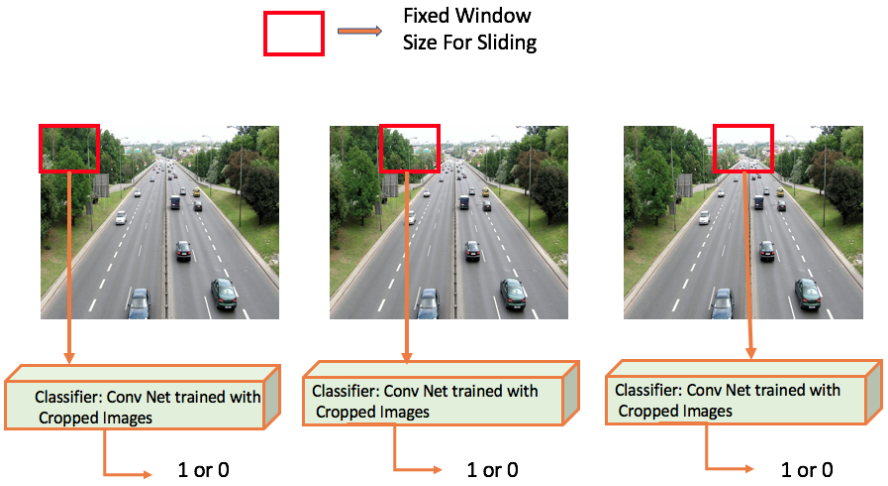
Figure 3.9: Closely cropped image for use in sliding window algorithm

[Label 1 signifies this is a car. And 0 signifies background. Image 2 source [18] Image 4 source [19] [16]]

Train a conv net as illustrated in 3.4 to identify the cars. Then use the trained conv net in sliding window detection as below:

1. Select a fixed window size.
2. Pass the fixed window frame over the image.
3. At each instance feed the region cover by the window into conv net classifier
4. The classifier determines if it is an image or not (0 or 1)

Figure 3.10: Sliding window technique Illustration



[Label 1 is car . Label 0 is background. This illustration implies that the conv net had been trained with cropped image. During testing, the sliding window of same size as cropped image used for training is sequentially applied over test image. A each instant, the conv net is used to test if an image is present or not.]

Lastly, resize the fixed window into a larger size and feed into the conv net - the entire process is repeated with larger window size. The name "sliding window" comes from using the square box and sliding it over the image at specified stride - stride is how much the rectangle in previous region over lap or shift as it moves to the next position. The draw back of this method is

1. It is computationally expensive when used with conv net. The conv net takes appreciable time to classify objects. Now, doing this for many window regions will be prohibitively expensive.
2. It might not be suitable for real time object detection for reason(1).

When used with HOG or SIFT this approach might be be less computationally expensive. However, the aim in this project is to leverage the use of state-of-the heart technique. The next section describes a more optimal way use this technique with conv net.

3.3 Convolutional Sliding Window: One Forward Pass Execution

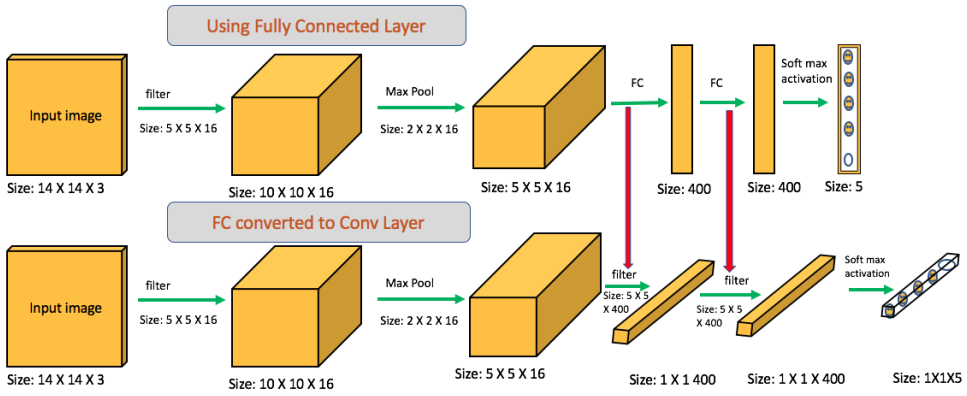
This section shows how to implement, in a computational efficient way, the sliding window algorithm with conv net as classifier. But before this, it is important to know

how to convert fully connected (see last chapter) layer to convolutional layer. This section takes inspiration from [20].

3.3.1 From Fully Connected Layer to Convolutional Layer

From last chapter, the fully connected layer was shown to be part of of the conv net architecture. This section documents how to make this into convolutional layer. First, consider the figure below:

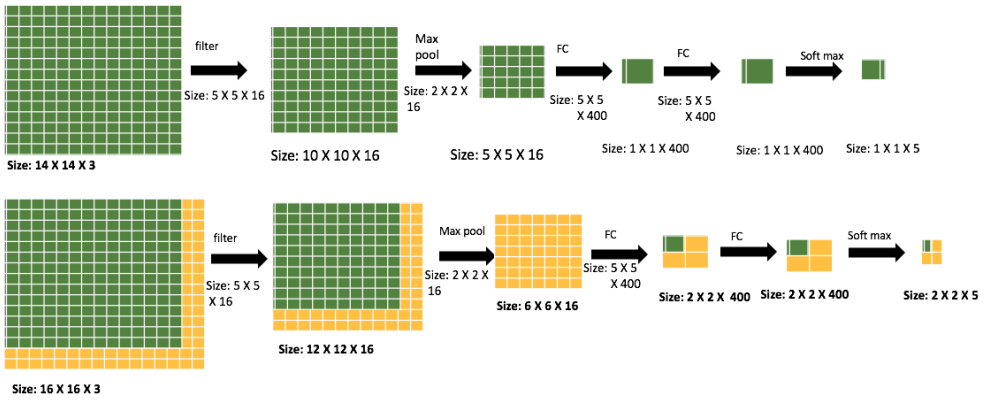
Figure 3.11: Conversion of Fully connected layer to Full convolutional layer



[There are 5 object casses in the classification problem. Hence the output of the softmax is 5. The empty circle signifies background or no object class. In actual implementation the size of the input image will be far bigger than the one illustrated.]

