

CAPITAL UNIVERSITY OF SCIENCE AND
TECHNOLOGY, ISLAMABAD



Monocular Depth Perception Using Deep Learning

by

Sharjeel Anwar Syed

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the

Faculty of Engineering

Department of Electrical Engineering

2020

Copyright © 2020 by Sharjeel Anwar Syed

All rights reserved. No part of this thesis may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, by any information storage and retrieval system without the prior written permission of the author.

This work is dedicated to my parents especially my late mother, my wife, my children, my brothers, my friends, my supervisor Dr. Imtiaz Taj and especially dedicated to my mentor Dr. Fida Muhammad.



CERTIFICATE OF APPROVAL

Monocular Depth Perception Using Deep Learning

by

Sharjeel Anwar Syed

MEE-163002

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Nabeel Ali Khan	FU,Rawalpindi
(b)	Internal Examiner	Dr. M.Tahir Awan	CUST,Islamabad
(c)	Supervisor	Dr. Imtiaz Ahmed Taj	CUST,Islamabad

Dr. Imtiaz Ahmed Taj

Thesis Supervisor

December, 2020

Dr. Noor Muhammad Khan
Head
Dept. of Electrical Engineering
December, 2020

Dr. Imtiaz Ahmed Taj
Dean
Faculty of Engineering
December, 2020

Author's Declaration

I, **Sharjeel Anwar Syed** hereby state that my MS thesis titled “**Monocular Depth Perception Using Deep Learning**” is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

(**Sharjeel Anwar Syed**)

Registration No: MEE-163002

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled **Monocular Depth Perception Using Deep Learning**” is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been dully acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

(Sharjeel Anwar Syed)

Registration No: MEE-163002

Acknowledgements

First of all, I would like to thank Allah SWT for His countless blessings showered upon me throughout my life. He has always given me the best opportunities regardless of my weaknesses. I pray that Allah SWT allows me to be His humble servant and blesses me and my family with steadfastness on His religion.

Then, I would express my heartiest gratitude and respect for my supervisor Dr. Imtiaz Ahmed Taj. This work would not have completed without his guidance and support. His strong command on my area of research and extra ordinary problem solving skills are the key factors in completion of this thesis. I will always remember his kind behavior while conveying the technical arguments about the topic of research. Indeed it was an honor for me to work with such a nice, thorough and dedicated professional.

I am thankful to all my course instructors Dr. Imtiaz Taj, Dr. Fazal ur Rehman, Dr. Amir Iqbal Bhatti and Dr. Raza Samar in Capital University of Science and Technology, Islamabad, for developing my knowledge base in the field of Electrical Engineering during the course work that helped me in choosing the area of research for Masters.

Special thanks to my mentors Dr. Fida Muhammad, Mr. Muhammad Saqib Mansur, Mr. Saqib Wazeer, for their valuable suggestions and support during these years. I am also thankful to Mr. Farooq Younas Bhatti and my friends for their assistance throughout my thesis. I would also like to take this opportunity to thank my family which includes my father, my late mother, my wife, my kids and my brothers for their determined support in continuation of my studies. They were always there to help me with all their abilities. Special thanks are also due to my wife for her encouragement and moral support from the start of this thesis, till this point.

I want to appreciate all the faculty members of Capital University of Science and Technology, especially the teachers who taught me during my course work, fornurturing my concepts.

Abstract

Depth estimation is one of the vital tasks in Robotic navigation, Autonomous Driving Systems, Advanced Driver Assistance Systems and Surveillance applications. Currently, LiDARs, RADARs, SONARs and Laser Range Finders are used for most of the above mentioned practical applications. Traditionally for vision based solutions to these tasks, areas such as Stereo Vision, Structure From Motion, Optical Flow, Motion Parallax and Active Focus / Defocus were explored, which always came up with bottlenecks in the existing technology. In recent past, Deep Convolutional Neural Networks (DCNNs) have been explored by a great number of researchers for vision based depth estimation and specifically supervised monocular depth estimation. However, there still is a need to explore light architectures, training techniques and efficient deep learning libraries that can provide compact and computationally efficient real-time solutions for these practical applications. This study endeavors at addressing these challenges. In this study, for the first time an approach is presented using a fully convolutional U-Net architecture with Resnet34 or Resnet50 as backbone architectures, and using the Fastai's efficient deep learning library. The Fastai's Dynamic U-Net was further tailored to suit the depth estimation task. Some best training practices adopted by leading deep learning practitioners have been used in this study which helped to produce highly accurate results. The proposed approach gives comparable error rates with state of the art techniques which employ computationally heavy architectures. The proposed architecture was trained and evaluated on the renowned KITTI dataset which contains images of outdoor scenes such as required in the above mentioned practical applications. The network was trained on Sparse ground truth images as originally provided with the dataset. The qualitative and quantitative results demonstrate the effectiveness of the approach adopted in this study.

Contents

Author’s Declaration	iv
Plagiarism Undertaking	v
Acknowledgements	vi
Abstract	vii
List of Figures	xi
List of Tables	xiii
Abbreviations	xiv
1 Introduction	1
1.1 Overview	1
1.2 From Stereo to Monocular Vision Based Depth Estimation	2
1.3 Deep Convolutions Neural Networks (DCNNs)	4
1.4 Research Challenges	6
1.4.1 Minimal Equipment and Time for Training	7
1.4.2 Light Architectures	7
1.4.3 Real Time Performance on Cost Effective Platforms	7
1.4.4 Overcoming the Limitation of Popular Data Sets with Sparse Data for Supervised Learning	8
1.5 Research Objective	9
1.6 Contributions	9
1.7 Thesis Organization	10
2 Literature Review	11
2.1 Binocular Vision Approaches	12
2.1.1 Ambiguity	13
2.1.2 Occlusion	13
2.1.3 Short Range	13

2.1.4	High Computational Cost	14
2.1.5	Hardware Limitations	14
2.2	Monocular Vision Approaches	14
2.2.1	Supervised Machine Learning	15
2.2.2	Unsupervised/Semi-Supervised Machine Learning	16
2.3	Research Gap	18
2.4	Problem Statement	20
2.5	Summary	20
3	Dataset, Libraries and Architecture Used	21
3.1	Dataset	22
3.1.1	KITTI	22
3.2	Fastai Library and Pytorch	23
3.3	Best Training Practices in Fastai	25
3.3.1	One Cycle Training – <i>Freezing/Unfreezing Layers</i>	25
3.3.2	Learning Rate Finder and Discriminative Learning Rates	25
3.4	U-Net Architecture	26
3.4.1	Encoder-Decoder Architecture	27
3.4.2	Resnet-34	30
3.4.3	Resnet-50	32
3.4.4	Fastai’s Implementation of Dynamic U-Net	33
3.5	Summary	34
4	Implementation Methodology	35
4.1	Proposed Methodology	35
4.1.1	Organizing the Dataset/Cleaning of Data	36
4.1.2	Training in Fastai	37
4.1.2.1	Creation of Item List	37
4.1.2.2	Labelling Function for KITTI Dataset	38
4.1.2.3	Creation of Image Data Bunch	38
4.1.2.4	Data Augmentation	38
4.1.2.5	Creation of U-Net-Learner/Model for Training	39
4.1.2.6	Applying Transfer Learning	40
4.1.2.7	Training Metric/Loss Function	40
4.1.2.8	Starting the Training – Fit One-Cycle	41
4.1.2.9	Hyper Parameter Setting/Tuning	41
4.2	Testing Methodology	42
4.2.1	Evaluation Metrics	43
4.3	Summary	44
5	Results	45
5.1	Resnet34	45
5.2	Resnet50	53
5.3	Comparison with Different Architectures	56
5.4	Analysis	58

5.4.1	Accuracy	58
5.4.2	Computational Time	58
5.4.3	Comparison of Resnet34 and Resnet50 in Terms of Training Cycles	58
5.4.4	Limitation	58
5.5	Summary	59
6	Conclusion and Future work	60
6.1	Conclusion	60
6.2	Future Work	61
A	Installation of Required Frameworks	70
A.1	Installation of Anaconda Python3	70
A.2	Installation of Pytorch, Torchvision , CUDA/ cuDNN Drivers for GPU	71
A.3	Installation of Fastai	73
A.4	Installation and Import of other Required Modules	74
B	<i>Labeling Function</i> for KITTI Dataset	76

List of Figures

1.1	Stereo Triangulation for Estimating 3D Position of Object.	2
1.2	Monocular Depth Estimation using Monocular Cues.	3
1.3	Resnet50 with 50 Convolutional Layers [4].	4
1.4	VGG-16 with 13 Convolutional, 2 Fully Connected Layers and 1 Softmax Layer[7].	5
3.1	Showing Plot of Learning Rate Finder.	25
3.2	U-net Architecture (Example for 32x32 Pixels in the Lowest Reso- lution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. . .	29
3.3	A simple Resnet module with skip connections and a FC layer at the end	30
3.4	A Simple Resnet34 Architecture	31
3.5	A simple Resnet50 Architecture.	31
3.6	Resnet34 Architecture in Fastai’s Dynamic U-Net Configuration . .	32
3.7	Fastai’s Implementation of Dynamic U-Net	33
4.1	Block Diagram of Proposed Methodology.	36
5.1	<i>Resnet34</i> Results (10 epochs) in Dynamic Unet Configuration (with- out sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).	48
5.2	<i>Resnet34</i> Results (20 epochs) in Dynamic Unet Configuration (with- out sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).	49
5.3	<i>Resnet34</i> Results (30 epochs) in Dynamic Unet Configuration (with- out sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).	50
5.4	<i>Resnet34</i> Results (40 epochs) in Dynamic Unet Configuration (with- out sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).	51

5.5	<i>Resnet34</i> Results (50 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).	52
5.6	<i>Resnet50</i> Results (10 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).	55
5.7	<i>Resnet50</i> Results (10 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).	56
A.1	64-Bit Graphical Installer of <i>Anaconda Python3</i>	71
A.2	Conda Install Pytorch Torchvision Cudatoolkit=10.2.	71
A.3	CUDA-Enabled GeForce and Titan Products.	72
A.4	Pytorch Installer for CPU only.	73
A.5	Fastai Installation using Pip as Package Manager.	73
A.6	Navigating Fastai Toolbox for Use in IDE.	74

List of Tables

2.1	Research Gap.	19
3.1	A Comparison of Keras with Fastai.[62]	24
4.1	Initial Hyper Parameters.	41
4.2	Final Hyper Parameters.	42
5.1	Results of Training and Validation: <i>Resnet34</i> in <i>Fastai's</i> Dynamic Unet Configuration trained on KITTI dataset.	46
5.2	Comparison with State of the Art with Results of Resnet34	47
5.3	Comparison with State of the Art with Results of Resnet50	53
5.4	Results of Training and Validation: <i>Resnet50</i> in <i>Fastai's</i> Dynamic Unet Configuration trained on KITTI dataset.	54
5.5	Leader Board for KITTI Data Set for Depth Prediction: Ranking Methods from Top to Bottom Based on the <i>SILog</i> Error Metric. . .	57

Abbreviations

AI	Artificial Intelligence
AbsErrorRel	Relative Absolute Error
ADAM	Adaptive Moment Estimation
ADAS	Advanced Driver-Assistance Systems
ADS	Autonomous Driving System
ANN	Artificial Neural Network
AvgPool2D	2-Dimensional Average Pooling Layer
BatchNorm2D	2-Dimensional Batch Normalization Layer
CNN	Convolutional Neural Network
Conv1D	1-Dimensional Convolution
Conv2D	2-Dimensional Convolution
CPU	Central Processing Unit
CRF	Conditional Random Fields
DCNN	Deep Convolutional Neural Networks
ENet	Efficient Neural Network
FCN	Fully Convolutional Neural Network
FC Layer	Fully Connected layer
GPU	Graphical Processing Unit
ICNR	Initialized to Convolution Neural Network Resize
IMU	Inertial Measuring Unit
ImageNet Stats	ImageNet Dataset's Statistics (Mean and Variance)
Int	Integer
iRMSE	Inverse Root Mean Squared Error
Kwargs	Key Word Arguments

LiDAR	Light Detection And Ranging
Lr	Learning Rate
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MLP	Multi Layer Perceptron
MRF	Markov Random Fields
MSE	Mean Squared Error
OptRange	Optional Range
O/P	Output
PReLU	Parametric ReLU
PNG	Portable Network Graphics
ReLU	Rectified Linear Unit
Resnet	Residual Neural Network
RADAR	Radio Detection and Ranging
RMSE	Root Mean Square Error
SFM	Structure From Motion
SILog	Scale Invariant Logarithmic Error
SequentialEx	Sequential Execution
SONAR	Sound Navigation and Ranging
sqErrorRel	Relative Squared Error
TPU	Tensor Processing Unit
UAV	Un-manned Ariel Vehicle
U-Net	U-Shaped Convolutional Neural Network
VGG	Visual Geometry Group

Chapter 1

Introduction

1.1 Overview

Perception is the ability to comprehend and shape impetuses received from environment in order to understand and behave effectively within it. One of the most significant sources of these impetuses or stimuli for us human beings is our vision system, which comprises of over one million axons in each eye, whose function is to capture light reflected by objects. The processing of such input use billions of neurons in the brain to build the perception of the world we see, in a process called vision. This powerful sense goes beyond simply capturing images, as in the case of cameras and employs a variety of mechanisms in which shape, color, size, movements and distances of objects are perceived.

The recognition of a location in space is vital in almost all daily activities like navigating through a place, avoiding obstacles, catching and throwing objects, reaching for and grasping objects etc. Humans do all these things naturally with their ability to extract 3D structure of the physical world from their 2D retinal images.

Depth perception has been traditionally linked to Stereopsis (i.e. perception of scene using binocular vision). However, there is more information in a 2D image that makes us perceive depth, such as, texture variations and gradients, color/haze

or aerial perspective, defocus, occlusion, linear perspective, relative and familiar size, relative height, and shadows of objects [1], [2]. This so called cue theory, is focused in identifying 3D information (depth of the scene) from 2D images. According to this theory we learn the connections between these cues and the actual depth from our accumulated experience about the spatial relations of objects in the world.

1.2 From Stereo to Monocular Vision Based Depth Estimation

In the field of computer vision, the reconstruction of 3D world from images has mainly focused on creating 3D models of real objects from two or more images. This is accomplished by first finding correspondences in two or more images (taken from two or more view points), and then triangulating matched elements to determine their position in 3D space as shown in the figure below:

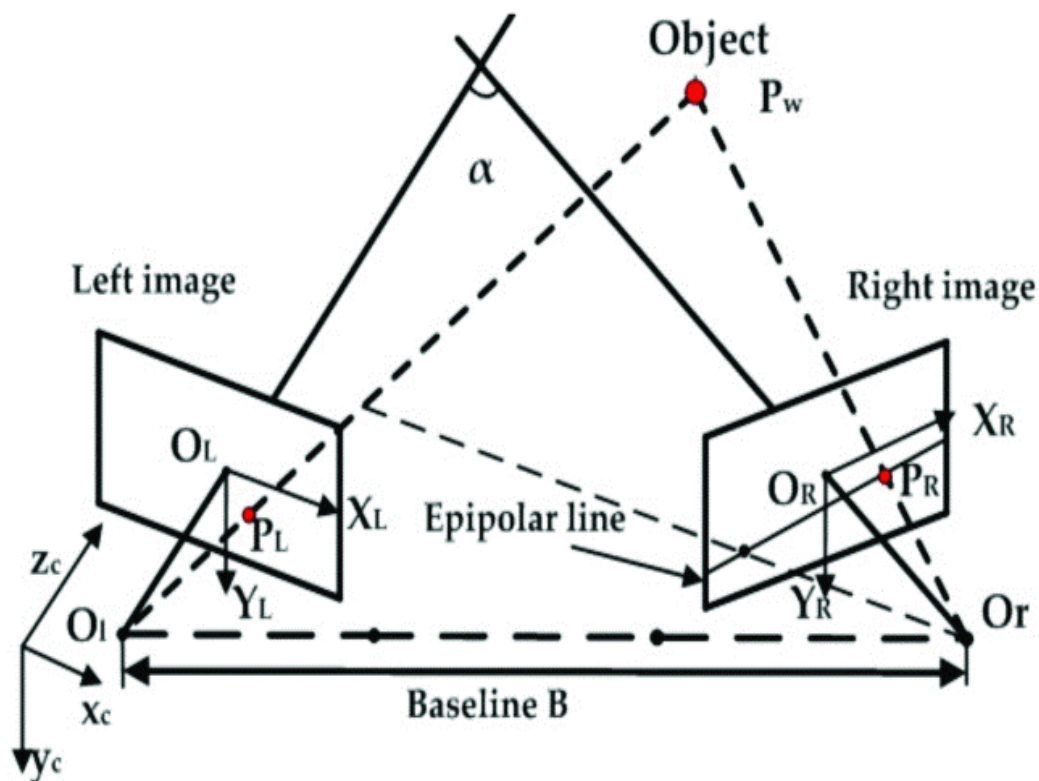


FIGURE 1.1: Stereo Triangulation for Estimating 3D Position of Object.

Recently, the problem of estimating depth from a single image has come in the spot light, where leading researchers from across the globe have come across different solutions to this problem. Current approaches are mainly focused on the design of sophisticated features and levels of reasoning for accurate depth estimation by inferring the 3D structure of the scene as good as possible.

This kind of information is essential in the development of robust guidance systems in most advanced driver assistance systems (ADAS) e.g. adaptive cruise control, collision avoidance, lane departure warning and autonomous driving etc. If accurate enough, estimating absolute depth from images (especially in monocular sense) can avoid their reliance on multiple sensors such as RADARs, LiDARs, ultrasonic sensors and even stereo-camera rigs which are bulkier, power intensive and with high computational cost (in case of stereo cameras).

