Autonomous Corridor Navigation of a Quadrotor using Deep Learning

A Project Report

submitted by

ASHUTOSH SINGH (AE13B051)

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY & MASTER OF TECHNOLOGY

(Dual Degree)

DEPARTMENT OF AEROSPACE ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **"Autonomous Corridor Navigation of a Quadrotor using Deep Learning"**, submitted by **Ashutosh Singh**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology and Master of Technology (Dual Degree)**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Joel George M
Research Guide
Assistant Professor
Dept. of Aerospace Engineering
IIT Madras, 600 036

Place: Chennai

Date: 1st May 2018

ACKNOWLEDGEMENTS

I wish to express my sincere thanks to my project guide Dr. Joel George for providing indispensable guidance throughout the course of this project. I am grateful to him for always steering me in the right direction and providing me valuable insights and equipment required for the project. His moral support, patience, immense knowledge and continuous encouragement motivated me to complete the work successfully. I would also like to thank research scholars, senior students and batch mates at IIT Madras for providing valuable assistance throughout this project. Last but not the least, I must express my profound gratitude to my friends and family for providing me with unfailing support throughout my stay at IIT Madras.

ABSTRACT

KEYWORDS: Depth estimation; Corridor navigation; Kinect data collection; Convolutional neural networks; Indoor navigation, AR Drone 2.0 automation.

Autonomous navigation for large Unmanned Aerial Vehicles (UAVs) is fairly straightforward, as expensive sensors and monitoring devices can be employed. In contrast, navigation remains a challenging task for Micro/Mini Aerial Vehicles (MAVs) which operate at low altitude in cluttered environments. Unlike large vehicles, MAVs can only carry very light sensors, such as cameras, making autonomous navigation through obstacles much more challenging. In this report, a system has been described that can be used for navigation of MAVs using only a single RGB camera to perceive the environment.

One of the most important cues for navigation is depth which is used by humans on day to day basis to navigate in both simple outdoor as well as highly cluttered indoor environments. Humans use stereo vision to estimate depth which is a relatively simpler problem as compared to depth estimation from a monocular image but the advent of deep learning has completely revolutionized the world of computer vision by achieving better accuracy than humans in many tasks like image recognition.

This report deploys different deep learning architectures on NYU RGB-D dataset (Nathan Silberman and Fergus, 2012) to estimate depth from an RGB image which is later used for navigation of drone in indoor environment like corridor. In this project, RGB-D data is also collected using Microsoft Kinect from specific indoor environments consisting of corridor to train a network which can more accurately predict depth in a corridor as compared to a network which is trained on generalized indoor environments.

In this report, an algorithm is proposed for smooth and autonomous navigation of the drone along a corridor. The algorithm uses the model weights and live camera feed from a drone to make control decisions. This algorithm has been successfully tested on

tested on AR Drone 2.0

In the end, a Graphical user interface has been developed in python which allows a user to easily create, save, modify and train a network on RGB-D dataset with the capability of live error rate visualization. The GUI also provides tools to visualize and save the predictions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

RGB	Red Green Blue
RGB-D	Red Green Blue - Depth
UAV	Unmanned Aerial Vehicle
MAV	Micro/Mini Aerial Vehicle
FOV	Field of View
IR	Infrared
MSE	Mean Squared Error
MAE	Mean Absolute Error

NOTATION

r	Radius, m
α	Exponential smoothing factor
β	Flight path in degrees
(f_x, f_y)	Focal length in pixels
(c_x, c_y)	Optical center (the principal point), in pixels
(p_x, p_y)	Pixel size in world units
s	Skew coefficient
n	Number of training data
Y	Ground truth
$\hat{Y}$	Predicted output
p	Probability of predicted class

CHAPTER 1

INTRODUCTION

1.1 Motivation

Unmanned Aerial Vehicles had humble beginnings and were used primarily for delivering bombs, or as aerial targets. They have now evolved into complex systems, and are being used in a wide variety of fields, most extensively in combat and for reconnaissance not only in outdoor but indoor environments as well. Most of the indoor military operations are conducted in unknown hostile environments and unlike outdoor environments it is very difficult to gain information and map the indoor environment beforehand using extrinsic methods like satellite imaging. Direct contact with hostile forces in such environments can lead to huge loss of life and property. Hence, there is a need of UAV which is capable of flying autonomously in highly cluttered indoor environments and sending useful intelligence to a remote station which can allow military to plan and execute the missions successfully.