The pink downward pointing arrow shows the point at which the FC layer is replaced with Conv layer. The benefit is that it makes for efficient use of the sliding window in the conv net. In particular it can be observed that certain the operations of the network are duplicated. This can be exploited to achieve efficiency - by sharing repeated operations across different parsing of the sliding window. Consider the figure below:

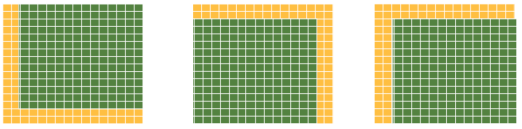
Figure 3.12: Implementation of sliding window algorithm with conv net



[The architecture leverages the FC conversion to Filter operation. It should be noted that the the dimensions have be represented by rectangular dimension - the channel depth has been omitted]

In the figure above, it is assumed that we trained with cropped input image of size $14 \times 14 \times 3$. Let's despicts the scenario where FC is used - no conversion of FC to filter. In the testing phase, assume test image of size $16 \times 16 \times 3$ and window size of $14 \times 14 \times 3$ corresponding to the size of image used during training. The window will fill in the position shown in the superimposed green on yellow grid which represent the test image. In the next iterations or epoch,the windows are shifted as depicted in the figure below:

Figure 3.13: Image shift positions in sequential execution Model

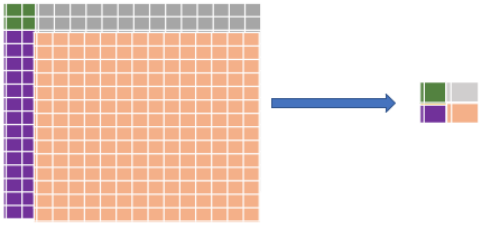


[Shift positions assume using fully connected layer was not transformed to conv layer. There are 4 shift in total. The first is represented in green grid on yellow background in fig 3.12.Each shift is run separately as new training epoch. So the operation is not optimal]

However when the FC is converted to conv layer, the entire 4 operation becomes a single operation, as depicted in the second conv net of fig3.12. This saves a lot of computation cost - in the actual model the size will be a lot bigger, hence this transformation is needed. The output of fig 3.12 is $2X2X4$, and it corresponds to the output from the four filter positions - thanks to the FC to Conv layer conversion, this is now a single operation.

To see this observe the figure below. Each filter position now corresponds to a particular point in the $2X2X4$, output. This is possible because most of the operations are duplicated:

Figure 3.14: Window position superimposed: Optimal execution



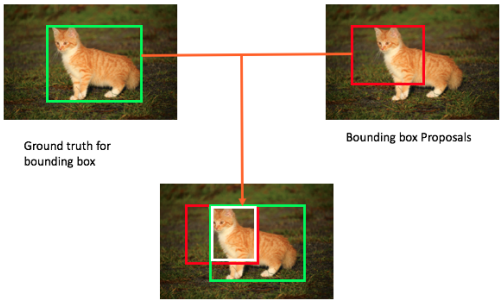
[In the left rectangular grid, the colors represent the 4 window shifts positions in 3.13 superimposed to show . Thus the 4 shifts are trained as single epoch now. This is possible thanks to FC to conv layer conversion. In addition there is sharing of shared operations implied.]

This section discusses the convolutional implementation of the sliding window. By using conv layer instead of FC we can go from the sequential sliding window implementation to one-forward pass implementation. To end this section, it should be observed that no mention was made of the bounding box position. Indeeed, the bounding box position - using the technique discussed up till now - won't be accurate. In the next section, we discuss how to a performance metric for bounding box

3.3.1.1 Evalualting Object Localization: Intersection Over Union

The intersection of Union (IoU) is a metric for evaluating bounding box. To understand IoU consider the figure below:

Figure 3.15: Intersection of union



[The Ground truth is green rectangle. The bounding box proposal is red. The intersection is the white rectangular region]

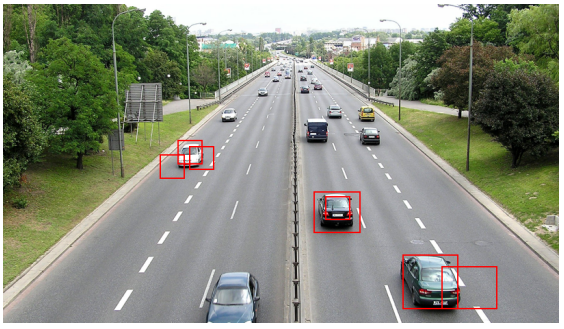
Given the ground truth bounding box and the bounding box proposals. The IoU is given as the area of the size of the intersection of the two rectangles (that is white rectangle) divided by the size of the union of the rectangles. It is a measure of overlap of the 2 bounding boxes.

$$IoU = \frac{\text{size of intersection region - represented by white rectangle}}{\text{size of union of the ground truth and proposal}}$$

If the ground truth matches the bounding box proposal then the IoU is 1. Typically, the proposal is accepted if the IoU ≥ 0.5, otherwise it is rejected. The bigger the IoU the more stringent is the the model and the better the bounding box. The model will also have to train on this metric: this will be dexplained later.

3.3.1.2 Non Maximum Suppression

Figure 3.16: Non maximum suppression



[The Non maximum suppression is use in situation when there are numerous detection for a given object. Note in the image this is denoted by the red rectangles around the cars. Only few cars were selected for illustration purposes]

As illustrated in the figure above, the Non max suppression is cleaning up multiple bounding box detection for a given object: it does so by first discarding bounding box with IoU ¡ 0.6, or any predefined threshold; then it selects the bounding box with the highest predictive class while deleting the bounding boxes that has high over lap with the selected one. The selected box is thereafter, proposed for the prediction algorithm. The predictive class represent the output probability of whether it contains an object or not - regardless of the class. This was discussed in earlier section. The non-max suppression algorithm can be carried on each of the object classes available during the training epochs.