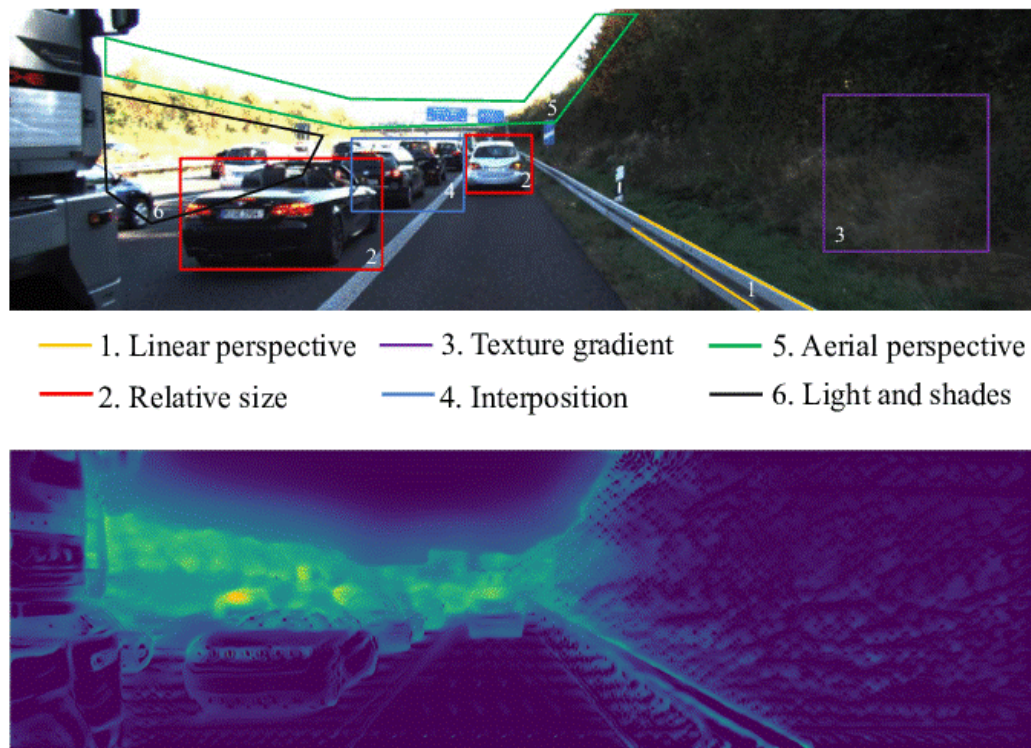


FIGURE 1.2: Monocular Depth Estimation using Monocular Cues.

Monocular depth estimation was for the first time investigated by A. Saxena et.al [1] in 2005, when they used Markov Random Fields (MRFs) to estimate depth from monocular images by training their model on Make 3D dataset. The problem of

Make 3D dataset was that it contained very few images (roughly 400 images) with their ground truth depth images that were taken from LiDAR which at that time had a maximum depth estimating range of only 80 meters. Later on, in 2014, Eigen et.al [3] for the first time investigated monocular depth estimation based on deep convolutional neural networks (DCNNs) on KITTI dataset. Afterwards, there has been a surge of research in this field using DCNNs.

1.3 Deep Convolutions Neural Networks (DCNNs)

DCNNs are a field of Deep Learning, which refers to training Neural Networks with many hidden layers as shown in figure below:

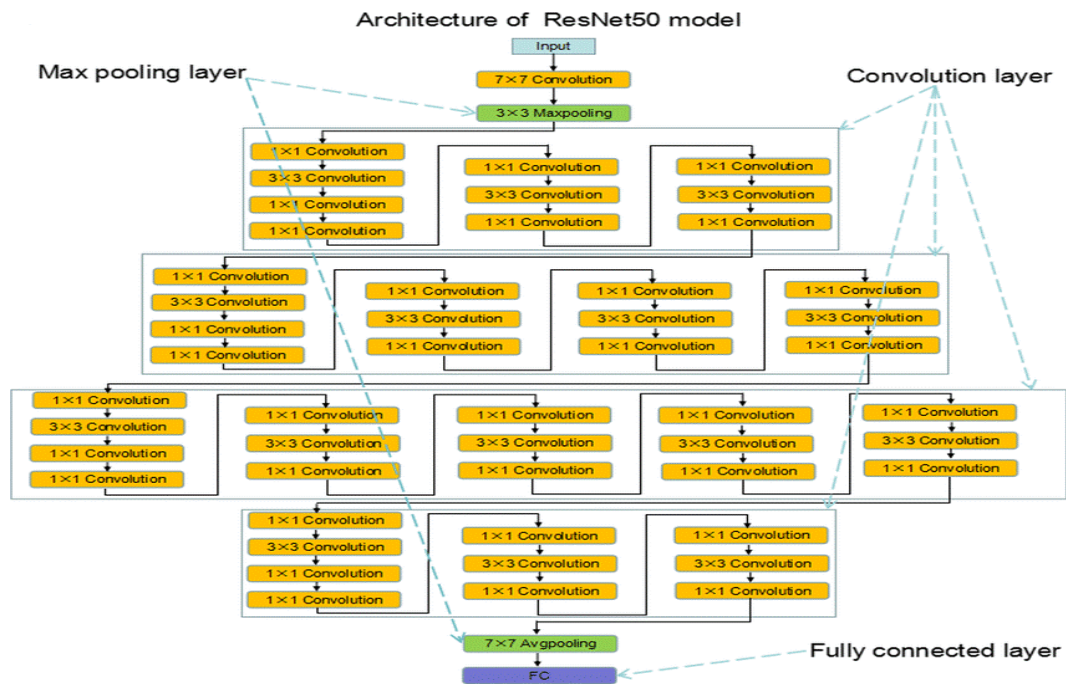


FIGURE 1.3: Resnet50 with 50 Convolutional Layers [4].

Traditionally in computer vision, handcrafted features or filters were used in pre-deep learning era for feature extraction for tasks like classification, semantic segmentation etc. However, with the advent of Deep learning era and access of high end computational capability in the form of GPUs and TPUs, deep learning based

models such as deep convolutional neural networks, or DCNNs, have proven to be a far better approach. Traditionally deep learning is not used with all the fully connected layers for the whole neural network in computer vision tasks. Yann Le Cunn in 1998 for the first time used the convolution operation in neural networks [5] leading way to convolutional neural networks. By using convolutional neural networks it is shown that using the convolution operation, features can be extracted with very less parameters [6]. It is explained in a simple example here. For example, take a simple $224 \times 224 \times 3$ image. For a simple fully connected network with only one hidden layer with 1000 hidden units and 10 outputs, the total parameters for this type of input (excluding biases) will be $224 \times 224 \times 3 \times 1000 + 10,000 = 150,538,000$ i.e. Over 150 million parameters for only single fully connected layered network for this type of input, as mentioned earlier. As discussed in the section, for so many parameters, it's difficult to get that amount of data to prevent a neural network from over fitting. On the other hand, take the famous VGG-16 network [7], which has 16 convolutional layers and takes the same $224 \times 224 \times 3$ input images. VGG-16 is shown in the figure below.

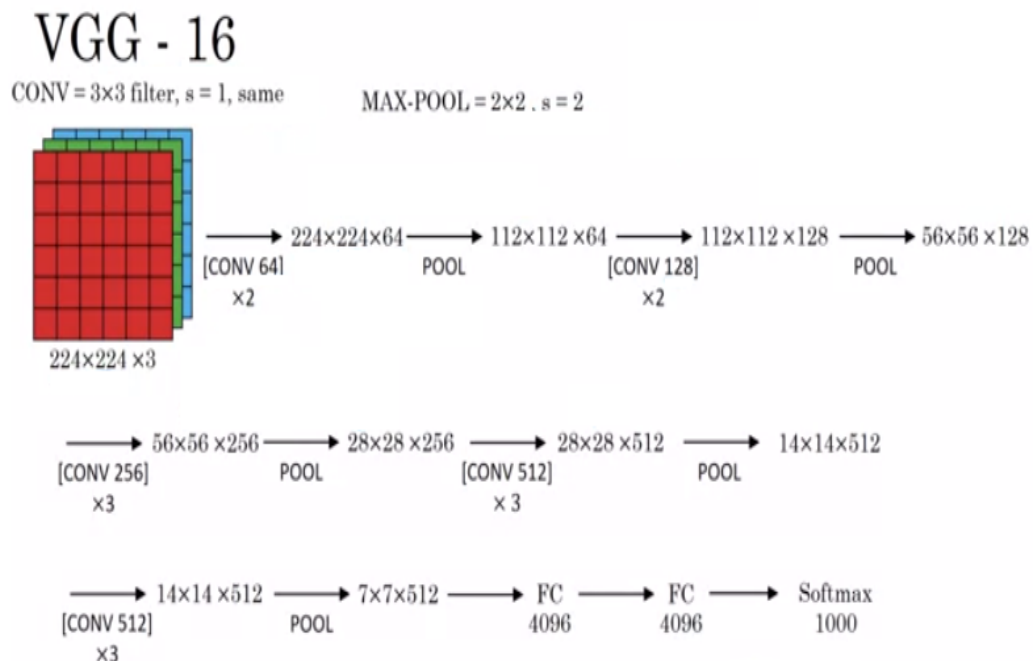


FIGURE 1.4: VGG-16 with 13 Convolutional, 2 Fully Connected Layers and 1 Softmax Layer[7].

Even with last 2 fully connected layers of size 4096 x 4096, and with 1000 Outputs, the network's total parameters are around 138 million, which is about 1.1 times less than a single layered fully connected network for 224 x 224 x 3 images and only 10 outputs. Nowadays, even VGG-16 is considered as a classical network and new fully convolutional architectures [6] have been proposed with way less parameters and hundreds of layers but with superior performance. Furthermore, the CNN architecture is very efficient in interpreting and extracting image information for tasks such as classification [8] or semantic segmentation [9], [10] when compared to other non-CNN architectures. The CNN solves vision based problems by finding low level feature representations for different objects in the image. In convolution neural networks, it has been shown that the first layers detect edges, then some later layers might detect parts of objects and then even later layers may detect parts of complete objects like people's faces, cars etc. [11]. These features are iteratively learned by the CNN during the training process and no hand-crafted engineering is used in their extraction, which stands as a main difference between deep learning methods and pre-deep learning era's computer vision methods. As mentioned in [12], as compared to fully connected networks, convolution operations benefit from the sparse interactions, meaning kernels of much smaller size than the input can be used to extract much less but valuable pixels or features. Another highlight of CNNs is the property of parameter sharing, which basically means that we share parameters for more than one function of a model. Apart from the classification tasks, recently CNNs have also proved their success for other computer vision tasks such as depth estimation [13], [14], [15], [16], [3], [17] and semantic segmentation [18], [19], [6] that both generate pixel-wise predictions. Despite the high performance of CNN architectures, there are still some challenges in this field.

1.4 Research Challenges

Although there have been a surge in past few years in the field of monocular depth perception using deep learning, there are still some issues which need to be

dealt with in order to design algorithms which are suited for practical applications. Some of these challenges are discussed below.

1.4.1 Minimal Equipment and Time for Training

Although nowadays people are training networks with thousands of layers [20] with huge memory requirements that require dozens of GPUs/TPUs to train on, there still is a requirement to explore such architectures and libraries that employ techniques which reduce convergence time and are less memory intensive. Fastai library[21] is one example which was in the headlines when young student developers of Fastai beat giants like google in a benchmark called DAWNbench, from researchers at Stanford [22, 23]. This benchmark uses a common image classification task to track the speed of a deep-learning algorithm per dollar of compute power. Fastai library has proved to be one in this case and needs to be explored in the task at hand.

1.4.2 Light Architectures

In humans we have billions of neurons that learn to accomplish routine tasks. But for a single task a relatively small subset of these neurons are required. Neural networks inspired from the humans tries to fulfill that gap of learning in machines. Recently more architectures [24], [6], [25] have been introduced which produce promising results but are quite memory intensive. There still is a need to explore light architectures to address challenging tasks like monocular depth perception that can be employed on less memory intensive platforms and thus are suited for practical applications.

1.4.3 Real Time Performance on Cost Effective Platforms

For practical systems like ADAS and UAVs, low cost and light weight systems are required which can produce real time performance with less error. Solution of

above mentioned challenges leads to the final challenge in deployment of a system i.e. Real time performance. Algorithms may use smart architectures requiring minimal equipment for training and producing good accuracy but will be slow to produce real time results and thus are not suited for practical applications. Thus there is a need of smart architectures, requiring minimal memory overhead, with less error rate and yet are Fast enough to suit real time applications.

1.4.4 Overcoming the Limitation of Popular Data Sets with Sparse Data for Supervised Learning

Usually popular benchmark data sets like KITTI and NYU Depth are used for monocular depth perception. NYU Depth was developed using Microsoft Kinect platform which use illumination sensors (infrared emitter) to estimate ground truth depth. It is however limited to only indoor scenes as the platform only measures accurate ground truth depth for a few meters and cannot be used for training networks that are to be employed in dynamic natural environments. For dynamic natural environments, KITTI data set is more suited which uses Lidar sensor to estimate ground truth depth of outdoor scenes but is limited to sparse data and misses out information of moving objects [26].

Furthermore, only 40-50 percent of depth information exists in these sparse ground truth images. To overcome this difficulty of acquiring completely filled ground truth data, recently unsupervised methods [27], [16], [28] have been explored to overcome this challenge.

Techniques like Structure From Motion (in which multiple frames of single camera from different viewpoints are used to train the network to learn depth and ego motion of camera) [27], [28] and Learning Reconstruction Loss (to obtain disparity maps by using rectified stereo pair of images at training time) [16] have also been investigated to address this problem.

However, supervised learning utilizing deep learning has proved to produce more accurate results and needs to be explored utilizing Depth Completion of popular data sets like KITTI.

1.5 Research Objective

The research objectives of this thesis are:

- To explore Fully convolutional architectures without any fully connected layers and with less parameters for monocular depth estimation.
- To explore an efficient library like Fastai.
- Utilizing modern best training practices and to achieve higher accuracy than contemporary heavy architectures.

1.6 Contributions

In this research study

- For the first time Fastai's Dynamic U-Net has been used for monocular depth estimation on KITTI Dataset.
- Fastai's Dynamic U-Net architecture is mainly designed for classification tasks with the presence of a 'Sigmoid layer' as the head of the network. In order to regress the depth linearly, the existing head of the network was removed and depth was regressed linearly by the final convolutional layer.
- Resnet34 and Resnet50 architectures were used as backbone architectures in Fastai's Dynamic U-Net, which are then compared with contemporary state of the art heavy architectures.
- Best training practices like Transfer learning, One Cycle training, freezing and unfreezing layers, learning rate finder and discriminative learning rates were used during training, to achieve higher accuracy for monocular depth perception as shown in the results.

1.7 Thesis Organization

In Chapter 1 absolute depth perception using images, monocular depth perception, use of deep learning for monocular depth perception, neural networks, network components and brief challenges in the field of monocular depth perception for practical systems are discussed. Chapter 2 deals with the background and literature review of binocular vision approaches, monocular vision approaches, use of supervised and unsupervised/semi-supervised learning for monocular depth perception. Chapter 3 discusses the popular benchmark dataset and sheds some light on Fastai library. In Chapter 4, Implementation methodology followed is discussed in detail, by first shedding light on the architectures used and then their implementation details. In Chapter 5 results on benchmark dataset are compared with the state of the art and in the end in Chapter 6, conclusion, limitations and future work is presented.

Chapter 2

Literature Review

Depth estimation utilizing camera images has inspired quite a few computer vision researchers for many decades now. In the latter half of the 20th century, inspired by humans, binocular or stereo vision was much used in estimating relative depth of the scenes for predicting their 3D structure. Same techniques were utilized in augmented/virtual reality applications [29], [30]. Stereo vision was also utilized for small range robot navigation [31], [32], [33], [34] and short range surveillance applications [35]. Along with stereo vision, these applications also utilized other techniques like Structure From Motion (SFM)[36], [37], [38], Optical Flow [39], [40], [41], [42], Motion Parallax [43], [44], [45], Active Focus/DeFocus [46], [47], [48] and by using Coded Aperture [49]. Along with their merits, the main shortcomings [28] were Short Range (In case of stereo-vision, where range is mainly dependent on baseline i.e. separation between cameras), Heavy duty equipment involving 2 or more cameras and mainly the High computational cost associated with them, which usually make them incompatible for practical systems in today's scenario where there is a need of more compact and light weight systems with less computational cost (ADAS, ADS, Biometric Scanners, small UAVs etc.). In order to address these issues, Saxena et.al [1] for the first time in 2005 gave the idea of Monocular Depth Estimation i.e. Absolute Depth estimation from single images, using Machine Learning. They used Markov Random Fields (MRF) as the machine learning technique for accomplishing said task. Although at that time

their dataset consisted of very few images and sparse ground truth data, their work laid the foundations of monocular depth perception which is now a popular computer vision and deep learning field. Afterwards, monocular depth estimation was for the first time investigated by Eigen et.al [13] using Convolutional Neural Networks (CNNs) in 2014 on the renowned KITTI and NYU datasets. Ever since, monocular depth perception using deep learning has become the field of research in computer vision community. The traditional Binocular Vision and current Monocular Vision approaches are briefly discussed in next sub sections.

2.1 Binocular Vision Approaches

For more than 50 years, Binocular Vision or Stereo Vision has been extensively studied for extracting 3D information of scene. In humans' binocular vision, both the eyes provide the brain with two different images and the separation between the eyes serve as the baseline.

The visual cortex of the brain first acquire the binocular overlap (the portion of the image information which is in common between images of both the eyes) by processing these images from left and right eyes, and afterwards infer relative depth using binocular disparities (the difference in image formation of objects based on the separation of eyes), as the relative position of objects which are separated in depth from the viewer will be different in both the eyes.