Operating a UAV in indoor environments brings with it a whole set of new challenges like restrictions in weight, dimensions and maximum altitude. It becomes extremely difficult to place sophisticated sensors on board due to aforementioned limitations. This encourages us to use cheapest and one of most commonly available sensor on drone i.e. camera. This report lays out series of steps explaining how to navigate a drone in corridor using camera only.

1.2 Deep Learning

Deep learning, over the past 5 years or so, has gone from a somewhat niche field comprised of a cloistered group of researchers to being so mainstream that even school kids are talking about it. The swift rise and apparent dominance of deep learning over traditional machine learning methods on a variety of tasks has been astonishing to witness,

and at times difficult to explain. Though the main ideas behind deep learning have been in place for decades, it wasn't until data sets became large enough and computers got fast enough that their true power could be revealed. There is a fascinating history full of successes and failures that goes back to the 1940s.

Consequently, interest in deep learning has sky-rocketed, with constant coverage in the popular media. Deep learning research now routinely appears in top journals like Science, Nature, Nature Methods. Deep learning has conquered Go, learned to drive a car, diagnosed skin cancer and autism, defeated humans in image recognition task and can even hallucinate photo-realistic pictures.

The first breakthrough came when E. Rumelhart *et al.* (1986) showed that neural nets with many hidden layers could be effectively trained by a relatively simple procedure called backpropogation. This allowed neural nets to get around the weakness of the perceptron because the additional layers endowed the network with the ability to learn nonlinear functions. Around the same time it was also shown that such networks had the ability to learn any function, a result known as the universal approximation theorem. These were key developments but still lack of computation power limited the flourish of deep learning during that period.

In 2010, a large database known as Imagenet containing millions of labeled images was created and published by Fei-Fei Li's group at Stanford. This database was coupled with the annual LSVRC, where contestants would build computer vision models, submit their predictions, and receive a score based on how accurate they were. In the first two years of the contest, the top models had error rates of 28% and 26%. Krizhevsky *et al.* entered a submission that halved the existing error rate to 16%. The model combined several critical components that have now become mainstays in deep learning models. Since then deep learning has become a dominant force in image classification, segmentation and labelling tasks.

1.3 Depth Estimation

Depth estimation from images has a long history in computer vision. Fruitful approaches have relied on structure from motion, binocular, and multi-view stereo. However, most of these techniques rely on the assumption that multiple observations of the

scene of interest are available. These can come in the form of multiple viewpoints, or observations of the scene under different lighting conditions. It would we very difficult to get such viewpoints from a single camera mounted on a drone. To overcome this limitation, there has recently been a surge in the number of works that pose the task of monocular depth estimation as a supervised learning problem (Eigen *et al.*, 2013; Godard *et al.*, 2017). These methods attempt to directly predict the depth of each pixel in an image using models that have been trained offline on large collections of ground truth depth data. These methods have shown very promising results for the task of depth estimation in single images. However, most existing approaches require vast quantities of corresponding ground truth depth data for training. Just recording quality depth data in a range of environments is a challenging problem. In Chapter 4 it is shown that algorithm proposed for corridor navigation may not require pixel wise accurate depth estimate but an average estimate of depth over a region is good enough. Hence, multiple smaller networks which require much less computational power both at the time of training and prediction are used for depth estimation. These models are trained both on standard NYU RGB-D v2 dataset (Nathan Silberman and Fergus, 2012) and data collected from Microsoft Kinect.

1.4 Microsoft Kinect 2.0

In recent years, Microsot Kinect has given a great boost to the video game industry by creating an immersive gaming experience using gesture recognition and motion tracking . The Microsoft Kinect sensor allows acquiring RGB, IR and depth images with a high frame rate. Because of the complementary nature of the information provided, it has proved an attractive resource for researchers with very different backgrounds (Pagliari *et al.*, 2015). In summer 2014, Microsoft launched a new generation of Kinect (version 2) in the market, based on time-of-flight technology. This device has become go to tool for indoor RGB-D data collection because of its low cost and very high accuracy. In this project also Kinect v2 has been used for indoor RGB-D data collection.