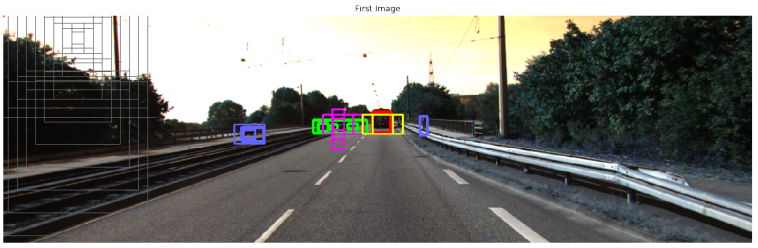
For example, In the middle vehicle for example the centre rectangle - which would we assume has low IoU and low class prediction compared to the outer one - will suppressed since it overlaps with the outer rectangle. Then the algorithm moves to other

rectangle that don't overlap with the centre vehicle. For example far left vehicle, and repeat the same process.

3.3.1.3 Anchor boxes

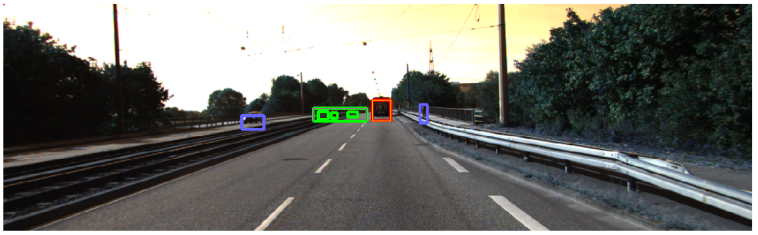
In other to optimize the detection of objects and bounding box prediction, this project makes use of default bounding box similar to [21]. Fixed-size collection of bounding boxes and scores for the presence of object class classes in those boxes, followed by a non-maximum suppression step to produce the final prediction. This is followed the classification of the object present in the box using the neural net. The bounding box is trained with a linear regression model. This is illustrated below:

Figure 3.17: Default Bounding Box Anchors



[Different bounding box anchors to capture various object in an image]

Figure 3.18: Default Bounding Box After non maximum suppression



[Default bounding box Normalized]

This approach enables multiple objects that fall into overlapping bounding box to be detected easily. An example would be when a pedestrian is in front of a car: this might just be treated as single bounding box because they fall into the object classes to be detected and the bounding box are within same area. Using anchor boxes in such scenario of various sizes however, these two objects can be detected separately.

3.4 Conclusion

In this chapter the focus was on a shift from a image classification to image detection algorithmic analysis. The fundamentals idea were discussed followed by the state-of-the-heart approach (Con net sliding window). The next chapter will focus on the experimental results from putting all this description in multiple implementation.

Chapter 4

Theoretical Model Analysis

4.1 Introduction

[1] Up until recently, most neural net architecture have been treated as black box with no theoretical explanation of why they work so well. The aim of this chapter would be to give some theoretical explanations of neural net architecture using concepts from information theory. This will help understand the behavior of some of the the models and experimental results that would be discussed in the next chapter.

This chapter starts by defining the learning problem from rate distortion theory perspective. Next, this is extended to describe the neural net inner architecture. Lastly, the training behavior of the model is explained using compression and estimation phase of the stochastic gradient descent.

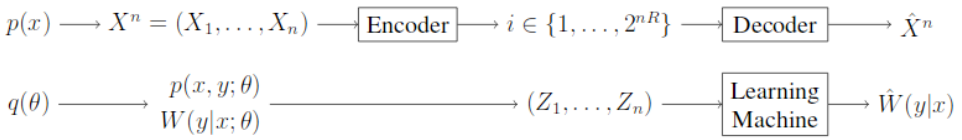
4.1.1 Supervised learning and rate distortion theory

It turns out that a good place to start studying the theoretical underpinnings of the learning mechanism discussed so far is by using analogy from rate distortion theory. This will help clarify further why a performance metric is important and also establish a bound on the achievable learning capacity [22].

The figure below is important for the explanation in this section:

[1] This section is motivated by my study on rate-distortion theory. Somehow, I reasoned that there is absolutely correlation between the rate distortion theory and supervised learning paradigm. Fortunately, I was able to find related papers that discusses such relationship exactly as I had imagined it in my mind. This might help us understand the neural net black box appropriately.

Figure 4.1: rate distortion encoder and decoder versus supervised learning



[Image source [22].]

4.1.1.1 Rate Distortion for encoder and decoder

In rate distortion theory, given a source distribution p(x), which produces an n-length sequence X^n, the encoding function is defined as :

$$f_n : X^n \to \{1, 2, ..., 2^{nR}\}$$

where R is the encoding Rate. The decoding function does an approximate or quantized reconstruction (reconstruction problem) of X^n like so:

$$g_n : \{1, 2, ..., 2^{nR}\} \to \hat{X}_n$$

Typically, a distortion function :

$$d : x \times \hat{x} \to R^+$$

measures the cost of representing the symbol x by $\hat{x}$: consequently the average per symbol distortion of the elements of the sequence is then given as [23]:

$$D = Ed(X^n, g_n(f_n(X^n)))$$

- In summary it is desired that the distortion function is bounded.
- At each instance the smallest distortion region that best approximate the sent sequence is selected
- This is similar to supervised learning where our distortion function would normally be the performance function described in the previous chapter.
- Again like the previous chapter , the aim is to minimize the distortion (increase performance or decrease error).

Next, an elaborate bayesian statistical analogy is described in next section.

4.1.1.2 Bayesian learning perspective

Using fig 4.1, for simplicity we assume a parametric statistical framework - by this we mean that the sample data comes from a population that follows a probability distribution based on a fixed set of parameters [24].

Consider a simple classification model: let the data points X $\in \mathbf{X} \subset R^d$ and its label Y $\in [M]$ be distributed according to $p(x, y|\theta)$ where $\theta \in \Lambda \subset \theta$ indexes a parametric family of distributions $\mathbf{D} = \{p(x, y|\theta) : \theta \in \Lambda\}$.

Suppose there is a gennie that chooses $\theta \in \Lambda$ - this is analogous to choosing each source symbol in a sequence randomly according to distribution p(x) as described in last section. The learning machine (like the decoder) obtains a sequence of n samples $Z^n = (X^n, Y^n)$ where each pair $Z_i = (X_i, Y_i)$ is drawn i.i.d according to $p(x, y|\theta)$ [22]. The task of the learning machine is to select a classifier defined by [22]:

$$w : \mathbf{X} \to [M]$$

(this is the analogy of the reconstruction problem described above). Just like in the encoder-decoder analogy, there is need to define a performance metric.