In computer vision, the same phenomena is used to infer depth, i.e. first obtain two images from two different locations/viewpoints, and then process them together to find correspondences (matching similar points of one image in the other) and then infer depth of corresponding points as a function of Focal length of cameras, Baseline (separation between the cameras) and the Disparity between corresponding points, in a process of Triangulation. This is shown by the following equation below:-

$$Depth = \frac{(bxf)}{d} \quad (2.1)$$

where

b = Baseline

f = Focal Length

d = Binocular Disparity

However, stereo matching has its disadvantages and problems dealing with some pixels in the image, due to a couple of main difficulties. Some typical problems are discussed below:-

2.1.1 Ambiguity

In stereo pair of images, a point on object in the physical world is seen in one image but is imperceptible in the other due to the horizontal separation (baseline) between cameras and the change in perspective. Thus, it creates errors in the depth map when an algorithm finds a matching pixel for the point which does not exist in the other image.

2.1.2 Occlusion

In stereo pair of images, a point on object in the physical world is seen in one image but is imperceptible in the other due to the horizontal separation (baseline) between cameras and the change in perspective. Thus, it creates errors in the depth map when an algorithm finds a matching pixel for the point which does not exist in the other image.

2.1.3 Short Range

As evident from Equation 2.1, to obtain correct estimate of absolute depth for distant objects, large baseline is necessary (which is not suited for practical applications like ADAS, ADS and as replacement of Lidars or Sonars), as the system

suffers from constant disparity problem [28], i.e. producing same disparity values for all the distant objects. High resolution cameras can cater this problem to some extent but at the same time they increase the computational cost to such extent that becomes infeasible for practical applications, as discussed below.

2.1.4 High Computational Cost

When dealing with two images, the computational cost automatically doubles as compared to purely monocular case. The processes of developing correspondences, triangulation and the initial synchronization of images are further added to this computational cost.

2.1.5 Hardware Limitations

As discussed above, getting correct estimates of distant objects (e.g. Upto 200m, in the case of autonomous driving), either very large baseline is required or very high resolution cameras (accompanied by high computational cost), rendering the system infeasible for many practical applications.

2.2 Monocular Vision Approaches

Estimating absolute (metric) depth from images is a vital tool in a range of applications, especially in the field of robotics. While ranging sensors, such as LIDAR and structured light sensors, provide superior depth accuracy compared to visual methods, they are not suited for all applications.

LIDAR units are relatively big and expensive and suffer from producing sparse depth map of the scene, while structured light sensors have poor detection range (upto a few meters only). Cameras remain a very cost effective and compact sensor choice for small robotic platforms, such as drones or ground robots. Traditionally, binocular or trinocular camera arrangements are used and depth is estimated as

the inverse of image disparity. On space limited platforms, however, the baseline of the stereo cameras may be too small to provide meaningful disparity measurements beyond a few meters [50], [28].

To solve this problem mainly two types of approaches were followed, One to use learning based methods to obtain meaningful disparity estimates by learning Image reconstruction loss in an un-supervised manner [16] and Second to use Monocular depth estimation based on machine learning [1], [13], [14], [51], [52], [53], [17], [10]. Estimating depth from a single image is an inherently ill posed problem as the same input image can project to multiple plausible depths.

To address this, learning based methods have shown themselves capable of fitting predictive models that exploit the relationship between color images and their corresponding depth. Saxena et.al [1] were the first ones to introduce monocular depth estimation with supervised learning.

They used discriminatively trained Markov Random Field (MRF) models to train depth. In general these models are used as fair approximations, and the depth prediction time is quite inefficient,taking a couple of seconds at least to compute. Other main limitations back then was the use of dataset [54] with very few images and sparse ground truth data.

Eigen et.al [13], [14] were the first ones to use deep learning for monocular depth estimation using a two scale network for predicting a course and fine depth respectively. Ever since then there has been a surge of work in monocular depth estimation using deep convolutional neural networks as briefly explained in the next sub sections.

2.2.1 Supervised Machine Learning

Supervised methods are used to optimize models based on known inputs and their respective ground truth data (Depth maps). Presumably, supervised learning techniques for monocular depth estimation is the most popular approach. Recently, deep convolutional neural networks (DCNNs) have gained huge success in the field of monocular depth perception. Eigen et.al [13], [14] introduced DCNNs for this

task using a two stack network. First stack was used to make a coarse global prediction based on the entire image, and the other stack refined this prediction locally.

A scale-invariant error was also used to help measure depth relations rather than scale, reasoning that a large source of uncertainty for the task comes from the overall scale. Although achieving state of the art results back then on benchmark datasets like KITTI and NYU Depth, more recent methods [51], [16], [53], [17] achieve far better results with improved architectures and loss functions.

DenseDepth is a much more recent method based on the paper High Quality Monocular Depth Estimation via Transfer Learning [55]. DenseDepth relies on transfer learning, a process that uses previous knowledge (in the form of learned parameters by the network) derived from a learning problem (image classification in this case) to help solve another (depth estimation) more efficiently [56] [60].

Transfer learning allowed this method to provide a simpler and modular architecture with similar or even better results than other methods. Their network architecture follows an encoder-decoder structure. The encoder is where the transfer learning occurs, specifically using DenseNet-169 pre-trained on ImageNet [57], which is an image database for image classification and object recognition.

Having a pre-trained encoder section resulted in reduction of validation loss compared to a completely random weights initialization. The decoder section is composed of basic convolutional and transposed convolutional layers.

By now, various approaches, such as combining local predictions, non-parametric scene sampling, through to end-to-end supervised learning [13], [14], [51], [52], [53], [17], [10] have also been explored.

2.2.2 Unsupervised/Semi-Supervised Machine Learning

Supervised monocular depth perception requires vast amounts of corresponding ground truth depth data for training, recording of which is a challenging problem. Goddard et.al [16] introduced the idea of unsupervised monocular depth estimation by replacing the use of explicit ground truth depth data during training with

easier-to-obtain binocular stereo footage.

This is done by posing depth estimation as an image reconstruction problem during training. The intuition given was that, for a calibrated pair of binocular cameras, if a function can be learned that is able to reconstruct one image from the other, then the network has learned something about the 3D shape of the scene that is being imaged.

Specifically, at training time, with access of both the left and right color images from a calibrated stereo pair, captured at the same moment in time, instead of trying to directly predict the depth, an attempt was made to estimate the dense correspondence field that, when applied to the left image, would enable the network to reconstruct the right image. Similarly, the left image can also be reconstructed, given the right one. The model then learns to predict disparity, which is a scalar value per pixel. Given the baseline distance b between the cameras and the camera focal length f , depth can then be trivially recovered from this predicted disparity estimate using Equation 2.1. Zhou et.al [58] presented better results on KITTI dataset by using Unsupervised learning of depth. While Goddard et.al [16] uses stereo image pairs in training, [58] uses monocular video. [58] uses a pose prediction network to warp an image given its predicted depth into the views from the neighboring temporal frames. These reconstructed views are compared to the training video frames with a photometric loss term. The depth and pose estimation networks are trained in an unsupervised manner from this loss. Like in [16], however, the depth network from this work only considers a single view. K. S. Chan [28] has argued that these single-view methods will be unable to resolve the inherent scale ambiguity problem in monocular depth perception. They will primarily learn implicit sizes for different objects in the training scenes to predict depth, essentially over fitting to the training environment. Thus, generalizing poorly to new environments with previously unseen objects. K. S. Chan [28] have shown that one potential way to generalize monocular depth estimation methods is to utilize multiple views of the scene. They have proposed that Structure from motion can resolve the scale ambiguity in the monocular depth estimation task to within a scale factor of camera displacement [59], which is also relatively easy to

measure on some robotic platforms like ground rovers by using wheel encoders or IMUs. Thus, allowing these algorithms to recover absolute depth. Deep learning methods for monocular depth estimation have been shown to generalize better to new scenes by learning structure from motion rather than implicit object sizes [60], but are data intensive to train. Their work focuses on extending monocular depth estimation techniques to incorporate multiple views of the scene and leverage motion cues in an unsupervised manner, thus enabling the estimator to generalize to new environments well without a significant increase in the required inputs for training. Their work finds that multi view networks achieve comparable performance to single view networks and generalize to certain test datasets better than single view networks. Although these methods have enjoyed success, but as discussed in section 2.1, they will tend to suffer one of the fundamental problems of Stereo vision i.e. Occlusion, by matching occluded pixels (in establishing correspondences), which do not exist in other images thus increasing the error rate.

2.3 Research Gap

As discussed earlier, the most investigated technique for visual depth estimation had been stereo vision which had its shortcomings like ambiguity, occlusion, requirement of high computational cost and requirement of large baseline for estimation of depth at longer ranges. In past, some methodologies had been discovered for monocular depth estimation without using any machine learning techniques, but these techniques didn't get much recognition due to being slow and practically infeasible. Moreover, in recent past many techniques have been proposed for monocular depth estimation based on supervised and semi-supervised machine learning algorithms. Currently, the latter has been the focus of research in research community. The research Gap of the literature review discussed in the previous section is shown by the following table:

TABLE 2.1: Research Gap.

Technique	Applicability & Techniques	Limitation and Drawbacks
Stereo Vision	Augmented/Virtual Reality [33], [34]	Ambiguity: find correct correspondence for pixel of same colors Occlusion: Finding a matching pixel for a point which doesn't exist in the image. [32] Short Range: Constant Disparity Problem. [32], [54] High Computational Cost [32] Hardware Limitation [32]
	Robot Navigation [35], [36], [37], [38].	
	Short Range Surveillance [39].	
Monocular Vision-without Learning based Approaches	Optical Flow [43], [44], [45], [46].	Short Range [32]
	Motion Parallax [47], [48], [49].	Poor Accuracy
	Active Focus/Defocus [50], [51], [52].	Real Time Performance
	Coded Aperture [53]	Hardware Complexity
Monocular Vision-Learning based Approaches (Supervised Learning)	MRF [1]	Less Training Data & Sparse Depth Data [5], [68]
	CNNs [1], [3], [5], [6], [14], [55], [56], [57].	Handling High resolution Input Data, Accuracy, Performance / Computational Cost, Training Data, Data Bias and Generalization [68]
	HYBRID [3]	Complex Architecture
Monocular Vision-Learning based Approaches (Semi-Supervised" Learning)	Left Right Consistency [62]	Primarily Learn Implicit Sizes of Objects [32] Poor Generalization Ability [32]
	Depth & Ego Motion from Video [16] .	Primarily Learn Implicit Sizes of Objects [32] Poor Generalization Ability [32] Prone to Two Stages of Error

2.4 Problem Statement

In this research study, fully convolutional architectures with lesser parameters are explored utilizing Transfer learning, along with the use of an efficient library (Fas-tai) to address the challenges of Training with less memory overhead and obtaining higher accuracy than contemporary heavy architectures with more parameters.

2.5 Summary

Literature review was discussed in this chapter, highlighting the inherent shortcomings of stereo vision, the advent of monocular depth estimation techniques and recent surge in research on monocular depth estimation using deep convolution networks. Research gap was presented to show the shortcomings of existing and previous schemes.

In the coming chapter, the dataset used in this research work, the use of best training practices in an efficient framework and the architecture used for training is discussed

Chapter 3

Dataset, Libraries and Architecture Used

This chapter sheds light on the dataset used in this research work along with the framework utilized for training and finally the architecture used for training on dataset.

Section 3.1 deals with the description of KITTI dataset which was used in this research study, by discussing the size of training and validation/test set, the locations (outdoor environments) where the images/scenes were captured, size and format of images in the dataset and how they were used for training.

Section 3.2 and 3.3 discusses the Fastai library, which is the framework used for training in this research work.

A comparison of Fastai with keras is also being made to show the effectiveness of this framework.

It also highlights the use of best training practices undertaken by deep learning practitioners which are incorporated in Fastai by default.

Finally section 3.4 discusses the U-Net architecture used for training on KITTI dataset for monocular depth perception. It also highlights the differences between the initially proposed U-Net architecture and Fastai's Dynamic U-Net architecture and how it was tailored to suit the depth regression task.

3.1 Dataset

For the task of single image depth prediction, the model was trained on the renowned challenging KITTI vision Benchmark suite. The KITTI dataset [26] includes images and their corresponding ground truth depths of outdoor scenes taken at different intervals and in different outdoor environments. In the collection of this dataset a Volkswagen hatchback was mounted with 2 stereo camera rigs (one stereo pair of cameras was used to capture color images and the other for capturing grayscale images) and a velodyne laser scanner (for capturing absolute of the scene).

The dataset consists of left and right images of both the color and grayscale stereo pair of cameras, as well as their ground truth depth data obtained from Lidar scans. The KITTI vision Benchmark suite also consists of annotated depth maps for semantic segmentation as well as IMU data (given as text files for each image) for visual odometry. This thesis is restricted to supervised depth prediction task, so only images with their ground truth depth data was used. Description of this dataset is explained in the following sub-section.

3.1.1 KITTI

The depth prediction evaluation benchmark of KITTI dataset consists of over 93 thousand images of outdoor scenes with their corresponding ground truth depth maps. All the 134 scenes from the “city,” “residential,” “road”, “campus” and “person” categories of the raw data were used in this thesis. The RGB images are originally 1224 x 368, and down sampled to 224 x 224 to form the network inputs. Roughly 86 thousand images are used for training and 14224 images for validation. Evaluation metrics are also provided for the purpose of single image depth prediction for KITTI dataset. The evaluation table provided by the dataset [26] ranks all methods according to square root of scale invariant logarithmic error (SILog).

Ground truth depth maps in KITTI dataset were acquired by accumulating 3D

point clouds from a 360 degree Velodyne HDL-64 Laser scanner and a consistency check using stereo camera pairs. Depth maps (annotated and raw Velodyne scans) are saved as uint16 PNG images. A 0 value indicates an invalid pixel (i.e., no ground truth exists). For training, the depth for a pixel was computed in meters by converting the uint16 values to uint8 by dividing it by 256, as shown in equation 3.1:

$$disp(u, v) = \frac{((float)I(u, v))}{256.0}; \quad (3.1)$$

$$valid(u, v) = I(u, v) > 0; \quad (3.2)$$

Where; $I(u,v)$ is the ground truth depth image and u,v correspond to pixel indices. Both left and right RGB cameras were used during training, but were treated as un associated shots. The training set included almost 48K unique images and using left and right images made a total of 85898 images in training set.

3.2 Fastai Library and Pytorch

Among other frameworks in Python for deep learning like Tensorflow, caffe' and keras, Pytorch is also a deep learning framework of Python language, which has been developed and maintained by Facebook researchers. In this thesis we have trained Pytorch models using the Fastai library. The Fastai library uses best training practices employed by competition winners and thus the training is simplified. It provides “out of the box” solutions for specifically four applications namely tabular, vision, text and collab (collaborative filtering) models. It came to lime-light when according to MIT Tech review , Students from Fast.ai, created an AI algorithm that outperformed code from Google’s researchers, as measured using a benchmark called DAWNbench, from researchers at Stanford. Some very interesting functionalities like Leslie Smith’s [61] Learning Rate Finder and One Cycle

Training are also provided in this library which at the moment is not provided in the contemporary libraries like Keras, Caffe etc. As here we are only interested in the vision application, so we will only discuss support provided by Fastai for vision i.e. in Fastai.vision module.

A simple example in terms of Fastai's performance is depicted by the table 3.1. Table 3.1, shows Fastai's performance in terms of speed and accuracy, on Oxford-iiit-dataset (dataset for classification of Cats and Dogs breeds) [62], with respect to Keras, which right now is the only other language which makes deep learning easy to use.

TABLE 3.1: A Comparison of Keras with Fastai.[62]

Performance metrics	Fastai(Resnet34)	Fastai(Resnet50)	Keras
Line of Codes(excluding imports)	5	5	31
Stage 1 Error	0.70%	0.65%	2.05%
Stage 2 Error	0.50%	0.50%	0.80%
Test Time Augmentation Error	0.30%	0.40%	N/A
Stage 1 Time	4:56	9:30	8:30
Stage 2 Time	6:44	12:48	17:38

It is perceptible from table 3.1 that Fastai outperforms Keras in all respects i.e. accuracy, time and lines of code. e.g. In terms of Lines of code, Fastai requires only 5 lines, whereas, keras requires 31 lines of code, which means that in Keras, you need to set much of the hyper-parameters by yourself, where on the other hand, Fastai sets these hyper parameters according to the best modern practices. This fact is highlighted in the increased accuracy of results for Stage-1 and Stage-2 error, as shown in the above figure.

Similarly use of best modern practices in Fastai also result in quick convergence time as also highlighted in table 3.1. Time required for Stage-2 training for Fastai is almost 3 times less than Keras.

3.3 Best Training Practices in Fastai

Few best modern practices are used by Fastai during training. Some of them are described below in the sub-sections:

3.3.1 One Cycle Training – *Freezing/Unfreezing Layers*

One-Cycle Training was introduced by [61], and has been used in Fastai library extensively. In One-Cycle training, we train the network with some learning rate (usually high, default is 0.003).

3.3.2 Learning Rate Finder and Discriminative Learning Rates

One of the exciting technique presented in Fastai is the use of Learning Rate Finder. It is used to select the best learning for the model by ‘competition winners’ (who spent a lot of time in hyper parameter tuning to achieve the lowest error). It is done in Fastai by first providing different learning rates during training of some batches of the training set. Resultantly, the losses are plotted for these learning rates. The point from where the losses shoot (almost exponentially), is selected and we move ten step backwards, i.e. if the losses shoot at $1e-2$, then we will move to $1e-3$. This is selected as the upper limit of learning. This is shown in the figure below, taken from training KITTI Dataset.