1.5 AR Drone 2.0

AR.Drone provides an open API game development platform. Due to this open platform, affordability, and wide range of onboard sensory equipment, the AR.Drone has become an increasingly popular tool in research and education (Krajnik *et al.*, 2011). It has been used for experiments with visual-based autonomous navigation, autonomous surveillance, and human-machine interaction. Research in these areas has resulted in third party applications being released, some open source, that extend the official capabilities of the drone. Due to aforementioned reasons AR.Drone has huge online community which makes experimentation really easy. Hence, all the flight testing in this project is done on AR Drone 2.0 platform.

1.6 Autonomous Navigation

Since the last decade deep network have been used extensively in aerial robotics for eg Maturana and Scherer (2015) proposed an autonomous UAV landing system, where deep learning is only used to classify terrain, Tai *et al.* (2016) proposed a deep network solution towards obsatcle avoidance in indoor environments by providing controls to steer in five directions. Although in the method proposed by Tai *et al.* (2016), network directly gives navigation direction, it is still limited as drone can move only in 5 directions from any given state. In this project decision making is done in two steps, firstly depth is estimated using deep learning and then navigation direction is computed through algorithm described in Chapter 4. This approach does not limit the motion of drone in fixed number of directions and hence is more likely to provide optimal path for navigation.

1.7 Organization

The report is organized as follows:

Chapter 2 describes different model used for estimating depth from an RGB image. It starts by discussing the structure of deep neural network. This is followed by a brief discussion on performance metric and optimization algorithm.

Chapter 3 describes the working principle of Microsoft Kinect. This section gives the specifics of how data is collected from kinect and pre-processed before feeding into a neural network. A brief discussion on RGB-D data registration in also provided.

Chapter 4 discusses the algorithm used for autonomous corridor navigation. This section also gives the specifics of experimental setup to test the algorithm using AR Drone. In the end a brief discussion about time complexity of algorithm is also provided.

Chapter 5 discusses about the layout of Graphical User Interface for estimating depth. It then continues to explain functionality of each sub frame present in the main GUI window.

Chapter 6 summarizes the project by providing a discussion on results and future prospects.

CHAPTER 2

Depth Estimation

2.1 Introduction

This chapter walks through the basic components of a deep neural network. Architecture of different models used to estimate depth and their results are also summarized. For all the models except Model 1, NYU RGB-D v2 (Nathan Silberman and Fergus, 2012) is used for training and implementation is done using keras library (Chollet, 2015) in python. A sample code for the same is given in A.

2.2 Convolutional neural networks

Convolutional Neural Networks are similar to ordinary Neural Networks (figure 2.1).Like a simple fully connected network, they are also made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and then follows it with a non-linearity. The whole network expresses a single differentiable loss function: from the raw image pixels on one end to output at the other.

The major difference is that ConvNet architectures make the explicit assumption that the inputs are images, which allows to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the network which helps in preventing over fitting.

Architecture

Neural Networks receive an input (a single vector), and transform it through a series of hidden layers(figure 2.1). Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections.

The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores.

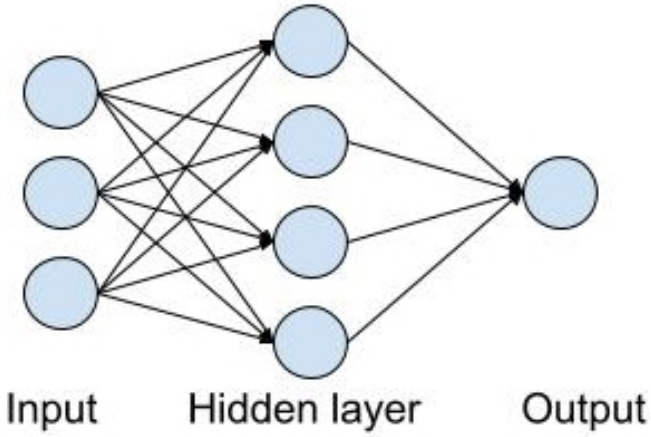

Figure 2.1: Fully connected neural network

Regular Neural Nets don’t scale well to full images. For a small image of size $32 \times 32 \times 3$ (32 wide, 32 high, 3 color channels), a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. $200 \times 200 \times 3$, would lead to neurons that have 200*200*3 = 120,000 weights. Clearly full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting and training will become slower.