To do this, define $q(\theta)$ to be a prior distribution over the parametric family. If $q(\theta)$ is the true distribution over the parameter space, then the performance metric collapses to average loss over many instances of learning problem. Furthermore let,

$$W(y|x, \theta = p(y|x, \theta)$$

describes a regression function that takes in x $\in \mathbf{X}$ and produce as output M dimensional vector of probability that x belongs to class y. Then $\hat{W}(y|x)$ is an estimate of W. The loss is then defined as L_p distance between the M-dimensional vector formed by $W(.|x; \theta)$ and $\hat{W}(.|x)$[22]:

$$L_p(x, \theta; W, \hat{W}) = \left(\sum_{y=1}^{M} |W(y|x, \theta) - \hat{W}(y|x)|^p \right)^{\frac{1}{p}}$$

The L_p loss provide a comprehensive sense of classifier performance. Hence, the justification for using this in the model - this will be discussed in later section. Lastly, the Kullbeck-Leibler divergence between W and $\hat{W}$ [22]:

$$L_{KL}(x, \theta; W, \hat{W}) = \sum_{y=1}^{M} W(y|x, \theta) \log\left(\frac{W(y|x, \theta)}{\hat{W}(y|x)}\right)$$

This corresponds to the cross entropy loss or log loss mentioned in the previous chapter. The pinker's inequality states that:

$$L_{KL} \geq \frac{L_1^2}{2}$$

Hence there is a relationship between L1 loss and the L_p loss. Thus using this L_{KL} is perfectly valid. Moreover a lower bound on L_1 risk can be translated to a bound on the L_{KL} divergence between W and $\hat{W}$ averaged over θ [22].

The following lemma follows from the rate distortion analysis : There exist a Bayesian learning rule with Bayes Risk $< \epsilon$ only if

$$I(X^n, Y^n; \theta) \geq \min_{\substack{p(W|\hat{W}) \\ E[L_1(W|\hat{W}] \leq \epsilon}} I(W; \hat{W})$$

where I is used to denote the mutual entropy. Thus to achieve a given low error rate the mutual information on the left hand side which is a function of the joint distribution (X,Y) must be greater than mutual information from the map rule. As will be shown seen, this is analogous to the information bottleneck - information bottleneck is a function of jount distribution of X,Y. The aim of training is to make ϵ as small as possible and make learning approach the right hand side. As will be shown in the later section of this chapter, when this error becomes small, an efficient representation of input is then possible - thus the model becomes more optimal.

- Summary of results [22]: Following the analysis in [22], it has been shown that , in order to derive the average L_p error below a threshold ϵ the mutual information between training samples and the parameters θ must be at least as great as the differential entropy of the posterior plus a penalty term that depends on ϵ and a sample theoretic term quantity called the interpolation dimension .The interpolation dimension is the cardinality of the smallest (finite) interpolation set. It measures the number of data points from the posterior distribution needed to uniquely interpolate the entire function [22].

To give an overall picture ,the true model parameter θ is not always known a priori. However, we can choose a parameter say $\hat{\theta}$ from a parametric family $q(\theta)$ and try to estimate how well it describes the true parameter which describes the data or model - the Kullbeck divergence is meant to test this "how well".

In summary, it is explained (statistically speaking) that the problem at hand is same as trying to model a stochastic phenomenon by a regular parametric family of

distributions $\mathbf{F} = \{F_\theta : \theta \in \Theta\}$ where $\Theta \in R^p$. However, we observe "N" iid outcomes from the phenomenon $\{X_1...X_n\} \overset{iid}{\sim} F_\theta$ for some $\theta_0 \in \Theta$ (for example consider the image classes randomly selected) but do not know/observe the $\theta \in \Theta$ that generated them - we don't observe the weights and biases that will correspond to a correct classification of the randomly selected images by the algorithm. The obvious inference task is : how can the θ that generated the sample $\{X_1...X_n\}$ be estimated? Since the $\{X_1...X_n\}$ is all we have, we can use a function of the input $\{X_1...X_n\}$ - called estimator [2]- to learn this . The estimator, used in this project is the Conv net and the parameters would correspond to the weights and biases discussed in chapter 2. The KL distance is used to test how well the functional approximator is doing. This is similar to the reconstruction problem discussed in the last subsection.

However wouldn't it be nice if the conv net can be opened to observe how it works? The next section does that.

4.2 Opening the Neural net Black Box

First, the section starts with discussion on Information Bottleneck and then proceed to leverage this concept to have a deeper understanding of how the black box (neural net) described in the previous chapter works in depth.

4.2.1 Information Bottleneck (IB)

The Information bottleneck is introduced here, to express the relevant information that an input random variable $X \in \mathcal{X}$ contains about an output random variable $Y \in \mathcal{Y}$. Given the joint distribution $P(X, Y)$, this defines the mutual information $I(X; Y)$ [23].

Statistically, it represents the minimum sufficient statistics of X wrt Y, denoted $\hat{X}$ [25] - the minimmum representation that captures the mutual information. Assuming the markov chain

$$Y \to X \to \hat{X}$$

From Data processing inequality[23] we have:

$$I(X : \hat{X}) > I(Y : \hat{X})$$

Thus finding an optimal representation (or reconstruction) is equivalent to the Lagragian minimization:

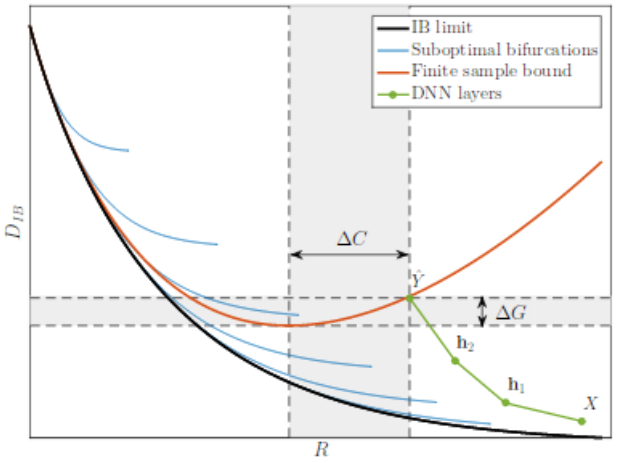
[2]the estimator is required to be sufficient statistics for learning the true parameters

$$\mathcal{L} = I(X : \hat{X}) - \beta I(Y : \hat{X})$$

where β is tradeoff parameter between representation rate, $R = I(X : \hat{X})$ and relevant information $I(Y : \hat{X})$ [25].