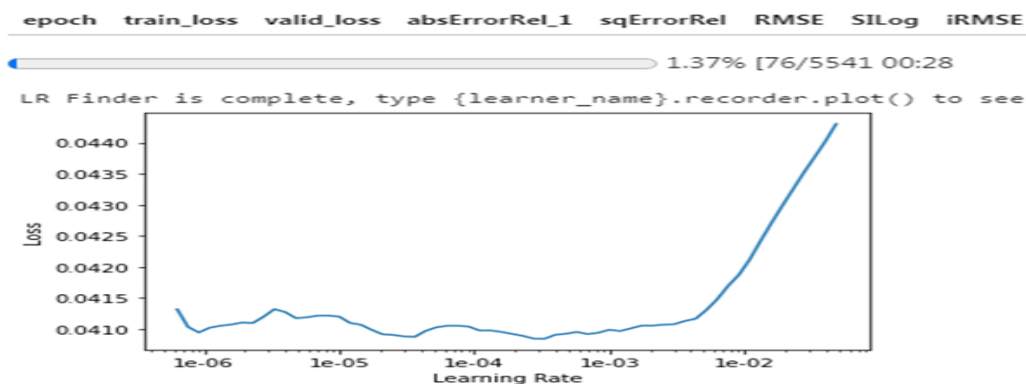


FIGURE 3.1: Showing Plot of Learning Rate Finder.

As shown in the figure 3.1, a red box is marked on the region of interest i.e. the value of learning rate from where the losses shoot abruptly ($1e-2.5$) to ten step backwards ($1e-3.5$), as marked by red circles and intersection lines. It can also be seen in the figure above from the yellow highlighted box, that only 76 batches (of size 32 images) were used for subject purpose. As discussed above, this value of learning rate ($1e-3.5$ in the above example), is taken as the upper bound of learning rate. The lower bound is selected as the same learning rate used for One-Cycle training, the default is 0.003. From this we come to Discriminative Learning Rates.

The Discriminative Learning Rates are defined as providing different learning rates to different sets of layers of model. This is done by slicing the learning rates between Upper and Lower bound, provided in the fit function for training the model. The number of slices (5, 10 etc.) depends on the depth of the model and is done automatically by the Fastai library. The intuition behind this scheme is that usually the lower layers require less learning rate value as compared to upper layers of the network, especially in the context of Transfer learning, where the lower layers are assumed to be already near a global minima.

In this way, Fastai library tries to provide optimal learning rates to different sets of layers of the network. Thus Fastai has shown to achieve quick convergence and better accuracy as compared to contemporary frameworks, as already shown in Table 3.1.

3.4 U-Net Architecture

The typical use of convolutional networks had mostly been on classification tasks, where the output to an image is a single or few class labels. However, in many visual tasks, such as image segmentation, the desired output should include localization, i.e., a class label is supposed to be assigned to each pixel. In case of monocular depth estimation considered in this research study, this pixel-wise localization can be seen as pixel-wise depth regression (where the final layer will perform pixel-wise regression instead of classification). Thus models suited for

such pixel-wise localization can better fit to this application as well.

Fully Convolutional Networks (FCN) architecture for semantic segmentation (and spatially dense prediction tasks) were initially proposed by Long et. al [6]. They achieved state of the art segmentation of PASCAL VOC and NYUDepth V2. Their architecture was further improved in U-Net architecture by Ronneberger et.al [24].The main idea in Long et. al's [6] work is to supplement a usual contracting network (the encoder part) by successive layers (the decoder part), where pooling operators get replaced with up sampling operators.

Hence, these layers in decoder part of the network increase the resolution of the output. In order to localize, high resolution features from the contracting path are combined with the up sampled output i.e. by concatenating the high resolution feature maps (activation maps after passing through the activation layer) in the encoder part of network, with the up sampled output in the decoder part. Afterwards, in order to let the model learn a more precise output based on this information, a convolution layer is applied (to this up sampled output which was combined with the high resolution feature maps of encoder part).One important difference in U-Net architecture from the FCN architecture, is that in U-Net architecture, in the up sampling part there are a large number of feature channels. These large number of feature channels allow the U-Net architecture to propagate context information to higher resolution layers. As a consequence,the expansive path (decoder part of network) is more or less symmetric to the contracting path (encoder part of network), and yields a u-shaped architecture. The network does not have any fully connected layers.

3.4.1 Encoder-Decoder Architecture

The network architecture presented in [24], is illustrated in Figure 3.2. It is an encoder-decoder type of architecture, which consists of a contracting path (left side) and an expansive path (right side). The encoder part follows the typical architecture of a convolutional network. In Fastai a slightly different U-Net architecture is used, namely Dynamic U-Net.The original U-Net architecture mentioned

in [24], consisted the application of un padded 3 x 3 convolutions applied twice, each followed by a ReLU activation, followed by down sampling step in which a 2 x 2 max pooling operation with stride 2 is used. A total of four down sampling steps were proposed in [24] and number of feature channels are doubled at each down sampling step. In the decoder part each step consists of an up sampling of the feature map followed by a 2 x 2 convolution (up-convolution or Transposed convolution) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the encoder part, and two 3x3 convolutions, each followed by a ReLU. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes/Outputs. In a typical network shown in the paper [24], with input of 572 x 572, a total of 23 convolutional layers were used. However, it is worth mentioning here that the backbone architecture in the contraction part of U-Net architecture can be any architecture, depending upon the input size of images. For example, Resnet18, Resnet34 and Resnet50 can be easily used for input size of 224 x 224, but when using Resnet101 as backbone architecture for input image size of 224 x 224, the contraction part is extended beyond 1-pixel in order fit-in the desired architecture (While implementing in Fastai, an error pops up for the same reason as network architecture doesn't fit the input size). The Fastai's dynamic U-Net implementation can be seen in Figure 3.6 and Figure 3.7, using backbone architectures of Resnet34 and Resnet50, respectively, in the encoder part. The differences in [24] and Fastai's dynamic U-Net implementation can be seen when comparing Figure 4.1 with Figures 3.6 and 3.7. The first difference is the filter size and number of convolution layers at the very start before the max-pooling operation. In Fastai's Dynamic U-Net, only one convolution layer is applied with a filter size of 7x7, stride 2 and a padding of 3 x 3, as compared to two un padded 3x3 convolutions applied in [24], before the max-pooling operation. Secondly, in [24] all the convolutions applied are un padded convolutions as perceptible in figure 3.2, from the reduced spatial size of feature maps after every convolution layer, whereas, in Fastai's dynamic U-Net, mostly all the convolutions applied are padded convolutions. The 3rd prominent difference is the application of Batch norm layer after almost every convolution

layer in Fastai’s dynamic U-Net implementation, which is absent in the [24]. The 4th main difference is in the decoder part, where up sampling is done using pixel shuffle ICNR with a factor of 2. And the 5th difference is in the architecture design of the decoder part as can be seen in figures 3.6 and 3.7.

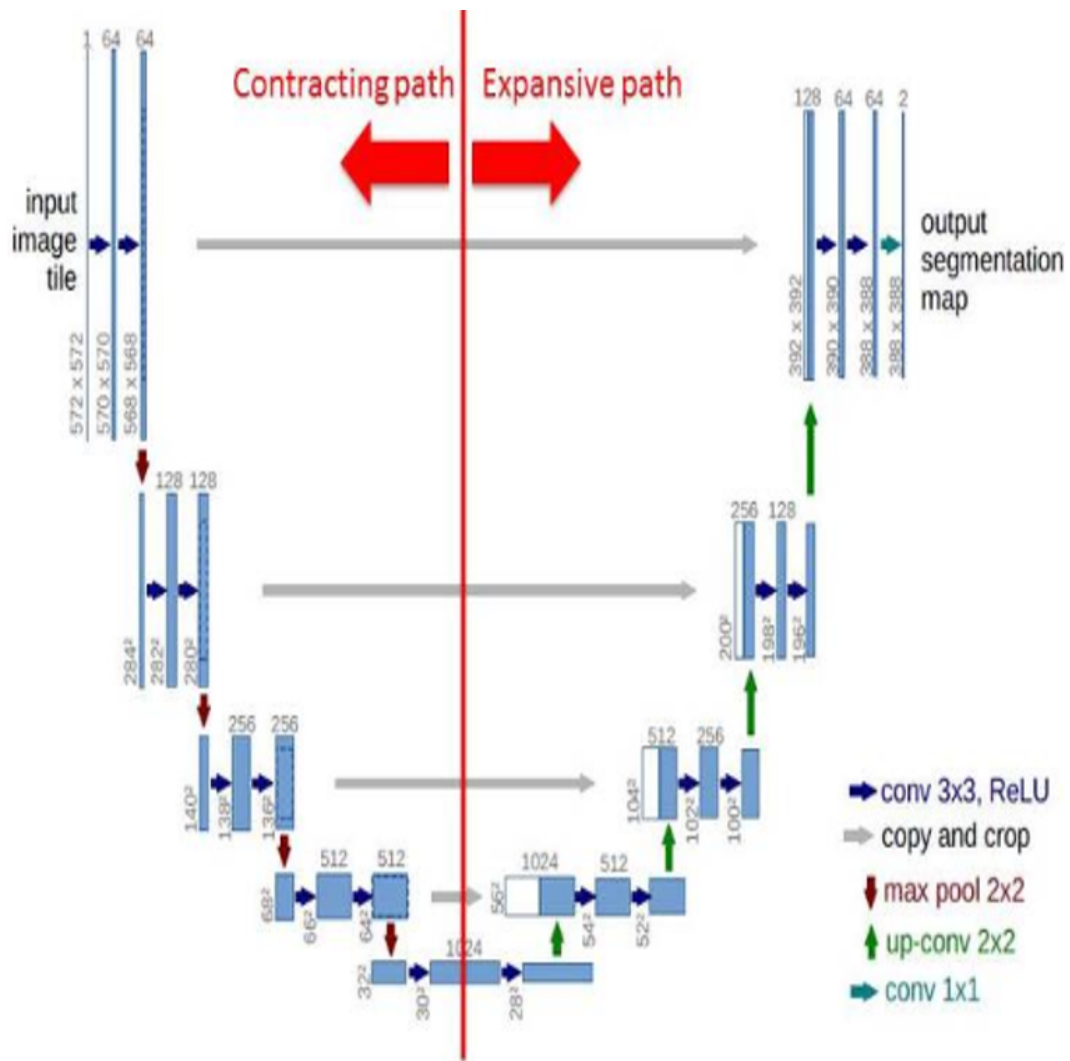


FIGURE 3.2: U-net Architecture (Example for 32x32 Pixels in the Lowest Resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

The following sections shed some light on the Resnet architectures used in our application. U-Net models for each of the Resnet34 and Resnet50 architectures are shown diagrammatically in sections 4.1.2 and 4.1.3, respectively.

3.4.2 Resnet-34

A Resnet module shown in figure 3.3, consists of skip connections and has proven to be a ground breaking architecture when training deep neural networks. It caters for the problem of vanishing and exploding gradients in deep neural networks using these feed forward or skip connections. Resnet34 and Resnet50 architectures (without U-Net configuration) are shown in figure 3.4 and 3.5, respectively. Whereas, figure 3.6, shows the Resnet34 architecture in U-Net configuration, in which the encoder part is a pre-trained model on Imagenet. The total trainable parameters for resnet34 in dynamic U-Net configuration are 41,405,588 (approximately 41.4 million). As highlighted in [5], for depth regression, the depth should be 'linearly' regressed by the network. However, the Fastai's Dynamic U-Net architecture (with any backbone architecture like Resnet18, Resnet34, Resnet50, Resnet101 e.t.c) is mainly designed for classification tasks with a final 'sigmoid layer' as the head of the network. To suit the depth regression task at hand, this last sigmoid layer was removed and depth was regressed linearly by the final convolutional layer.

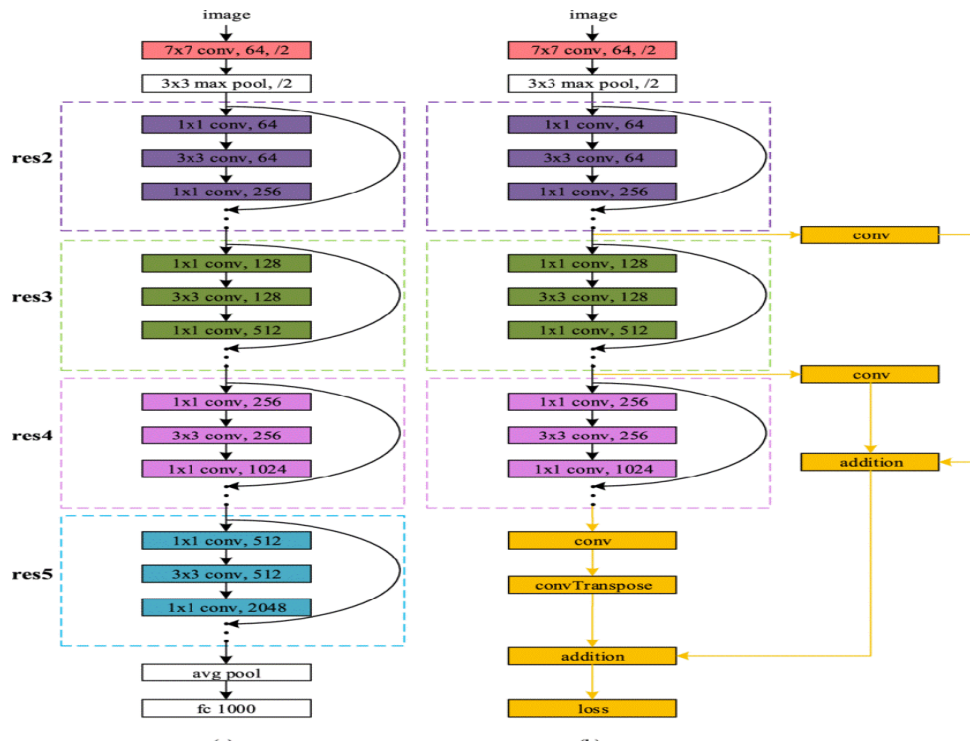


FIGURE 3.3: A simple Resnet module with skip connections and a FC layer at the end

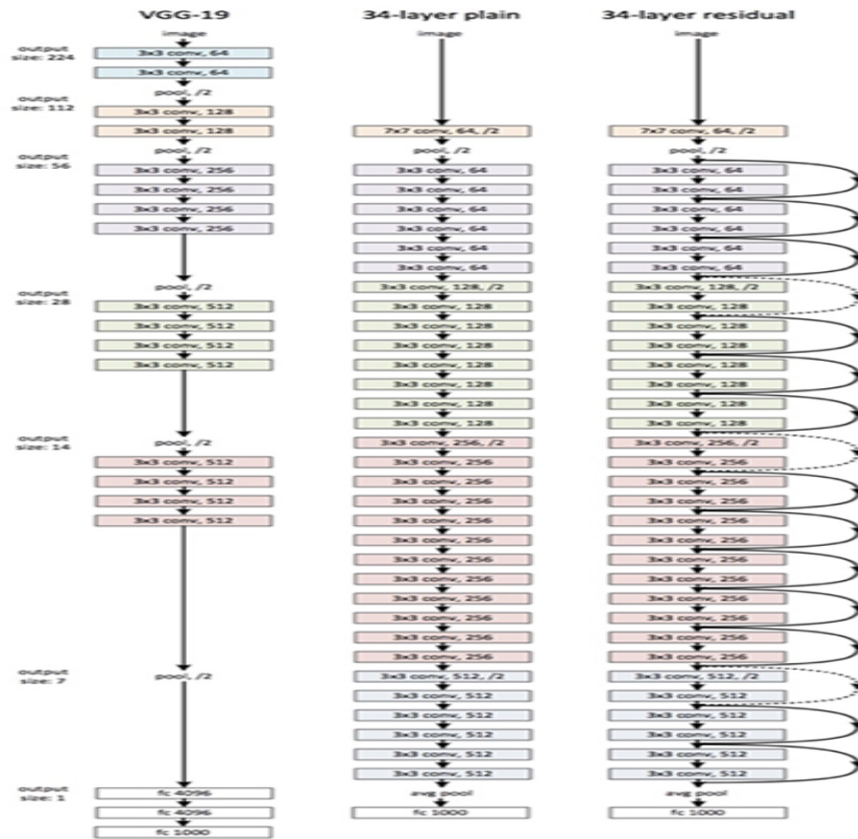


FIGURE 3.4: A Simple Resnet34 Architecture

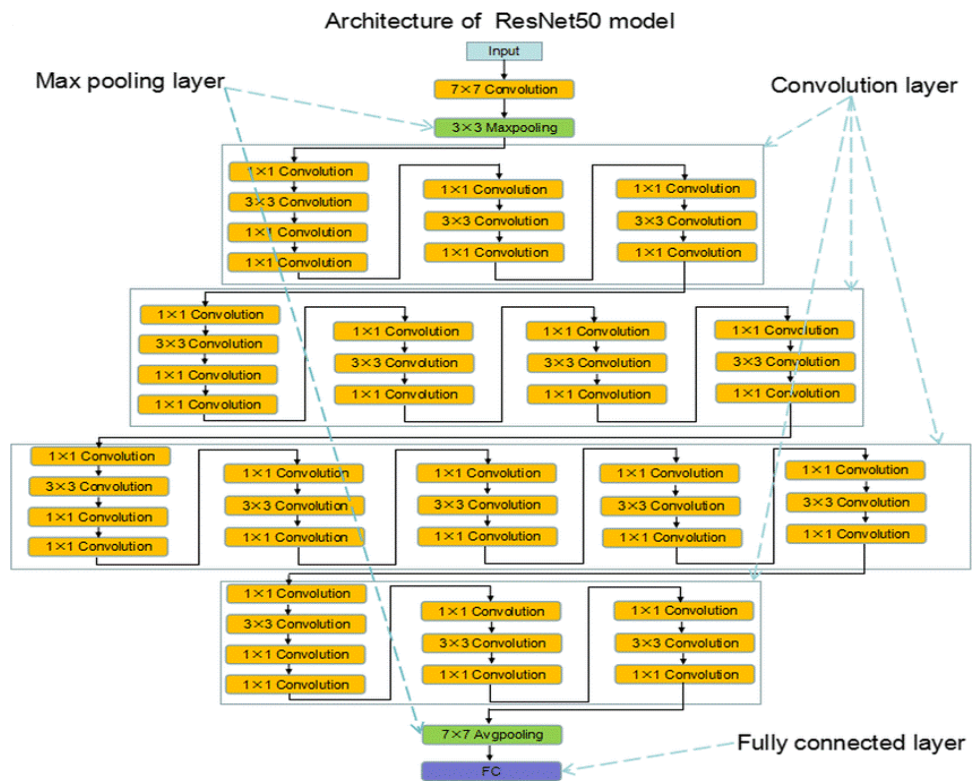


FIGURE 3.5: A simple Resnet50 Architecture.