Convolutional Neural Networks (figure 2.2) take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, an image the volume has dimensions $32 \times 32 \times 3$ represents width, height, depth respectively. The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. This will not only decrease the total number of parameters drastically but also makes use of the spatial structure of an image.

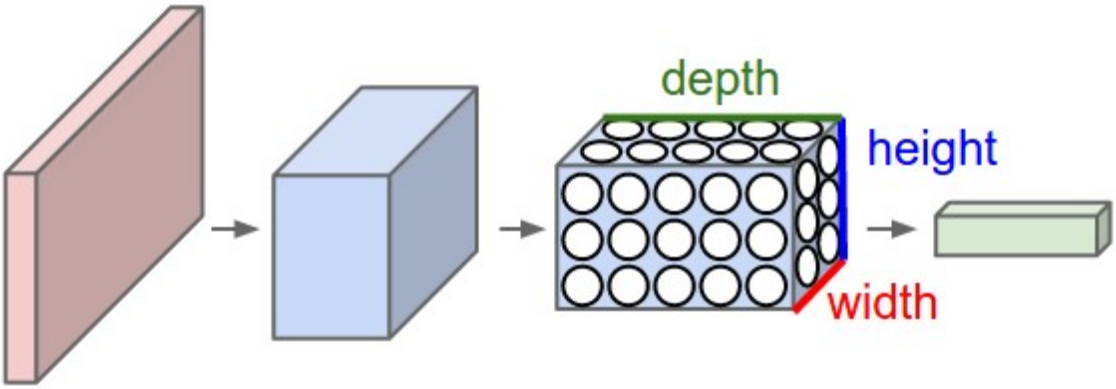

Figure 2.2: Convolutional neural network (*source: cs231n.stanford.edu*)

2.3 Loss function

The function that is used to compute the error is known as Loss Function. Different loss functions will give different errors for the same prediction, and thus have a considerable effect on the performance of the model. The loss functions used in this project are listed below.

Mean squarred error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.1}$$

Mean absolute error

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i| \tag{2.2}$$

Binary crossentropy

$$crossentropy = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \log p_i + (1 - Y_i)\log(1 - p_i)) \tag{2.3}$$

2.4 Backpropogation

Backpropogation algorithm was proposed by Rumelhart *et al.* (1986). It is a method used in artificial neural networks to calculate the gradients that is needed in the calcula-

tion of the weights to be used in the network. It is commonly used to train deep neural networks.

Assumptions

Loss function must satisfy the following conditions for applying backpropogation algorithm:

- The first is that it can be written as an average $E = \frac{1}{n}\sum_x E_x$ over loss function E_x, for individual training examples, x. The reason for this assumption is that the backpropagation algorithm calculates the gradient of the loss function for a single training example, which needs to be generalized to the overall loss function.
- The second assumption is that loss function can be written as a function of the outputs from the neural network.

Algorithm

Phase-1 Propogation

1. Propagation forward through the network to generate the output value(s)
2. Calculation of the cost (loss term)
3. Propagation of the output activations back through the network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

Phase-2 Weight update

1. The weight's output delta and input activation are multiplied to find the gradient of the weight.
2. A ratio (learning rate) of the weight's gradient is subtracted from the weight.

2.5 Adam optimizer

Kingma and Ba (2014) introduced an optimization algorithm called Adam. Unlike Stochastic gradient descent algorithm it does not maintains a constant learning rate but learning rate is adapted based on the average first moment and second moment of the gradients. Currently it has become the most popular optimizer because of its computational efficiency and little memory requirements.