Of relevance to this project is the IB curve, shown below. The IB curve shows that it is possible to have bifurcations to sub optimal curves at critical values of β. This will be very important in explaining why some models might be better for certain object scale than others - in essence some models might perform sub optimally.

Figure 4.2: Information curve for Deep Neural net



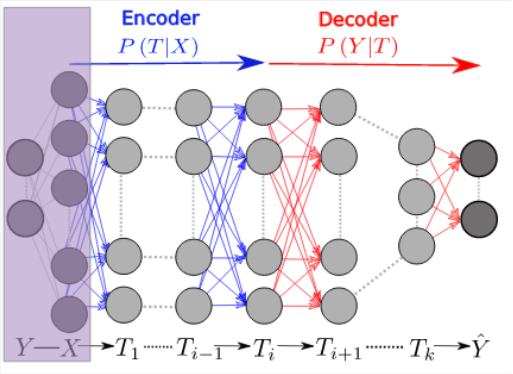
[The black line is the optimal achievable IB limit.The blue line are sub-optimal bifurcations obtained by remaining in the same representation. The green line shows hypothesized path for the hidden layers. Though the training distortion may be low (green line) the actual distortion may be as high as the red line. The aim in training model is to shift the DNN layers as close as possible to the optimal curve and obtain lower complexity and good generalization. **Source**[25].]

The bifurcations represent the phase transitions between different topological representations of $\hat{X}$ and are properties of joint distribution independent of the model [25]. At optimal point, the neural model should be able learn to extract the most efficient informative features, or approximate minimal sufficient statistics, with the most compact architecture[25]. In fact as will be shown in later chapter varying the size of the hidden layer will affect model accuracy and give better representation of $\hat{W}$ for different object class input.

4.2.2 Stochastic Gradient Descent and Information Bottleneck

The figure below will be important for this section:

Figure 4.3: Markov View of DNN layers



[Image source [26].]

This needs little explanation. The layers labelled T form the hidden layers and the arrows signifies they form a Markov chain of successive internal representation of input X [26]. The desired output layer is Y; it is seen only during training,through a finite sample of the joint distribution, $p(X, Y)$, and is used for learning the connectivity matrices between consecutive layer . The output layer is $\hat{Y}$.

The representation is grouped into an encoder, $P(T|X)$, quantified by

$$I_x = I(X;T)$$

and a decoder, $P(Y|X)$, quantified by :

$$I_y = I(T;Y)$$

Interestingly, the IB can be viewed as representing the optimal representation: that compress the input for any desired mutual information on the output Y. The total information stored in the network is given : $\frac{I(Y;\hat{Y})}{I(X;Y)}$ [26].

The information path along the network follows directly from the Data processing inequality [23], as shown below:

$$I(X;Y) \geq I(T_1;Y) \geq I(T_2;Y) \geq I(T_3;Y) \geq I(T_4;Y)...I(T_k;Y) \geq I(\hat{Y};Y)$$

- Summary of results from [26]: In [26] it is shown that Stochastic Gradient Descent with which our model will be trained consist of 2 different phases:

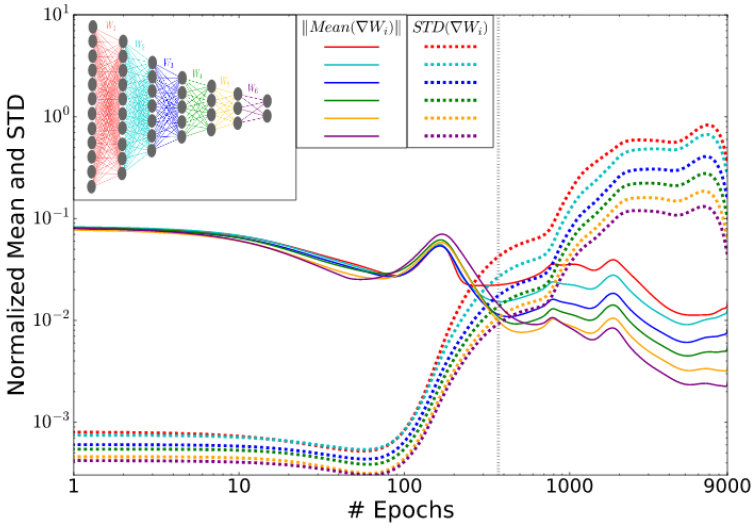
– *empirical error minimization phase (ERM) or drift phase*

– *compression phase or diffusion phase*

The ERM last for a few hundred epochs, and during this phase the neaural layers tend to increase the information on the labels (increase I_Y) while preserving the Data Processing Inequality order. In the compression phase, gradient fluctuation is high, causing reduction in mutual information on variable X.

- The cost of the training epochs in standard deep learning are spent on compression of the input to efficient representation and not on fitting the training labels
- The representation compression phase begins when the training errors becomes small. Additionally, the Stochastic Gradient Decent (SGD) epochs change from a fast drift (smaller training error) into a stochastic relaxation, or random diffusion, constrained by the training error value .
- Adding hidden layers reduce the number of training epochs for good generalization.

Figure 4.4: Graph of Stochastic gradient descent phases across epoch



[Image source [26].]

Above is a plot of the normalized mean and standard deviations of the weights stochastic gradients (in the samples batches), for every layer of a DNN (shown in the inset), as function of the SG epochs.

The grey line marks the transition between the first phase, with large gradient means and small variance and the second phase, with large fluctuations and small means (diffusion, low SNR) Shwartz-Ziv and Tishby [26].

4.3 Conclusion

In this section, an information theoretic view of the internal mechanism of the model is presented. This gives us a good theoretical underpinnings with which behavior of the models in the next chapter will be explained.

Indeed the topic in this chapter represent a recent development in deep learning. It is believed that in the nearest future deeper analysis of some of the behavior of the network will be available.