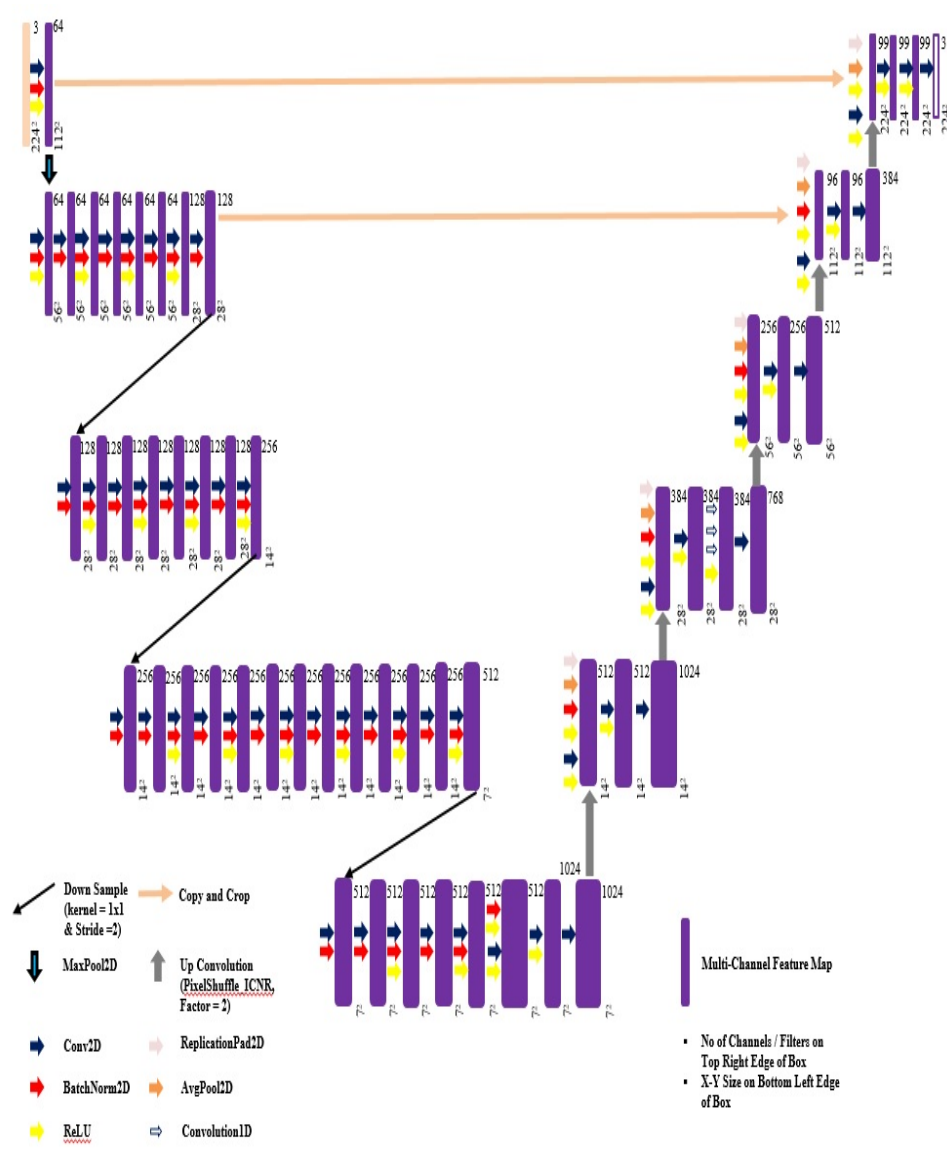


FIGURE 3.6: Resnet34 Architecture in Fastai’s Dynamic U-Net Configuration

3.4.3 Resnet-50

Resnet50 was also used for training the KITTI Dataset. The Resnet50 architecture in the U-Net configuration is shown in figure 3.6. As highlighted in Chapter 5 (Results), the Resnet50 architecture produced superior qualitative results as compared to Resnet34. The total trainable parameters for resnet50 in dynamic U-Net configuration are 342,019,412 (approximately 342.01 million). As discussed in 4.1.2, the last sigmoid layer of Fastai’s Dynamic U-Net was removed and depth was regressed linearly by the final convolutional layer.

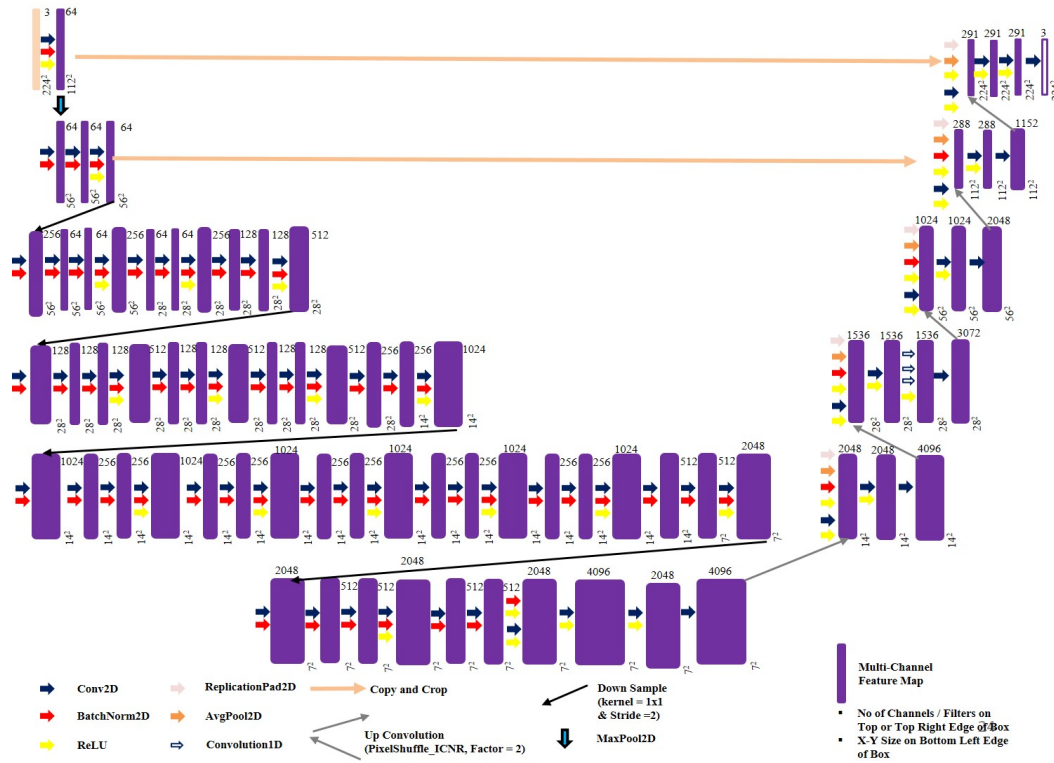


FIGURE 3.7: Fastai’s Implementation of Dynamic U-Net

3.4.4 Fastai’s Implementation of Dynamic U-Net

The U-Net learner was built on using the data bunch and backbone architecture (Resnet34, Resnet50). The model used is a Dynamic U-Net. In Fastai, the Dynamic U-Net module builds a dynamic U-Net from pre-trained backbone architecture. The main differences between Fastai’s Dynamic U-Net and the original U-Net proposed in [24] have already been described in section 4.1.1.

```

Class Dynamic U-Net DynamicU – Net(encoder : Module, n_classes :
int, img_size : Tuple[int, int] = (256, 256), blur : bool = False, blur_final =
True, self_attention : bool = False, y_range : OptRange = None, last_cross :
bool = True, bottle : bool = False, **kwargs) :: PrePostInitMeta :: SequentialEx

```

This U-Net architecture has two parts – One is the encoder which can be a pre-trained model and second is the decoder part which will have the final output of user defined classes (in depth estimation’s case, an output image of reasonable resolution as compared to input image. ‘blur’ is used to avoid checkerboard artifacts

at each layer, 'blur-final' is specific to the last layer. 'self-attention' determines if we use a self attention layer at the third block before the end. If 'y-range' is passed, the last activations go through a sigmoid re scaled to that range. 'last-cross' determines if we use a cross-connection with the direct input of the model, and in this case bottle flags if we use a bottleneck or not for that skip connection.

3.5 Summary

In chapter 3, the peculiarities of KITTI dataset were discussed along with the description of how to use this dataset during the training process. The benefits of using an efficient framework/library were also discussed, which incorporates the modern best training practices like One-Cycle training, learning rate finder and use of discriminative learning rates. Finally the U-Net architecture is discussed and how it suits the pixel-wise depth regression task. The use of Fastai's Dynamic U-Net architecture, its difference with the initially proposed U-Net architecture and how it was tailored to suit depth regression task was also discussed.

The coming chapter discusses the proposed training methodology by highlighting the setting up of KITTI dataset, its cleaning and labelling process, the formation of Image data bunch, the data augmentations and normalization performed on dataset, the initial and final Hyper-parameters setting and tuning. It also discusses the loss function used for backpropagation and testing/evaluation metrics during the testing phase.

Chapter 4

Implementation Methodology

This Chapter deals with the implementation methodology, starting from the network architecture utilized in this work to network training and testing methodology implemented. In section 4.1, the network training with Sparse ground truth data is discussed along with the Training methodology (including data augmentation). Section 4.2, finally discusses the Testing methodology and evaluation metrics used.

4.1 Proposed Methodology

How the model was trained from scratch on KITTI dataset is discussed in this section. Starting from organizing the dataset for use and cleaning of data, training in Fastai is discussed step-wise moving from creation of an item list, labelling function used for KITTI dataset, creation of Image Data Bunch and creation of U-Net-learner, showing how data augmentation and transfer learning is done in Fastai.

At the end of this section, the training metric/loss function and Hyper-parameter settings used for the application is discussed, followed by how finally the training process starts by using One-Cycle training and how Hyper-parameters tuning is performed. The block Diagram of this training methodology is shown below:

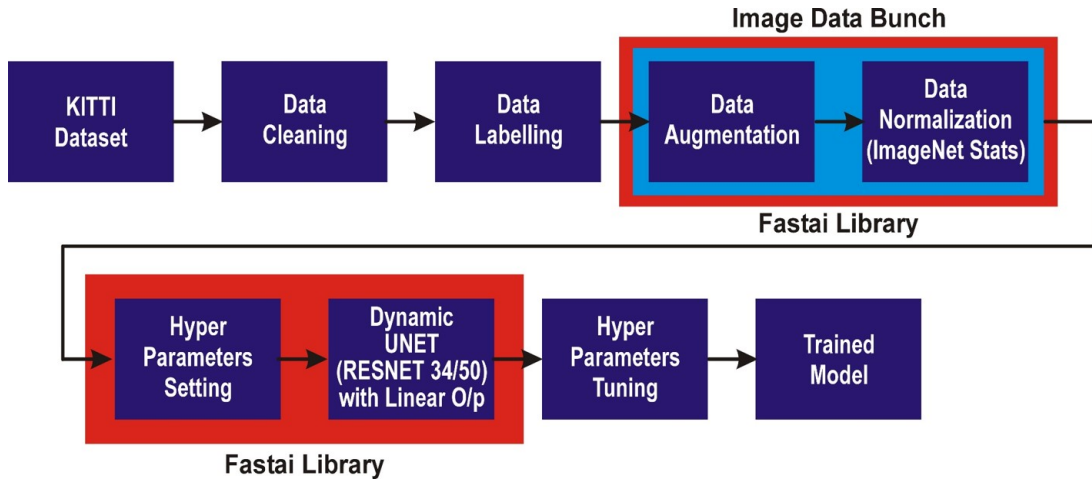


FIGURE 4.1: Block Diagram of Proposed Methodology.

4.1.1 Organizing the Dataset/Cleaning of Data

Organizing the dataset for use is one of the foremost tasks before any model can be trained on it. The Eigen split [13], [14] for KITTI dataset was not used in this thesis, in which only 56 scenes from the “city”, “residential”, and “road” categories of the raw data were used (28 scenes were used for training and 28 scenes were for validation/testing). Instead around 85k images of KITTI dataset have been used for training, which come from 134 scenes of all 5 categories of the raw data namely: “city”, “residential”, “campus”, “road” and “person”. However, the KITTI dataset also provides the same number of grayscale images for these 5 categories, which were not in this thesis. Both the left and right images of the stereo pair of images of the scenes were used. The same test set as provided by KITTI Benchmark suite for depth prediction have been used in this thesis, which consists of around 14224 images. The image IDs of depth images start from 5 (i.e. 0000000005.png), as the depth estimation algorithm (from velodyne laser scanner) required accumulation of 11 frames (± 5 around the current frame). So, depth images for the first 5 and the last 5 images of the stereo pair of RGB images do not exist in the raw data. In order to align/match the RGB training images with their corresponding ground truth images (so that proper labelling of each training image is done), these first 5 and last 5 images of both the left and right cameras were manually deleted in all the 134 scenes. Otherwise, an error would pop up while labelling the item list.

4.1.2 Training in Fastai

Training of deep CNNs in Fastai simply means that you have a dataset of images in which both the Training and test sets along with their ground truth labels (ground truth depth images in this case) are present. In the start, a deep CNN architecture is selected. Then an objective function is defined which is also called a loss function or a training metric which calculates error / loss in the forward pass i.e. all the training images are put to the network as input (in batches) and the output thus obtained is compared with their corresponding ground truth labels (ground truth depth images in this case), as per the loss function criteria. The error calculated in the forward pass is then back propagated through the network in the backward pass and the training process continues until the number of epochs defined has reached. But by looking at this training process step-wise in Fastai, it simply means that first make an Image Data Bunch i.e. simply make a Data object of all the training and validation/test set images along with their ground truth labels. This usually consists of two steps. In the 1st step an item-list is created, which is simply a collection of all the training and test set images. Then in the 2nd step, these training and test set images are assigned to their corresponding ground truth labels in a step called labelling by using a labelling function. During the creation of Image Data Bunch, the data augmentation (transformations) is also defined which is required to be performed in the training process, along with the input size of images to be put to the network (and some other fine details). After creation of Image Data Bunch, the network architecture along with all the hyper-parameters and the loss function in a learner object (cnn-learner or U-Net-learner) is defined, followed by the commencement of training process. All these processes are explained step-wise in the following sub-sections.

4.1.2.1 Creation of Item List

Suppose that the dataset is stored in a folder named 'KITTI Dataset', the address of which is stored in a path variable "path". This main folder (named as 'KITTI Dataset') then further contains images in different subfolders for training

and validation (Training set images are present in the folder named ‘training’ and test set images are present in the folder named ‘validation’). In Fastai, in order to make an ‘item list’ of images present in the main folder of KITTI Dataset, following line of code will be used. `item list = ImageItemList.from folder(path, extensions=['.png']).split by folder(train= ‘training’, valid= ‘validation’)`

4.1.2.2 Labelling Function for KITTI Dataset

As discussed, once an item list has been created, the next step is to label this item list i.e., label the images present in the item list. This labelling process is simply a step in which all the Training and test set images will be assigned with their corresponding ground truth labels (which in our case will be Depth Images), by using a labelling function. This is simply done by the following line of code: `itemlist-labeled = itemlist.label-from-func(Labelling-Function);`

4.1.2.3 Creation of Image Data Bunch

Once the labelling process is finished, we are finally ready to make the Image Data Bunch, in which all the transformations required for data augmentation, size of the input images, Batch size and normalization techniques are defined for the already labelled item list, (defined in the previous section as “itemlist-labeled”). This is accomplished by the following lines of code: `bs = Batch-Size (e.g., 64, 32, 16, 8, 4, 2, 1 etc.) data-train = (itemlist-labeled.transform(tfm-y = True, size=(224,224)).databunch(bs=bs).normalize(imagenet-stats))`

4.1.2.4 Data Augmentation

Data augmentation was already discussed in section 3.2.5. While creation of Image Data Bunch, we specify all the transformations / data augmentations, that need to be performed on our data set. For KITTI dataset, 4 transformations were performed, Scaling, Flips, Color and Translation. As argued by Eigen et.al [13],

[14], image scaling and translation do not preserve the world-space geometry of the scene. This is easily corrected in the case of scaling by dividing the depth values by the scale 's' (making the image 's' times larger effectively moves the camera 's' times closer). Although translations are not easily fixed (they effectively change the camera to be incompatible with the depth values), they found that the extra data these translations provided benefited the network. The other transforms, flips and in-plane rotation (performed on NYU V.2 dataset by Eigen et.al [13], [14]), are geometry-preserving. Following parameters were used for data augmentation:

- Scale: Input and target images are scaled by $s \in [1, 1.05]$, and the depths are divided by s.
- Translation: Input and target are randomly cropped to the sizes of 224 x 224.
- Color: Input values are multiplied globally by a random RGB value $c \in [0.9, 1.1]$, i.e. between 0.8 and 1.2.
- Flips: Input and target are horizontally flipped with 0.5 probability.

These are the only transformations considered suitable by many researchers [13,29] for monocular depth estimation task.

4.1.2.5 Creation of U-Net-Learner/Model for Training

For full image pixel-wise regression, a U-Net-learner will be needed, which creates a U-Net architecture from any backbone pre-trained architecture. Here, the training methodology followed is shown step-wise. Creation of U-Net-learner is simply done by using following lines of code in our case: `arch = models.Resnet34` `def create-gen-learner(): return U-Net-learner(data=train, arch, wd=wd, blur=True, norm-type=NormType.Weight, self-attention=True, y-range=y-range, loss-func=loss-gen, metrics=[absErrorRel-1, sqErrorRel, RMSE, SILog, iRMSE])` `model = create-gen-learner()` Fastai has different pre-trained models, namely Resnet18, Resnet34,

Resnet50, DenseNet, XResnet-etc. These are used as the backbone architectures in the contraction part of the U-Net architecture. These pre-trained models leads to the concept of transfer learning, which is explained in the next sub-section.