2.6 Model-1

In this section, a binary classification model is built to classify the position of a pole with respect to a reference point. Data is divided into two classes C_1 and C_2 indicating whether pole is at a distance less than $1m$ or more than $1m$ from the reference point. Figure 2.3 shows the network architecture. It is clearly evident from figure 2.4 that both validation and training loss converge. Also, it can be seen in table 2.1 that both training and testing accuracy are very high but this might be due to the reason that data is collected from limited sources. Hence, results may not generalize well to a completely new out of sample data.

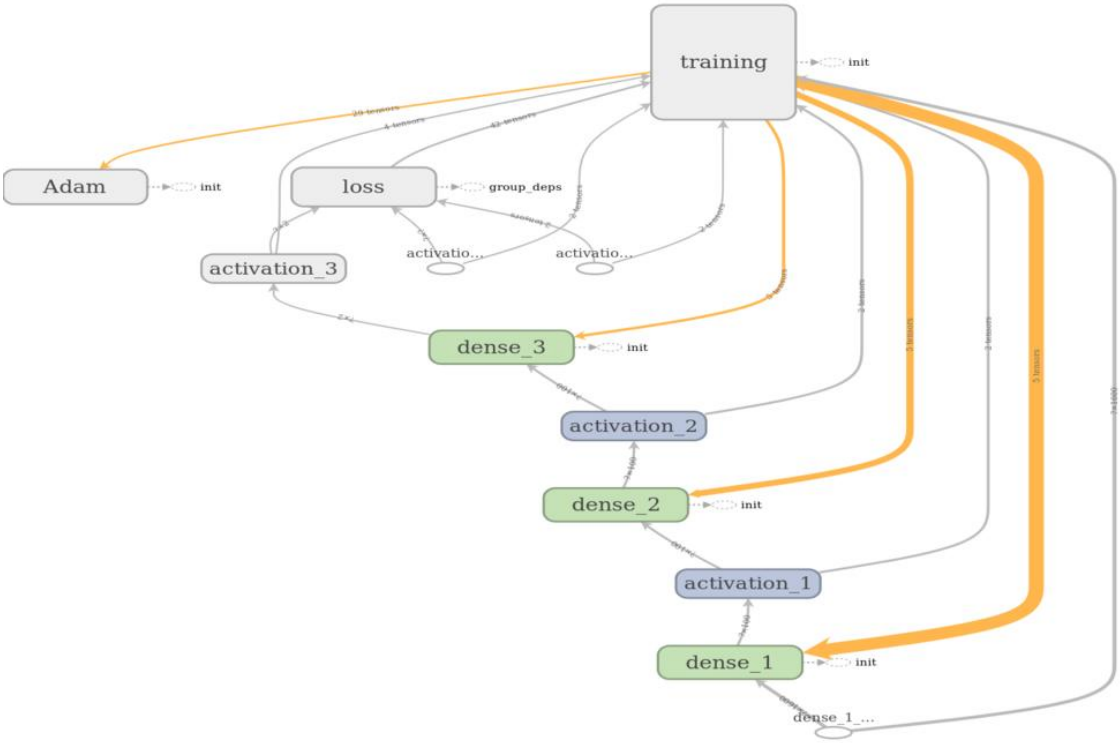

Figure 2.3: Network architecture

Training data	4000
Testing data	1000
Training accuracy	99.80%
Testing accuracy	99.60%