Chapter 5

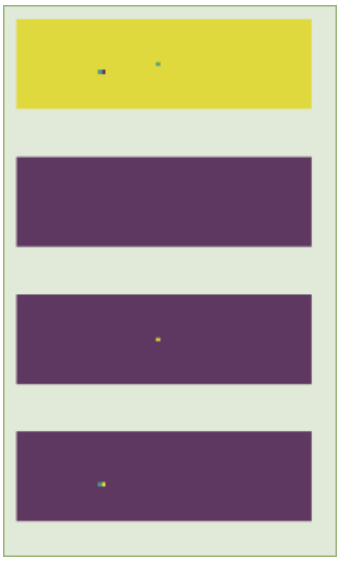
Experimental Results

In this chapter, the results of the detection model training is discussed. The chapter starts with additional optimization techniques used in actual implementation of the model. This is followed by an results on the behavior of the model performance as the sale of the object varies. Lastly, more results from different runs of the model will presented. The explanation of the model behavior will rely mostly on theoretical analysis from previous chapter and in some cases discussions on discussions from other chapters. This part of the project put everything together to achieve implementation results.

5.1 Optimization Technique Used in Implementation

Generally the label (desired output,Y, used during training) of the network consist of a image mask for different object classes plus a mask for background (negatives).

Figure 5.1: Input



[The label used for the network is a image mask. The negative signifies the background - plain purple of second rectangle.]

This makes the entire model discussed in this chapter quite different from other public models - it is not known either if this difference might help performance. The object classes considered are : Small pedestrian, big pedestrian, cars, bicycles.

5.1.1 Dropout And learning rate

Dropout technique has been effective in reducing model overfitting. A dropout of 0.5 was used generally. The effect of the drop out is to make the model implementation able to generalize and force the network to be more accurate. This is done by dropping out input and output connection of neural unit with probability set to 0.5. This is equivalent to randomly sampling "a network" from the full network.

A learning rate of 0.5 was used during training. The update of weights is generally given as $W := W + \Delta\mathcal{L}$, where $\mathcal{L}$ is the cost function, Δ is the learning rate, and W the weight. Thus the learning rate, determines how the model parameters are updated - or how often the model learns.

In some implementation, an exponentially decaying learning rate is used. This is defined as :

$$\text{learning rate} \times \text{decay rate} \times \frac{global\_steps}{decay\_steps}$$

Other parameters aside the learning rate are arbitrarily tweaked to give optimal performance.

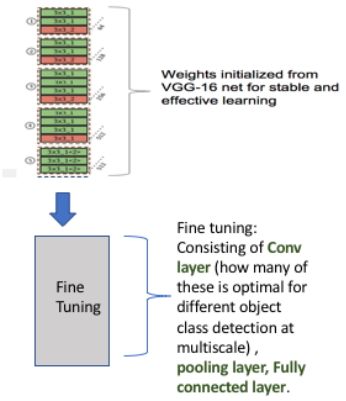
5.1.2 Data Augmentation

Since the project takes a data driven approach to object detection. One way to solve the scarcity of data for learning the large number of model parameters is through data augmentation. The general approach consist in using simple geometric transformations (such as rotating, shifting). In the initial training of the models, no augmentation was used but later training were done with augmented set

5.1.3 Weight Initialization

Conv net has huge parameters which might be difficult to train. To facilitate training and convergence, the parameters are initialized using a pre-trained model - specifically VGG 16 [27]. Pr3e-trained weights have been shown to give better performance [28].

Figure 5.2: Model Finetuning



[With the initialisation set as above, the implementation focuses on finetuning training for object detection.]

5.1.4 Mini Batch Stochastic gradient

Model parameter update is achieved through gradient descent using the back propagation algorithm. The (stochastic) batch gradient descent provides a more optimal way to train. The bacth size is set to 64 for each training epoch

5.1.5 Batch Normalization

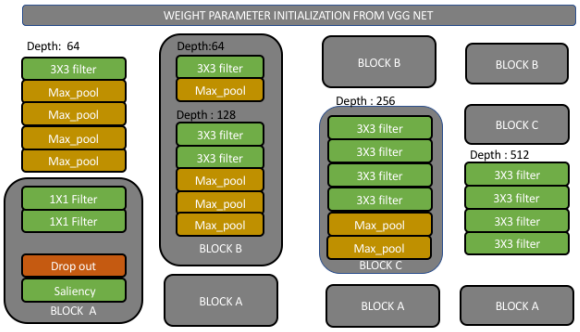
Batch normalization has been shown to improve improve covariant shift [29], reduce dependence of gradients on parameter size or initial values[29]. Thus for each training epoch and batch size 64 we perform batch normalization as below where x is input data in a batch:

$$\hat{x_i} \leftarrow \frac{x_i - mean_b}{\text{standard\_deviation}_b}$$

5.2 Model Results

5.2.1 First Model Results

Figure 5.3: Network Architecture

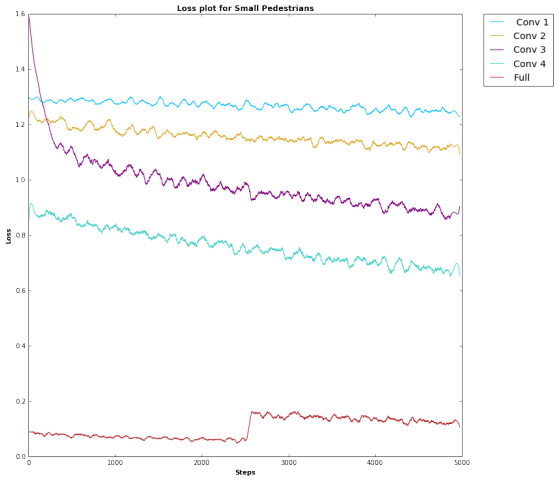


[From left to right the architecture is referencede as conv 1, conv 2, conv 3, conv 4. From left to right observe that the network depth is increasing. The aim is to find which architecture better captures varous object class]

The aim of this architecture is to understand the variation of model accuracy for multiscale object as the conv layer becomes deeper. From left to right, the figure shows a progression from a simple model to a more deeper architecture. Each architecture was trained separately. Modules that are repeated are made into blocks. They are then used as part of the next architecture. A full network consisting of the Depth:512 block above replicated twice was also trained . A plot of error was made for each model. The plot was smoothened using savizkygolay[1] filter using a window size of 51 and polynomial of degree 3. For these architectures, Adam gradient Optimizer was used with a decaying learning rate.