4.1.2.6 Applying Transfer Learning

The `cnn-learner` method or Dynamic U-Net learner method in Fastai, is used to fetch a pre-trained model for transfer learning. As discussed in the previous sub-section that in Fastai, we can use Resnet18, Resnet34, Resnet50, DenseNet, XResnet etc as backbone architectures in the Dynamic U-Net-learner. All these models had been pre-trained on ImageNet. One just needs to pass the required architecture (`models.Resnet34`, `models.Resnet50` etc.) in the ‘arch’ argument of the U-Net-learner, and select ‘pre-trained = True’ for selecting the pre-trained model of backbone architecture. The same was done in this research work.

4.1.2.7 Training Metric/Loss Function

Training Metric commonly known as the loss function, is the metric which calculates the loss at the final output layer of the network between the predicted output and the ground truth used for training. The gradient of the loss function w.r.t. the network weights is also computed at the output and is then sent back through the network via back propagation. Here, the Scale Invariant Error as the training metric in this thesis. Scale Invariant logarithmic error was initially used by Eigen et.al [13], [14] both as a training and evaluation loss for their network. It was also adopted by KITTI benchmark suite [26] as the main evaluation metric for ranking different methods/ submissions. Eigen et.al [13], [14] argued that the global scale of a scene is a fundamental ambiguity in depth prediction and much of the error accrued using element wise metrics may be explained simply by how well the mean depth is predicted. They explained from the example that Make 3D trained on NYUDepth obtains 0.41 error using RMSE in log space. However, by only substituting the mean log depth of each prediction with the mean from

the corresponding ground truth reduces the error to 0.33, a 20 percent relative improvement. Likewise, for their system, these error rates were 0.28 and 0.22, respectively. They argued that thus, just finding the average scale of the scene accounts for a large fraction of the total error. Motivated by this, they used a scale-invariant error to measure the relationships between points in the scene, irrespective of the absolute global scale. For a predicted depth map y and ground truth y' , each with n pixels indexed by i , they defined the scale-invariant mean squared error (in log space) as:

4.1.2.8 Starting the Training – Fit One-Cycle

After creation of U-Net-learner and model for training, finally, we are ready to start the training process. This is accomplished by using following line of code:

```
model.fit-one-cycle(1, pct-start=0.8)
```

Here, the learning rate or differential learning rates can be specified in the arguments of ‘.fit-one-cycle’. The fit one-cycle training technique was discussed in above section and has proved to be amongst the best training practices in the past 3 years.

4.1.2.9 Hyper Parameter Setting/Tuning

Following hyper parameter setting were selected initially;

TABLE 4.1: Initial Hyper Parameters.

S.no	Hyper Parameters	Values
1	Learning Rate	1e-3 (1-5 Epochs)
2	Momentum	[0.95, 0.85, 0.95]
3	Weight Decay Rate	1e-3
4	Batch Size	32 (Resnet34),8 (Resnet50)
5	Image Size during Training	224x224

The learning rate was selected by using learning rate finder, whereas default values of momentum and weight decay rate were selected as shown above. These were further tuned by observing the output of each set of training cycle and by using the learning rate finder before each set of epochs/training cycle. The final hyper parameters setting are shown below.

TABLE 4.2: Final Hyper Parameters.

S.no	Hyper parameters	Values
1	Learning Rate(Resnet34)	1e-3 (1-5 Epochs), 1e-5 (6-15 Epochs), 1e-6 (16-50 Epochs)
2	Learning Rate(Resnet50)	1e-3 (1-5 Epochs), 1e-4 (6-10 Epochs), 1e-6 (11-20 Epochs)
3	Momentum	[0.975, 0.93]
4	Weight Decay Rate	1e-4
5	Batch Size	32 (Resnet34), 8 (Resnet50)
6	Image Size during Training	224x224

By monitoring the values of training error, higher values of momentum were selected to avoid error getting stuck in a local minima. It was observed that by selecting these values of Hyper-parameters, abrupt changes in the training loss and validation metrics were avoided.

4.2 Testing Methodology

In test phase, the trained convolutional neural network is made to see images which were not used during the training phase. Hence, the network is made to estimate error/loss on previously unseen images. The error/loss thus obtained is called validation loss. As during the test phase no more training is meant, so the validation loss is not back propagated. The validation loss is estimated by loss function / training metric. So here, the validation loss is obtained using Scale Invariant Error which was used as the training metric. However, other evaluation metrics can also be calculated on this test set. These metrics are only applied to

the test set and not on training set. In this thesis, same 4 metrics were used as proposed by Eigen et.al [13], [14]. Same evaluation metrics are also used as reference for the leader board by KITTI Benchmark suite [26]. These metrics are further discussed in detail in the following section. For KITTI, a 14224 no of images are used in the test set. A separate test set of 500 images is also provided for submitting results on [26]. In this work, both the Training and Validation was performed in a single step as our Image Data Bunch was constructed in such a way that it consisted of both the Training and test sets along with their corresponding ground truth images. Although, the Testing or Validation step can be performed separately in Fastai.

4.2.1 Evaluation Metrics

As discussed earlier, 4 evaluation metrics were used, namely Relative Squared Error, Relative Absolute Error, Root Mean Squared error, Inverse Root Mean Squared Error and Scale Invariant Logarithmic Error. The brief description of these error metrics is given below:

- SILog: Scale invariant logarithmic error [$\log(m)*100$]

$$D(y, y^*) = \frac{1}{n} \sum_{i=1}^n d_i^2 - \frac{1}{n^2} \left(\sum_{i=1}^n d_i \right)^2 \quad d_i = \log y_i - \log y_i^* \quad (4.1)$$

- sqErrorRel: Relative squared error (percent)

$$\frac{1}{|T|} \sum_{y \in T} \|y - y^*\|^2 / y^* \quad (4.2)$$

- absErrorRel: Relative absolute error (percent)

$$\frac{1}{|T|} \sum_{y \in T} |y - y^*| / y^* \quad (4.3)$$

- RMSE: Root Mean Square Error

$$\sqrt{\frac{1}{|T|} \sum y \epsilon^T \|y - y^*\|^2 / y^*} \quad (4.4)$$

- RMSE: Root Mean Square Error

$$\frac{1}{\sqrt{\frac{1}{|T|} \sum y \epsilon^T \|y - y^*\|^2 / y^*}} \quad (4.5)$$

4.3 Summary

This chapter discussed the proposed training methodology. How the dataset was cleaned, the labelling process by associating each input RGB image with its corresponding ground truth depth map, the image transformations and normalization applied after the formation of Image data bunch and how the hyper parameters were initially selected and after tuning what were the final hyper parameters. The testing methodology was also discussed by highlighting the evaluation metrics used.

The coming chapter discusses the results obtained by using Resnet34 and Resnet50 as backbone architectures in Fastai's Dynamic U-Net configuration and their comparison with the state of the art. Qualitative analysis is also presented.

Chapter 5

Results

The architectures used in this research study were trained on KITTI dataset for the problem of depth prediction. As discussed earlier, Resnet34 and Resnet50 were used as the backbone architectures in Unet configuration. Their performance was evaluated using NVIDIA Tesla V100 (16 GB). As it can be seen from figure 3.6 and 3.7, by comparison, Resnet50 in dynamic Unet configuration is a very heavy architecture with even up to 4000 feature channels in the lowest resolution. Resultantly, it is memory intensive with greater training time and most importantly puts a limitation on the batch size. These differences in performance and evaluation of Resnet34 and Resnet50 are discussed in the following sub-sections:-

5.1 Resnet34

The Resnet34 architecture is comparatively light weight and can accommodate even up to 64 images in a batch. However, a batch size of 32 images was used to aid the training process. The learning rates were selected using the learning rate finder and as described in section 4.2.3.8. While training of KITTI dataset using Fastai's dynamic Unet architecture with Resnet34 as backbone architecture, it was observed that training of the network could not be achieved even after 50 epochs with different learning rates. It was observed that this was attributed to the

last Sigmoid layer which added non-linearity to the predicted output, however, as discussed in [14], the last layer needs to be linear for linear prediction of depth. For the same purpose, as discussed in section 4.1.2, the last Sigmoid layer was removed from the architecture and training was done with a learning rate of 0.001. It can be seen from table 5.1, that the training loss kept on decreasing during the training process. The qualitative results shown in the figures below also demonstrate the successful training process where the network started to learn to predict depth as perceptible from the predictions of 20-50 epochs. Some of the qualitative results are shown in figures below 5.1-5.5:-

TABLE 5.1: Results of Training and Validation: *Resnet34* in *Fastai's* Dynamic Unet Configuration trained on KITTI dataset.

Epochs	Train_loss	Valid_loss	absErrorRel	sqErrorRel	RMSE	SILog	iRMSE
1	0.044507	0.023822	1.043539	0.277342	0.342754	4.078214	29.558901
2	0.020986	0.019410	0.878170	0.227262	0.329850	2.951670	24.883390
3	0.018777	0.018202	0.836591	0.213758	0.325253	2.545815	23.296675
4	0.018314	0.017954	0.824463	0.210398	0.323854	2.391117	22.986847
5	0.018113	0.017768	0.818567	0.213603	0.326852	2.611352	24.568414
6	0.017962	0.017683	0.814865	0.209689	0.324307	2.384638	24.082594
7	0.017943	0.017644	0.812557	0.209460	0.32446	2.328062	22.942423
8	0.017828	0.017569	0.805442	0.213079	0.327347	2.459449	22.548231
9	0.017814	0.017494	0.806299	0.209596	0.324967	2.306800	23.408339
10	0.017819	0.017467	0.810561	0.205911	0.322031	2.125575	21.208149
11	0.017767	0.017462	0.810819	0.206548	0.322444	2.162168	21.688185
12	0.017743	0.017458	0.812211	0.204394	0.320893	2.073627	21.176130
13	0.017124	0.016753	0.870199	0.191745	0.306157	1.606615	16.753599
14	0.016859	0.016596	0.903645	0.193347	0.302613	1.596483	16.753317
15	0.016801	0.016494	0.885137	0.194478	0.305597	1.660061	17.045897
16	0.016785	0.016462	0.892332	0.194795	0.304751	1.667236	17.223061
17	0.016735	0.016404	0.913681	0.194017	0.301243	1.590326	16.754614
18	0.016724	0.016340	0.888654	0.195166	0.305186	1.686453	17.299728
19	0.016643	0.016222	0.904936	0.914466	0.302158	1.631068	17.008587
20	0.016641	0.016149	0.900397	0.195866	0.303161	1.674783	17.247437

The state of the art results shown on the KITTI dataset site and their comparison with the results obtained from using Resnet34 as backbone architecture in Dynamic U-Net architecture is shown in table 5.2:

TABLE 5.2: Comparison with State of the Art with Results of Resnet34 .

	SILog	sqErrorRel	absErrorRel	iRMSE
State of the Art [30]	11.12 (MPD)	2.07 (MPD)	8.78 (DL 61 - DORN)	11.56 (MPD)
Ours (Resnet34)	1.674	0.195	0.900	17.247

It can be observed from the figure 5.1 that network after 10 epochs of training has not yet learned to predict depths. This can be observed from the shadows of objects seen in the predictions.

One can say that the network has learned some gray level values of the input image.

For the same reason the network was trained again for next 10 epochs with the same learning rate, results of which are shown in the figure 5.2.

It can be observed from the figure 5.2, that the network has now started to learn something about the depth. It can be seen from the shading of depth values where the objects closer to the camera seems to be darker than those farther away.

The network was trained again for next 30 epochs in 10 epochs cycles. The results of 30, 40 and 50 epochs, respectively, are shown in the figures 5.3-5.5.

Furthermore, it can be observed from the figures shown below, that the difference in the qualitative results after 30 epochs is very small.

However, the predictions are more refined after 50 epochs than that obtained after 30 epochs. The training was stopped after 50 epochs and model was saved.



FIGURE 5.1: *Resnet34* Results (10 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).

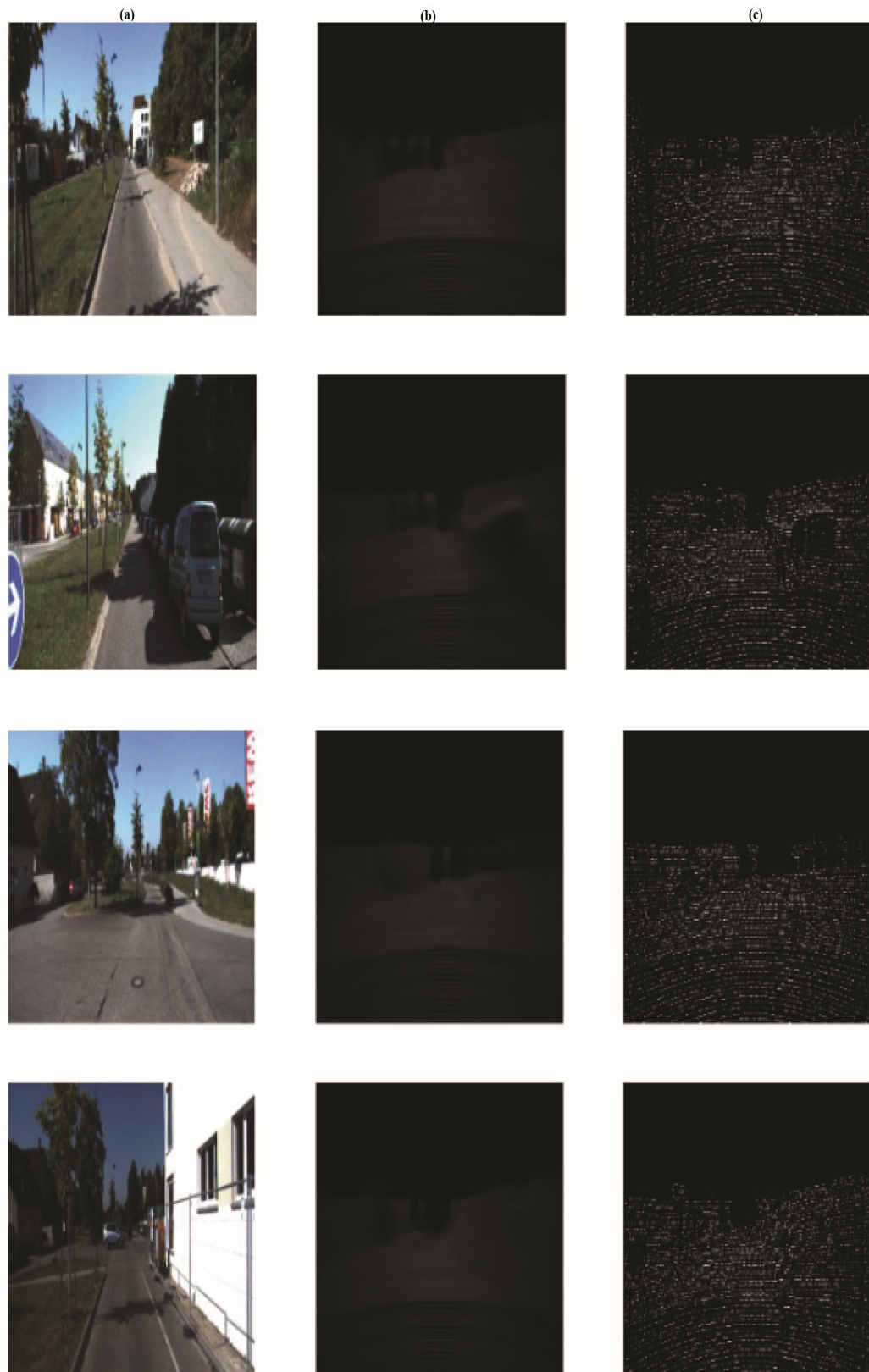


FIGURE 5.2: *Resnet34* Results (20 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).



FIGURE 5.3: *Resnet34* Results (30 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).



FIGURE 5.4: *Resnet34* Results (40 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).



FIGURE 5.5: *Resnet34* Results (50 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).

It can be observed from the figures shown above, that the difference in the qualitative results after 30 epochs is very small. However, the predictions are more refined after 50 epochs than that obtained after 30 epochs. The training was stopped after 50 epochs and model was saved.

5.2 Resnet50

The Resnet50 architecture is computationally heavy (approximately 8.5 times heavier than Resnet34) and is quite memory intensive. For the same reason a batch size of 8 was selected with input images cropped to sizes of 224 x 224 each. For setting the hyper-parameters, same procedure was followed as done with Resnet34.

While training of KITTI dataset using Resnet50, it was observed that the qualitative results of Resnet50 were more promising than that of Resnet34. The qualitative and quantitative results of Resnet50 obtained after first 10 epochs are comparable with those obtained after 50 epochs using Resnet34.

This can be seen from Table 5.2 and figures shown below:- There is a mark difference in performance when comparing qualitative results of Resnet34 with those obtained from Resnet50. Due to the deeper architecture of Resnet50, the qualitative results obtained from it are very fine and very closely resemble with the original ground truth images.

The state of the art results shown on the KITTI dataset site and their comparison with the results obtained from using Resnet50 as backbone architecture in Dynamic U-Net architecture is shown in table 5.3.

TABLE 5.3: Comparison with State of the Art with Results of Resnet50 .

	SILog	sqErrorRel	absErrorRel	iRMSE
State of the Art [30]	11.12 (MPD)	2.07 (MPD)	8.78 (DL 61 - DORN)	11.56 (MPD)
Ours (Resnet50)	1.618	0.195	0.923	17.105

TABLE 5.4: Results of Training and Validation: *Resnet50* in *Fastai's* Dynamic Unet Configuration trained on KITTI dataset.