Table 2.1: Model-1 Performance metrics

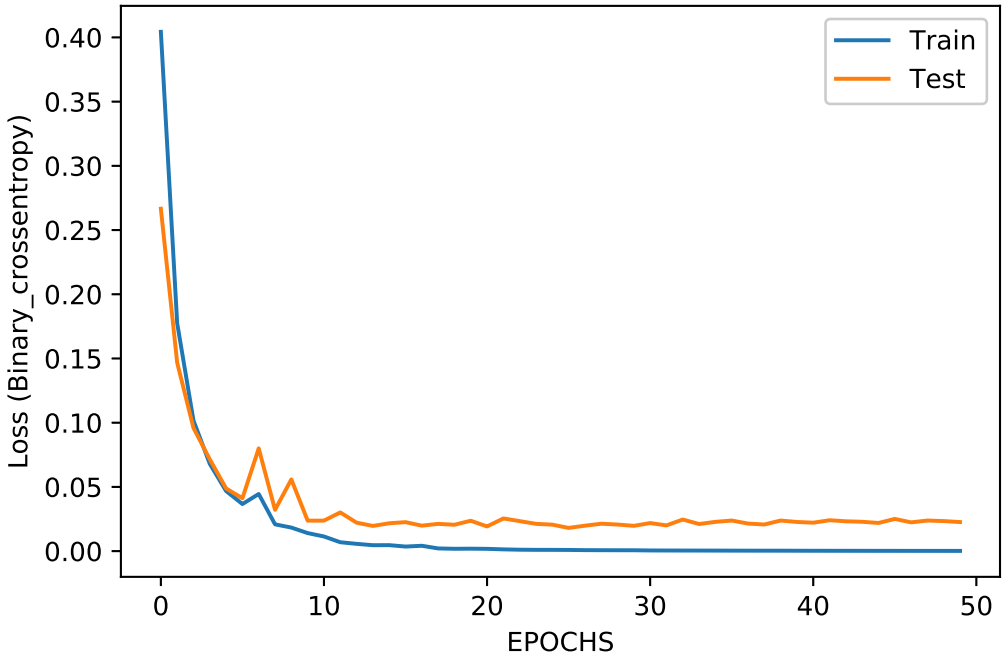

Figure 2.4: Model-1 loss

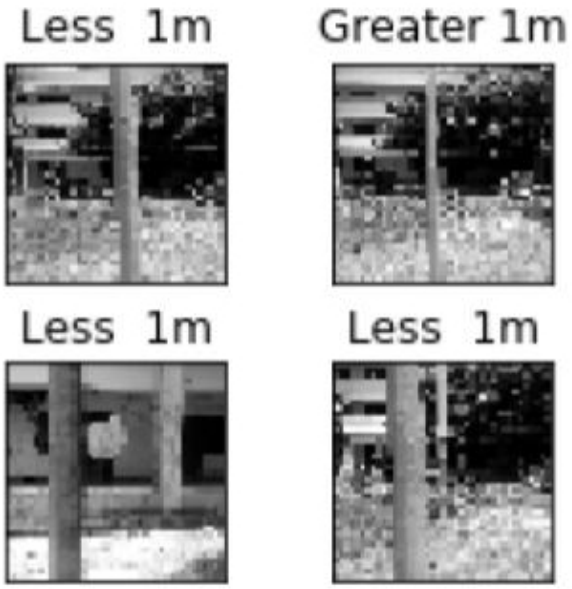

Figure 2.5: Model-1 predictions

2.7 Model-2

In this model-2 NYU v2 RGB-D data is used for training the network. Instead of using complete image for training, here each image is divided into 9 segments of same size and then average depth is predicted for each segment using the architecture shown in figure 2.6.