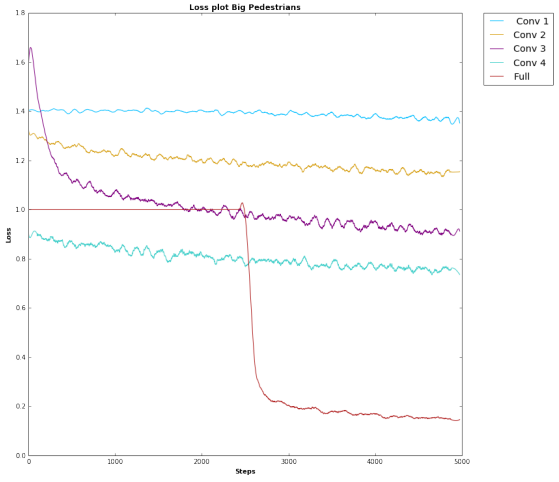
[1] "https://en.wikipedia.org/w/index.php?title=Savitzky%E2%80%93Golay_filter&oldid=807248668"

Figure 5.4: Cross entropy loss for small pedestrians



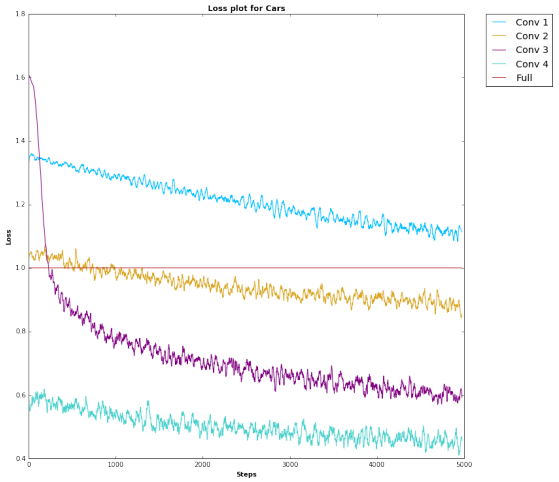
[Each of the architecture in fig 5.3 has been labelled conv 1 to 4 in this figure. The last architecture is the full layer.]

Figure 5.5: Cross entropy loss for Big pedestrians



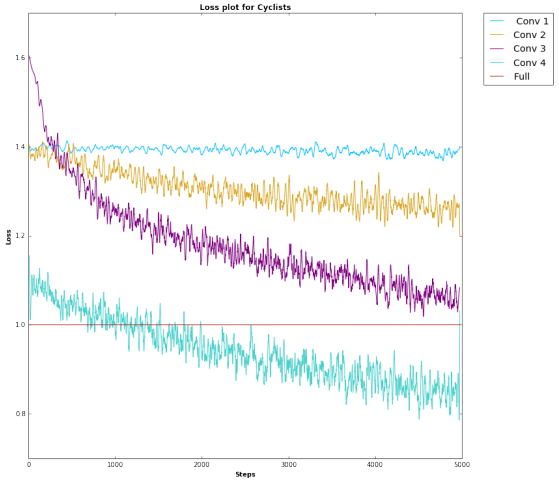
[Each of the architecture in fig 5.3 has been labelled conv 1 to 4 in this figure. The last architecture (not shown) is the full layer.]

Figure 5.6: Cross entropy loss for small pedestrians



[Each of the architecture in fig 5.3 has been labelled conv 1 to 4 in this figure. The last architecture is the full layer]

Figure 5.7: Cross entropy loss for cyclists



[Each of the architecture in fig 5.3 has been labelled conv 1 to 4 in this figure. The last architecture is the full layer]

The following conclusions can be made:

- The Full network is not always the best for all class labels
- The full network performs well on at least small and (after long iteration) big pedestrians
- For Cars Conv 4 performs absolutely better

Using the understanding from last chapter it is possible to explain the model behaviour. The total mutual information the full network retains $I(X^n, Y^n; \theta)$ about the sample input is much more than other models - since the parameter size θ is bigger. This induces a small training error. With a small error , entering into the compression phase during batch training will result in good internal representation of object classes. Since the Conv 4 (the last architecture in 5.3 is quite similar to the full network it is possible for similar performance to occur.

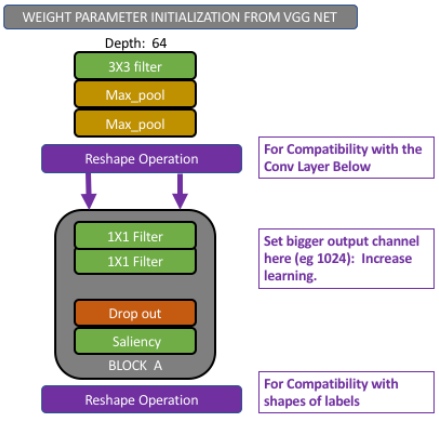
Since the models conv 1, conv 2,and conv 3 have less hidden units. Their poor performance can be attributed to bifurcation to suboptimal region in the information bottleneck curve during training. Thuis might have induced a greater error on the model. Thus, the compression phase will not result in good internal representation of data. Note that they also have less parameters than the conv 4 and full network

In particular the architecture in fig 5.3 above has serious caveat: the model architecture are in order of decreasing weight parameters. This causes the models with more parameters to perform well. The next model has same architecture as conv 1 but with increased parameters at the last layer. This causes it to have similar performance with the full model.

5.2.1.1 Single Conv 1 Model Architecture

Indeed we need a model that it simple to train for use in real time object detection. The model in this section consist of single model architecture (similar to first architecture,conv 1, in 5.3) that works well on all object classes.