Epochs	Train_loss	Valid_loss	absErrorRel	sqErrorRel	RMSE	SILog	iRMSE
1	0.016905	0.016666	0.880358	0.197651	0.308287	1.826330	18.151844
2	0.016874	0.016575	0.841671	0.198997	0.313791	1.879078	18.368332
3	0.016816	0.016360	0.904865	0.194935	0.302883	1.653086	16.945129
4	0.016727	0.016310	0.877334	0.196853	0.307284	1.765261	17.616001
5	0.016692	0.016163	0.904092	0.194550	0.301884	1.659872	17.038559
6	0.016627	0.016055	0.926666	0.197939	0.299499	1.656190	17.084084
7	0.016432	0.016028	0.894410	0.196713	0.303381	1.700606	17.255571
8	0.016531	0.016037	0.915216	0.201022	0.301724	1.735559	17.467339
9	0.016391	0.015777	0.910926	0.197853	0.300076	1.698460	17.454256
10	0.016351	0.015655	0.921312	0.197711	0.297320	1.656444	17.200312
11	0.016241	0.015677	0.914367	0.196569	0.297961	1.647794	17.209614
12	0.016222	0.015638	0.922395	0.197257	0.296910	1.636623	17.167606
13	0.016291	0.015652	0.923523	0.199488	0.297493	1.648013	17.248308
14	0.016138	0.015719	0.927439	0.197951	0.296691	1.624606	17.063349
15	0.016265	0.015654	0.930718	0.199987	0.296829	1.666598	17.361515
16	0.016108	0.015718	0.918306	0.204088	0.301175	1.798275	18.290274
17	0.016156	0.015698	0.926051	0.194696	0.296157	1.574602	16.798025
18	0.016135	0.015591	0.916005	0.199595	0.298452	1.703840	17.644588
19	0.016152	0.015486	0.932137	0.197486	0.294307	1.618911	17.127268
20	0.016140	0.015452	0.923136	0.195136	0.294110	1.618750	17.105338



FIGURE 5.6: *Resnet50* Results (10 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).

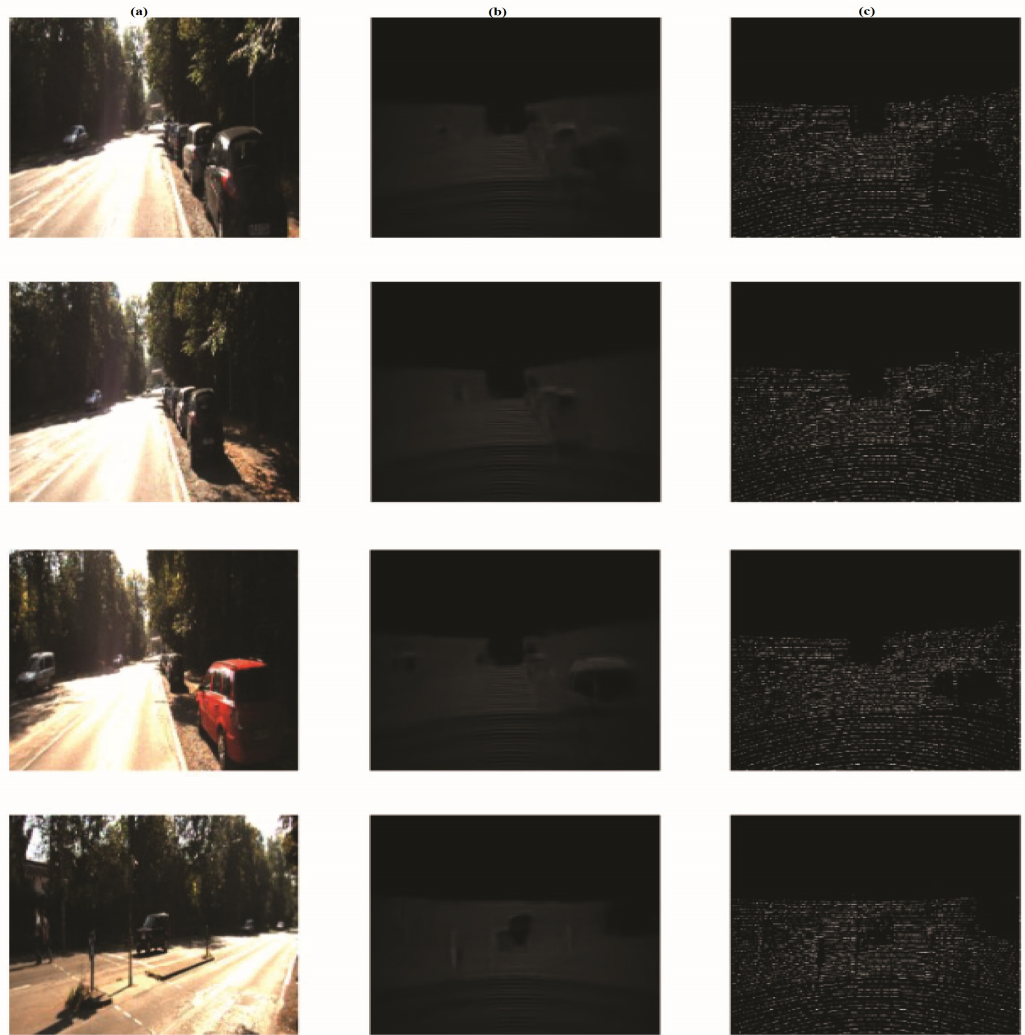


FIGURE 5.7: *Resnet50* Results (10 epochs) in Dynamic Unet Configuration (without sigmoid layer) From Left to Right: (a) Original RGB image, (b) The Prediction on Validation/Test Set Images (c) Raw LiDAR Scan Data (used for training as GT).

5.3 Comparison with Different Architectures

SILog (Scale Invariant Logarithmic Error) is considered as the main distinguishing metric when comparing results of different methods. The SILog calculated on the validation set using the approach mentioned in this thesis, came to be very low when comparing it with the state of the art (mentioned on the leader board of KITTI, as seen in Table 5.2). However, the results mentioned in Table 5.2 were evaluated on the test set by team managing KITTI dataset, nevertheless, it can

be assumed that the same high quality results can be achieved on the test set by the approach mentioned in this research study.

TABLE 5.5: Leader Board for KITTI Data Set for Depth Prediction: Ranking Methods from Top to Bottom Based on the *SILog* Error Metric.

Method	SILog	sqErrorRel	absErrorRel	iRMSE	Runtime
DeepLab	10.80	2.19	8.94	11.77	0.1 s
MPSD	11.12	2.07	8.99	11.56	0.1 s
GSM	11.23	2.13	8.88	12.65	0.06 s
siit	11.50	2.30	9.04	12.33	0.02 s
GSM	11.56	2.25	8.99	12.44	0.05 s
LCI	11.59	2.21	9.09	12.18	0.03 s
BANet	11.61	2.29	9.38	12.23	0.04 s
BTS	11.67	2.21	9.04	12.23	0.06 s
AcED	11.70	2.45	9.54	12.51	0.5 s
DL_61 (DORN)	11.77	2.23	8.78	12.98	0.5 s
RefinedMPL	11.80	2.31	10.09	13.39	0.05 s
BiNet-SOC	11.83	2.66	10.12	12.79	0.04 s
BTS-256	12.05	2.43	9.39	13.11	0.1
AcED	12.27	2.48	9.55	12.91	0.5 s
DL_SORD_SL	12.39	2.49	10.10	13.48	0.8 s
VNL	12.65	2.46	10.15	13.02	0.5 s
DS-SIDENet_ROB	12.86	2.87	10.03	14.40	0.35 s
DL_SORD_SQ	13.00	2.95	10.38	13.78	0.88 s
PAP	13.08	2.72	10.27	13.95	0.18 s
VGG16-UNet	13.41	2.86	10.60	15.06	0.16 s
DORN_ROB	13.53	3.06	10.35	15.96	2 s
SSDE	14.45	3.60	11.47	15.52	0.1 s
DABC_ROB	14.49	4.08	12.72	15.53	0.7 s
SDNet	14.68	3.90	12.31	15.96	0.2 s
APMoE_base_ROB	14.74	3.88	11.74	15.63	0.2 s
FIS-Nets	14.76	3.56	11.41	15.74	0.06 s
MonoDeMo	14.84	4.04	12.28	15.69	0.01 s
CSWS_E_ROB	14.85	3.48	11.84	16.38	0.2 s
HBC	15.18	3.79	12.33	17.86	0.05 s
SGDepth	15.30	5.00	13.29	15.80	0.1 s
semiDepth	15.34	4.20	11.73	16.66	0.02 s
DHGRL	15.47	4.04	12.52	15.72	0.2 s
AM-mono	15.77	67.30	81.95	499.22	0.02 s
Mono-pad-net	15.87	4.60	13.10	16.98	0.1 s
FCRN_ROB	15.93	4.06	12.10	16.51	0.2 s
MultiDepth	16.05	3.89	13.82	18.21	0.01 s
AI Mono Tech.	17.21	6.98	13.60	16.80	0.04 s
Modu_selfdriving_ROB	17.54	7.69	14.61	17.77	0.1 s
LSIM	17.92	6.88	14.04	17.62	0.08 s
BESEG	23.91	24.14	27.83	30.52	3 s
RVGNet_ROB	37.71	10.66	23.39	62.48	0.3 s
RVGNet	40.91	13.35	28.03	44.54	0.3 s

5.4 Analysis

5.4.1 Accuracy

In terms of validation metrics, Dynamic U-Net architecture achieved far better results than contemporary heavier architectures as shown in table 5.1 and table 5.3. This shows the effectiveness of Dynamic U-Net architecture in depth regression task. This can also be attributed to the best training approaches adopted in this research work.

5.4.2 Computational Time

The Computational Time for Resnet34 in U-Net Configuration for 1 epoch on NVIDIA TESLA V100 for a subset of KITTI dataset (approximately 6k images) was 1 minute and 46 seconds, for images of size 224 x 224 and batch size of 32. The Computational Time for Resnet50 in U-Net Configuration for 1 Epoch on NVIDIA TESLA V100 for a subset of KITTI dataset (approximately 6k images) was 11 minute and 54 seconds, for images of size 224 x 224 and batch size of 8.

5.4.3 Comparison of Resnet34 and Resnet50 in Terms of Training Cycles

By comparing the figure 5.5 (qualitative results of Resnet34 after 50 Epochs of training) with figure 5.6 (qualitative results of Resnet50 after 10 Epochs of training), it can be said that Resnet50 gives superior results (after 10 Epochs of its training) than Resnet34 even after 50 Epochs of its training. This shows that the shallower the network the more training cycles it requires.

5.4.4 Limitation

Some of the limitations of this research work are given below:

- The sparsity of LiDAR’s ground truth data is one of the main limitations of monocular depth perception as holes in the ground truth data are replicated in the predictions. Which ultimately gives error in depth filling.
- The fastest GPU (NVIDIA TESLA V100) was utilized (rented from floydhub cloud GPU platform) to perform training on KITTI dataset. Although still it was infeasible to utilize full image resolution for training of data. The original image size of KITTI dataset was 370 x 1252 which was down sampled to 224 x 224 to avoid longer training cycles, as for only a subset of KITTI dataset (approximately 6k images), the computational time for full sized images for Resnet34 was approximately 18 minutes and that of Resnet50 was 1 hour for 1 Epoch. This size of 224 x 224 is chosen because it is thought to be the ideal size for many deep learning applications by the deep learning practitioners [21]. Secondly, the intuition behind selecting this size of images was that, by down sampling the original resolution of images to half, gives a resolution of 185 x 626 which is still not computationally feasible. By further down sampling it gives a resolution of 92 x 313. But this resolution is not suitable at all in terms of loss of features in resolution of 1st dimension i.e. ‘92’ and it is considered by all deep learning practitioners, that resolution of dimensions below 128 behave strangely [21] and should never be selected
- One of the limitations of the original Fastai’s Dynamic U-Net was the presence of last Sigmoid Layer in Dynamic U-NET architecture, as it was designed for classification tasks and does not support image to image regression tasks as highlighted in the previous sections, so the same was removed.

5.5 Summary

In Chapter 5, both the qualitative and quantitative results of Dynamic U-Net with Resnet34 and Resnet50 as backbone architecture on a subset of KITTI dataset (with approximately 6k images) were presented. Comparison of Dynamic U-Net architecture with Resnet34 as backbone architecture and Resnet50 as backbone architecture was made with each other and as well as with contemporary state of the art.

Chapter 6

Conclusion and Future work

6.1 Conclusion

Depth estimation is one of the fundamental ambiguities in the field of computer vision. Currently, a great focus has been shifted on monocular depth perception, which is an inherently ill-posed problem. Usually machine learning and especially deep learning techniques are used to address this problem. Supervised machine learning has proved to give superior results employing a full image regression scheme. Such approaches usually require a computationally heavy architecture. Which brings us to one of the main challenges in the field of deep learning, i.e. to design deeper architectures with low computational cost and that can work equally well on small training sets. U-Net architecture is one in this case which was originally designed for medical image segmentation/classification tasks, where usually in many cases, very less training data exists.

For the same reason Fastai's Dynamic U-Net architecture was used for the first time for monocular depth perception on KITTI Dataset. It proved to be a very good choice after tailoring it for the depth regression task. Furthermore, using the Fastai library provided excellent support for the best training approaches adopted by competition winning deep learning practitioners. These approaches proved to

be pivotal in achieving high accuracy, especially the learning rate finder which helped to select the optimum learning rates in different training cycles and discriminative learning rates which helped to select the optimum learning rates for different sets of layers of the architecture.

Taking the case of Resnet34 as backbone architecture in Dynamic U-Net, it proved to be a deeper architecture with far lesser parameters than contemporary state of the art heavy architectures and yet provided better results.

Furthermore, for the task of depth regression, longer training cycles are required as compared to the requirement in the classification tasks. Moreover, the shallower the network, more is the length of training cycles. Larger values of momentum are also required in the task of depth regression. The results obtained show the effectiveness of approach followed in this thesis.

6.2 Future Work

The work presented in this thesis provides a solid base to incorporate and examine in a U net configuration, other state of the art architectures designed for classification (E Net, Dense Net, Inception Net etc.), for the task of depth prediction. The same technique with an addition of a guidance network can also be used for the task of depth completion. Depth completion can then serve as a preliminary step towards the final depth prediction task on filled LIDAR data rather than a sparse one.

Bibliography

- [1] A. Saxena, S. H. Chung, and A. Y. Ng, “Learning depth from single monocular images,” in *Neural Information Processing systems (NIPS)*, vol. 18, 2005.
- [2] D. Cheda, “Monocular depth cues in computer application,” Ph.D. dissertation, PhD Thesis, 2012.
- [3] Y. Kuznetsov, J. Stuckler, and B. Leibe, “Semi-supervised deep learning for monocular depth map prediction,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6647–6655.
- [4] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [6] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.
- [7] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

-
- [9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The cityscapes dataset for semantic urban scene understanding,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.
- [10] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, “Indoor segmentation and support inference from rgb-d images,” in *European conference on computer vision*. Springer, 2012, pp. 746–760.
- [11] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*. Springer, 2014, pp. 818–833.
- [12] F. Khan, S. Salahuddin, and H. Javidnia, “Deep learning-based monocular depth estimation methods—a state-of-the-art review,” *Sensors*, vol. 20, no. 8, p. 2272, 2020.
- [13] D. Eigen, C. Puhrsch, and R. Fergus, “Depth map prediction from a single image using a multi-scale deep network,” in *Advances in neural information processing systems*, 2014, pp. 2366–2374.
- [14] D. Eigen and R. Fergus, “Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2650–2658.
- [15] R. Garg, V. K. Bg, G. Carneiro, and I. Reid, “Unsupervised cnn for single view depth estimation: Geometry to the rescue,” in *European conference on computer vision*. Springer, 2016, pp. 740–756.
- [16] C. Godard, O. Mac Aodha, and G. J. Brostow, “Unsupervised monocular depth estimation with left-right consistency,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

- [17] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, “Deeper depth prediction with fully convolutional residual networks,” in *2016 Fourth international conference on 3D vision (3DV)*. IEEE, 2016, pp. 239–248.
- [18] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.
- [19] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.
- [20] H. Zhang, Y. N. Dauphin, and T. Ma, “Fixup initialization: Residual learning without normalization,” *arXiv preprint arXiv:1901.09321*, 2019.
- [21] Fastai. Accessed on 06 Jan 2020.[Online]. Available: <http://www.fast.ai>
- [22] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze, “Fastdepth: Fast monocular depth estimation on embedded systems,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6101–6108.
- [23] Technologyreview. Accessed on 06 Jan 2020.[Online]. Available: <http://www.technologyreview.com/2018/08/10/141098/small-team-of-aicoders-beats-googles-code>
- [24] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.
- [25] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *arXiv preprint arXiv:1905.11946*, 2019.

-
- [26] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.
- [27] P. Isola, J. Xiao, A. Torralba, and A. Oliva, "What makes an image memorable?" in *CVPR 2011*. IEEE, 2011, pp. 145–152.
- [28] K. S. Chan, "Multiview monocular depth estimation using unsupervised learning methods," Ph.D. dissertation, Massachusetts Institute of Technology, 2018.
- [29] M. Ribo, A. Pinz, and A. L. Fuhrmann, "A new optical tracking system for virtual and augmented reality applications," in *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No. 01CH 37188)*, vol. 3. IEEE, 2001, pp. 1932–1936.
- [30] T.-T. Lin, Y.-K. Hsiung, G.-L. Hong, H.-K. Chang, and F.-M. Lu, "Development of a virtual reality gis using stereo vision," *Computers and electronics in agriculture*, vol. 63, no. 1, pp. 38–48, 2008.
- [31] S. Livatino, G. Muscato, and F. Privitera, "Stereo viewing and virtual reality technologies in mobile robot teleguide," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1343–1355, 2009.
- [32] D. Murray and J. J. Little, "Using real-time stereo vision for mobile robot navigation," *autonomous robots*, vol. 8, no. 2, pp. 161–171, 2000.
- [33] S. B. Goldberg, M. W. Maimone, and L. Matthies, "Stereo vision and rover navigation software for planetary exploration," in *Proceedings, IEEE aerospace conference*, vol. 5. IEEE, 2002, pp. 5–5.
- [34] D. Brescianini, M. Hehn, and R. D'Andrea, "Quadrocopter pole acrobatics," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 3472–3479.

-
- [35] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka, “A stereo machine for video-rate dense depth mapping and its new applications,” in *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 1996, pp. 196–202.
- [36] O. D. Faugeras and F. Lustman, “Motion and structure from motion in a piecewise planar environment,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, no. 03, pp. 485–508, 1988.
- [37] J. J. Koenderink and A. J. Van Doorn, “Affine structure from motion,” *JOSA A*, vol. 8, no. 2, pp. 377–385, 1991.
- [38] J. L. Schonberger and J.-M. Frahm, “Structure-from-motion revisited,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4104–4113.
- [39] R. Ranftl, V. Vineet, Q. Chen, and V. Koltun, “Dense monocular depth estimation in complex dynamic scenes,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4058–4066.
- [40] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, “Mav navigation through indoor corridors using optical flow,” in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 3361–3368.
- [41] F. Kendoul, I. Fantoni, and K. Nonami, “Optic flow-based vision system for autonomous 3d localization and control of small aerial vehicles,” *Robotics and autonomous systems*, vol. 57, no. 6-7, pp. 591–602, 2009.
- [42] B. Herisse, F.-X. Russotto, T. Hamel, and R. Mahony, “Hovering flight and vertical landing control of a vtol unmanned aerial vehicle using optical flow,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 801–806.
- [43] J. Thatte, J.-B. Boin, H. Lakshman, and B. Girod, “Depth augmented stereo panorama for cinematic virtual reality with head-motion parallax,” in *2016*

- IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2016, pp. 1–6.
- [44] B. Luo, F. Xu, C. Richardt, and J.-H. Yong, “Parallax360: stereoscopic 360 scene representation for head-motion parallax,” *IEEE transactions on Visualization and Computer Graphics*, vol. 24, no. 4, pp. 1545–1553, 2018.
- [45] M. F. Bradshaw, A. D. Parton, and A. Glennerster, “The task-dependent use of binocular disparity and motion parallax information,” *Vision research*, vol. 40, no. 27, pp. 3725–3734, 2000.
- [46] W. N. Klarquist, W. S. Geisler, and A. C. Bovik, “Maximum-likelihood depth-from-defocus for active vision,” in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 3. IEEE, 1995, pp. 374–379.
- [47] V. P. Namboodiri and S. Chaudhuri, “On defocus, diffusion and depth estimation,” *Pattern Recognition Letters*, vol. 28, no. 3, pp. 311–319, 2007.
- [48] O. Ghita and P. F. Whelan, “A video-rate range sensor based on depth from defocus,” *Optics & Laser Technology*, vol. 33, no. 3, pp. 167–176, 2001.
- [49] A. Levin, R. Fergus, F. Durand, and W. T. Freeman, “Image and depth from a conventional camera with a coded aperture,” *ACM transactions on graphics (TOG)*, vol. 26, no. 3, pp. 70–es, 2007.
- [50] torres, “Comparison of the snapdragon flight realsense andzed stereo cameras,” 2017.
- [51] Y. Cao, Z. Wu, and C. Shen, “Estimating depth from monocular images as classification using deep fully convolutional residual networks,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 11, pp. 3174–3182, 2017.
- [52] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

-
- [53] L. Ladicky, J. Shi, and M. Pollefeys, “Pulling things out of perspective,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 89–96.
- [54] A. Saxena, M. Sun, and A. Y. Ng, “Make3d: Learning 3d scene structure from a single still image,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 824–840, 2008.
- [55] I. Alhashim and P. Wonka, “High quality monocular depth estimation via transfer learning,” *arXiv preprint arXiv:1812.11941*, 2018.
- [56] J. West, D. Ventura, and S. Warnick, “Spring research presentation: A theoretical foundation for inductive transfer,” *Brigham Young University, College of Physical and Mathematical Sciences*, vol. 1, no. 08, 2007.
- [57] I. L. S. V. R. Challenge, “Available online: <http://www.image-net.org/challenges/>,” *LSVRC/(accessed on 06 Aug 2020)*, 2014.
- [58] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, “Unsupervised learning of depth and ego-motion from video,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1851–1858.
- [59] D. Scaramuzza, F. Fraundorfer, M. Pollefeys, and R. Siegwart, “Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints,” in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 1413–1419.
- [60] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox, “Demon: Depth and motion network for learning monocular stereo,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5038–5047.
- [61] L. N. Smith, “Cyclical learning rates for training neural networks,” in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2017, pp. 464–472.

- [62] A. Vedaldi, “Cats and dogs,” in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3498–3505.

Appendix A

Installation of Required Frameworks

A.1 Installation of Anaconda Python3

As a 1st step, we need to download Python. We selected the latest version of Python3 (Python 3.7) for development. Python is a programming language but we want an Integrated Development Environment to use it. We have selected the Anaconda distribution for Python3 for subject purpose (as mostly used world-wide). The Anaconda distribution not only downloads and install Python but it further has different Integrated Development Environments (IDEs) including Jupyter Notebooks, which is a Web-based Interactive Computing Notebook Environment. We will further use the Jupyter Notebooks for development in our application. The Anaconda distribution for Python can be found at: '<https://www.anaconda.com/products/individual>'. After clicking on the required distribution (Windows, 64-bit Operating system and Python 3.7 in our case), the download process of its executable file will start as shown in the figure below:

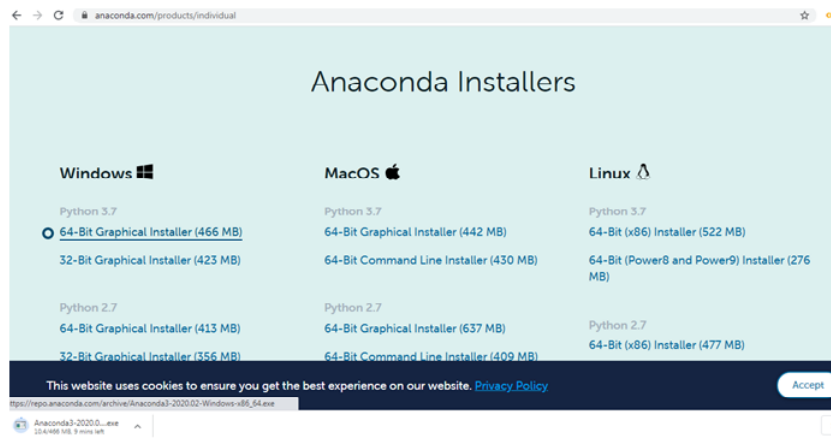


FIGURE A.1: 64-Bit Graphical Installer of *Anaconda Python3*.

The Anaconda Python3 can then be installed by running the downloaded executable file of Anaconda distribution.

A.2 Installation of Pytorch, Torchvision , CUDA/ cuDNN Drivers for GPU

Pytorch can be installed from <https://pytorch.org/> . After selecting the required preferences for your system (with GPU) as shown in the following figure, you can copy the install command as highlighted in the red box and run in the Anaconda prompt (Anaconda prompt should be run from startup menu as ‘Run as Administrator’, after right clicking its option).

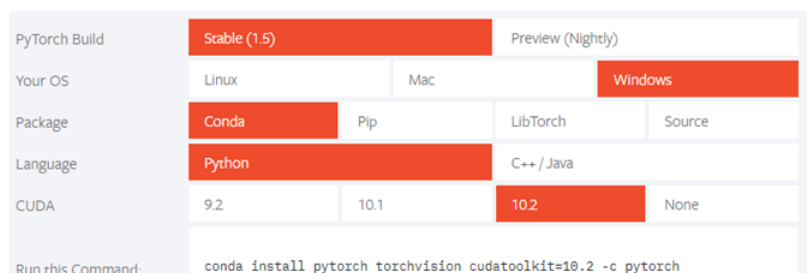
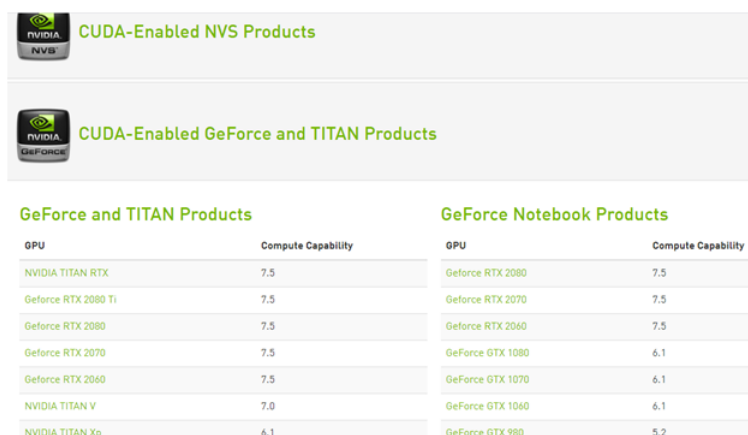


FIGURE A.2: Conda Install Pytorch Torchvision Cudatoolkit=10.2.

We were using Windows and selected ‘conda’ as the package manager. We also selected the latest CUDA drivers. For our case, the command ‘conda install pytorch torchvision cudatoolkit=10.2 -c pytorch’, will automatically download the latest torchvision module required for computer vision applications in deep learning. It will also automatically download all the required CUDA drivers for use of GPU and there will not be any need to go to Nvidia site and separately install CUDA/cuDNN drivers as per your GPU requirement. It is be NOTED that to successfully run deep learning platforms, your GPU compute capability should be greater than 3 (as highlighted by Andrew Ng, in his course of deep learning on coursera). The compute capability of GPUs can be found at ‘[https : //developer.nvidia.com/cuda – gpuscompute](https://developer.nvidia.com/cuda-gpuscompute)’. For example, for finding the compute capability of GeForce and Titan GPUs, we will go to the above mentioned site and click on the option of ‘CUDA-Enabled GeForce and Titan Products’. It will display following results:



GeForce and TITAN Products		GeForce Notebook Products	
GPU	Compute Capability	GPU	Compute Capability
NVIDIA TITAN RTX	7.5	Geforce RTX 2080	7.5
Geforce RTX 2080 Ti	7.5	Geforce RTX 2070	7.5
Geforce RTX 2080	7.5	Geforce RTX 2060	7.5
Geforce RTX 2070	7.5	GeForce GTX 1080	6.1
Geforce RTX 2060	7.5	GeForce GTX 1070	6.1
NVIDIA TITAN V	7.0	GeForce GTX 1060	6.1
NVIDIA TITAN Xp	6.1	GeForce GTX 980	5.2

FIGURE A.3: CUDA-Enabled GeForce and Titan Products.

If you do not have a GPU and want to download Pytorch/ torchvision for CPU only, then select ‘None’ in the ‘CUDA’ option (from <https://pytorch.org/>) , as shown in the following figure:

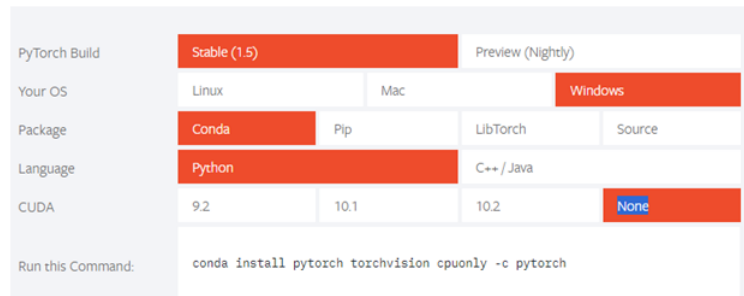


FIGURE A.4: Pytorch Installer for CPU only.

Running the install command ‘`conda install pytorch torchvision cpuonly -c pytorch`’, in the Anaconda prompt will download and install the desired Pytorch stable version for ‘cpu only’. After downloading Pytorch / torchvision, we are now ready to install the Fastai library, which is built on Pytorch.

A.3 Installation of Fastai

The detailed instructions in this regards can be found by following the fastai installation guide at ‘<https://docs.fast.ai/install.html>’. One can use the desired package manager (pip or conda) for its installation. For example, running the following command in the Anaconda prompt will do the job for us: `pip install fastai` This is shown in the following figure:

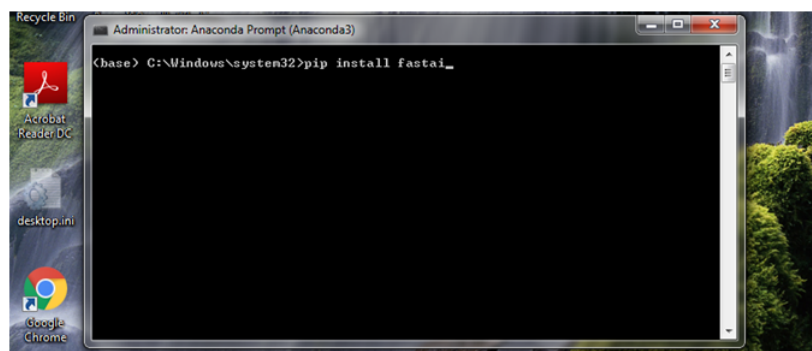


FIGURE A.5: Fastai Installation using Pip as Package Manager.

After installing fastai, it is necessary to RESTART the PC, in order to let your PC find fastai library. After running the Anaconda Navigator (as Administrator)

and Launching the Jupyter Notebook application in the Anaconda Navigator, one can then navigate through the PC and find the fastai course folder, as shown in the figure below:



FIGURE A.6: Navigating Fastai Toolbox for Use in IDE.

The course folder can be downloaded by following the instructions found at '<https://course.fast.ai/>'. If you have saved the fastai course folder on the desktop, then fastai tutorial lesson can be opened in Jupyter notebook by following these folder locations: 'http://localhost:8888/notebooks/Desktop/fastai/course-v3/nbs/dl1/00_notebook_tutorial.ipynb'. It is good for a start-up and one can then build deep learning models and start the Training process as per the instructions in section 5.2.3 Training in fastai.

A.4 Installation and Import of other Required Modules

Before proceeding further, it is necessary to download following modules from the start using either of the package manager (pip or conda), in the Anaconda Prompt:

- latest version of numpy (using either pip or conda).
- latest version of matplotlib (using either pip or conda).
- pypardiso (using conda only).
- cv2 (using conda only).

- `scipy` (using either `pip` or `conda`).
- `skimage` (using either `pip` or `conda`).
- `scikit.learn` (using either `pip` or `conda`)

It is better that the above mentioned modules should be imported in python from the very start as follows:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from fastai import *
from fastai.vision import *
from PIL import Image
from fastai.callbacks.hooks import *
from fastai.utils.mem import *
from fastai.basics import *
import torch
from torch import nn
import cv2
from pypardiso import spsolve
import scipy
import skimage
import imageio
```

Appendix B

Labeling Function for KITTI

Dataset

Labelling function for KITTI Dataset

```
def Labelling - Function(x) :
```

```
head-tail-1 = os.path.split(x); Tail-0000000005.pngparent-1 = x.parent; ; ; incl-dataparent-1 = head-tail-1[0]
```

```
head-tail-2 = os.path.split(parent-1); Tail-dataparent-2 = parent-1.parent; ; ; incl-image-02parent-2 = head-tail-2[0]
```

```
head-tail-3 = os.path.split(parent-2); Tail-image-02parent-3 = parent-2.parent; ; ; incl-2011-09-26-drive-0001-syncparent-3 = head-tail-3[0]
```

```
headtail-4 = os.path.split(parent-3); Tail-2011-09-26-drive-0001-syncparent-4 = parent-3.parent; ; ; incl-trainingparent-4 = head-tail-4[0]
```

```
head-tail-5 = os.path.split(parent-4); Tail-trainingparent-5 = parent-4.parent; ; ; incl-KITTI-Datasetparent-5 = head-tail-5[0]
```

```
if head-tail-5[1] == "training" and head-tail-3[1] == "image-02" :
```

```
child-4 = os.path.join(parent-5, "Y-training", "")
```


$child - 3 = os.path.join(child - 4, head - tail - 4[1], "")$

$child - 2 = os.path.join(child - 3, "proj - depth", "")$

$child - 1 = os.path.join(child - 2, "velodyne - raw", "")$

$child - 0 = os.path.join(child - 1, head - tail - 3[1])$

$child = os.path.join(child - 0, head - tail - 1[1])$

$final - path = child$

$elif head - tail_5[1] == "training" and head - tail - 3[1] == "image - 03" :$

$child - 4 = os.path.join(parent - 5, "Y - training", "")$

$child - 3 = os.path.join(child - 4, head - tail - 4[1], "")$

$child - 2 = os.path.join(child - 3, "proj - depth", "")$

$child - 1 = os.path.join(child - 2, "velodyne - raw", "")$

$child - 0 = os.path.join(child - 1, head - tail - 3[1])$

$child = os.path.join(child - 0, head - tail - 1[1])$

$final - path = child$

$if head - tail - 5[1] == "validation" and head - tail - 3[1] == "image - 02" :$

$child - 4 = os.path.join(parent - 5, "Y - validation", "")$

$child - 3 = os.path.join(child - 4, head - tail - 4[1], "")$

$child - 2 = os.path.join(child - 3, "proj - depth", "")$

$child - 1 = os.path.join(child - 2, "velodyne - raw", "")$

$child - 0 = os.path.join(child - 1, head - tail - 3[1])$

$child = os.path.join(child - 0, head - tail - 1[1])$

$final - path = child$

elif head - tail - 5[1] == "validation" and head - tail - 3[1] == "image - 03" :

child - 4 = os.path.join(parent - 5, "Y - validation", "")

child - 3 = os.path.join(child - 4, head - tail - 4[1], "")

child - 2 = os.path.join(child - 3, "proj - depth", "")

child - 1 = os.path.join(child - 2, "velodyne - raw", "")

child - 0 = os.path.join(child - 1, head - tail - 3[1])

child = os.path.join(child - 0, head - tail - 1[1])

final - path = child

return final - path