Figure 5.8: Single Conv architecture



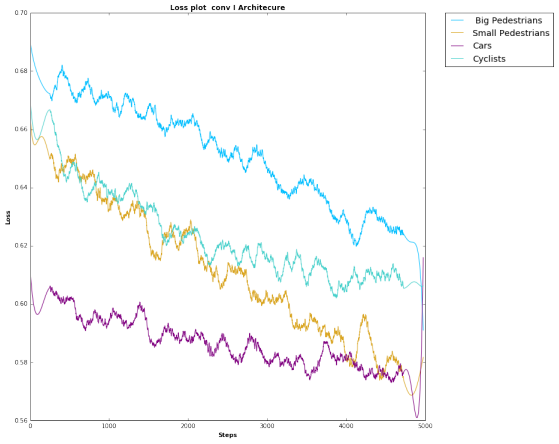
[New Architecture 1 with good performance]

Again, using the understanding from the last chapter, the total information stored in the network is $\frac{I(Y;\hat{Y})}{I(X;Y)}$. In additon we know that since the parameter space is now much larger,the joint mutual information $I(X^n, Y^n; \theta)$ will be high. Expanding this gives :

$$I(X^n, Y^n; \theta) = I(\theta; X^n) + I(Y^n; \theta | X^n)$$

It is easy to infer that this will also cause an increase in IB: $\mathcal{L} = I(X^n : \hat{X}^n) - \beta I(Y^n : \hat{X}^n)$ since the IB depends on the joint distribution of X,Y. This in turn causes the information stored in the network $\frac{I(Y;\hat{Y})}{I(X;Y)}$ to be high, and a good output layer representation is thus delivered.

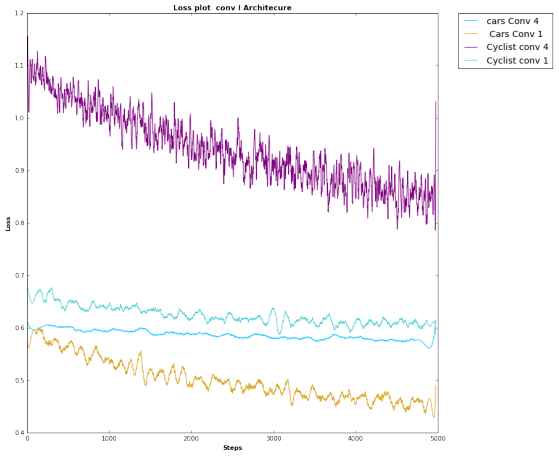
Figure 5.9: Training Loss plot for New Conv 1 Architecture



[Observe the overall good performannce on all lables]

In the figures below the performance of this model is compared with that of the best performance obtained in last architecture. Conv4 performed bast for class object cars and cyclist, thus it desirable to compare conv 4 for these object classes and the new conv1. The result is the figure below:

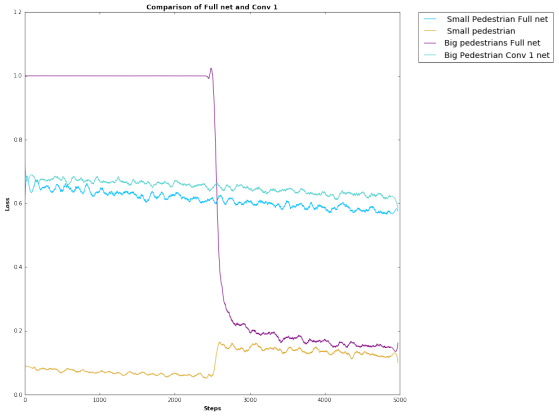
Figure 5.10: New Conv1 versus Conv4: Training loss



[Object classes cars and cyclist were considered here]

Similarly the full network performed best for class object big and small pedestrians , thus it desirable to compare full network for these object classes and the new conv1. The result is the figure below

Figure 5.11: New Conv1 vs Full network: Training loss



[Object classes Small and Big Pedestrians were considered here]

5.2.2 Second Model Results

The model in this section is motivated by the fact that it is not usually desirable to reduce the feature set : as done in this previous section. Moreover, in this implementation mean absolute error (MAE) was used to measure model performance. Rather than increase the parameter size, the function was used to pool features from the labels- . Apart from these differences, the model architecture remains same as illustrated 5.3.

Appendix A

Appendix : Gentle Introduction to CNN

Bibliography

[1] 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.

[2] 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].

[3] 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`.

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

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

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

[7] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34: 2189–2202, 11 2012. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.28. URL `doi.ieeecomputersociety.org/10.1109/TPAMI.2012.28`.

[8] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL `http://arxiv.org/abs/1504.08083`.

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

[10] R Hecht-Nielson. Theory of back propagation neural network. *Neural networks*, supplement 1, 1998.

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

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[13] Andrew Ng. Deep learning specialisation course, coursera, 2016. URL https://www.coursera.org/learn/convolutional-neural-networks/lecture/6UnU4/convolutional-implementation-of-sliding-windows. [Online; accessed 19-December-2017].

[14] pixabay. Cat.kitten, 2017. URL https://pixabay.com/en/cat-kitten-cat-baby-young-cat-1074644/. [Online; accessed 19-December-2017].

[15] Digital Image. Funny cat dog, 2017. URL https://i.ytimg.com/vi/daaoTw4VtGE/maxresdefault.jpg. [Online; accessed 19-December-2017].

[16] Digital Image. Cat.kitten, 2017. URL http://www.zastavki.com/eng/Auto/Audi/wallpaper-64028.htm. [Online; accessed 19-December-2017].

[17] Digital Image. road, 2017. URL http://www.peency.com/images/2015/05/13/road-sunset-wallpaper.jpg. [Online; accessed 19-December-2017].

[18] Digital Image. car, 2017. URL http://felixwong.com/2017/01/cars-in-cuba/. [Online; accessed 19-December-2017].

[19] Digital Image. car, 2017. URL http://pbrown.co.uk/photos/photos-5/files/page7-1007-full.html. [Online; accessed 19-December-2017].

[20] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. URL http://arxiv.org/abs/1312.6229.

[21] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. URL `http://arxiv.org/abs/1512.02325`.

[22] Matthew Nokleby, Ahmad Beirami, and Robert Calderbank. A rate-distortion framework for supervised learning. *IEEE International Symposium on Information theory*, 09 2015.

[23] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006. ISBN 0471241954.

[24] Wikipedia. Autonomous car — wikipedia, the free encyclopedia, 2017. URL `https://en.wikipedia.org/wiki/Parametric_statistics`. [Online; accessed 19-Dec-2018].

[25] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015. URL `http://arxiv.org/abs/1503.02406`.

[26] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

[28] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. *Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 153–160, 2009. URL `http://jmlr.csail.mit.edu/proceedings/papers/v5/erhan09a/erhan09a.pdf`.

[29] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015. URL `http://arxiv.org/abs/1512.07108`.