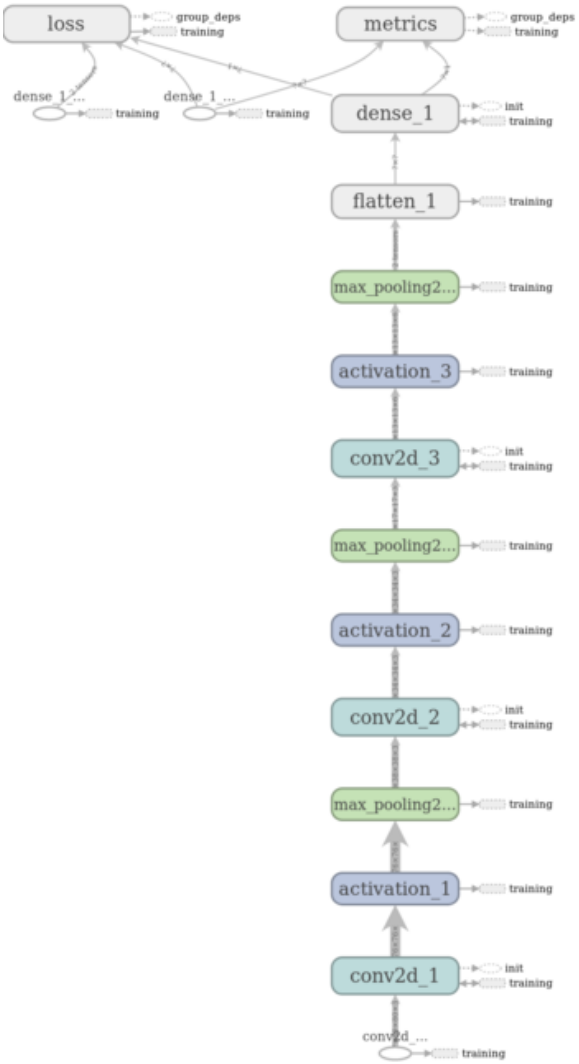

Figure 2.6: Model-2 architecture

From figure 2.7 it can be seen that model loss is not converging also the predictions (2.8) are not close to the ground truth. This is due to the loss of important information (like occlusion) because of segmentation.

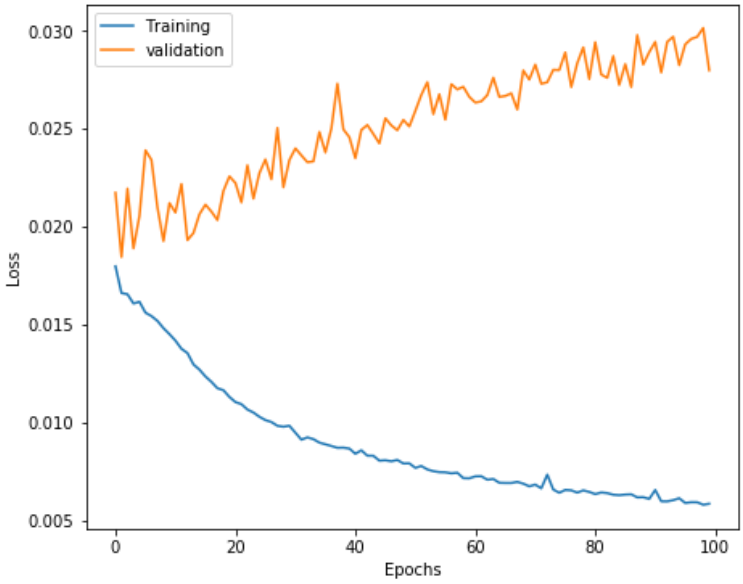

Figure 2.7: Model-2 loss

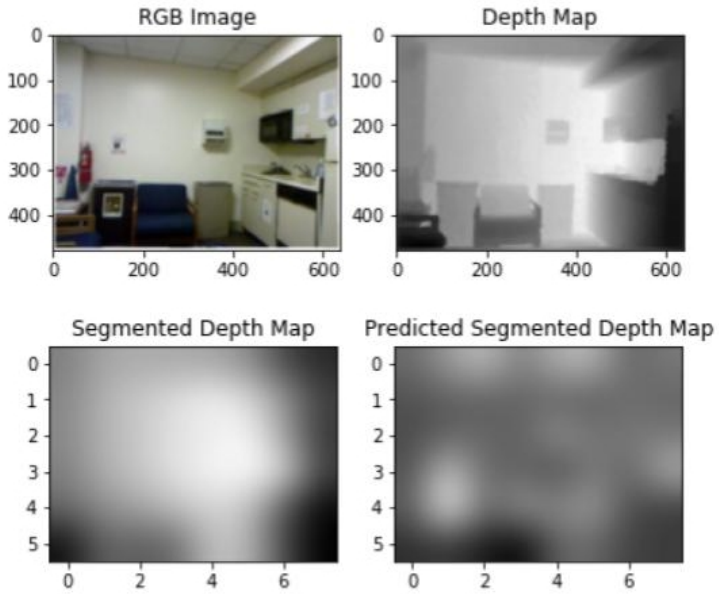

Figure 2.8: Model-2 predictions

Training data	1305
Testing data	145
Training MSE	0.006
Testing MSE	0.027

Table 2.2: Model-2 Performance metrics

2.8 Model-3

In this model, instead on segments complete image is used as input and ouput is pixel wise depth. Model architecture is exactly the same as used in previous model (figure 2.6). It can be seen from figure 2.9 predictions are much closer to the ground truth when compared to the previous model. Taking complete image certainly helped the network to learn new features based on occlusion and perspective which in turn improved the predictions.

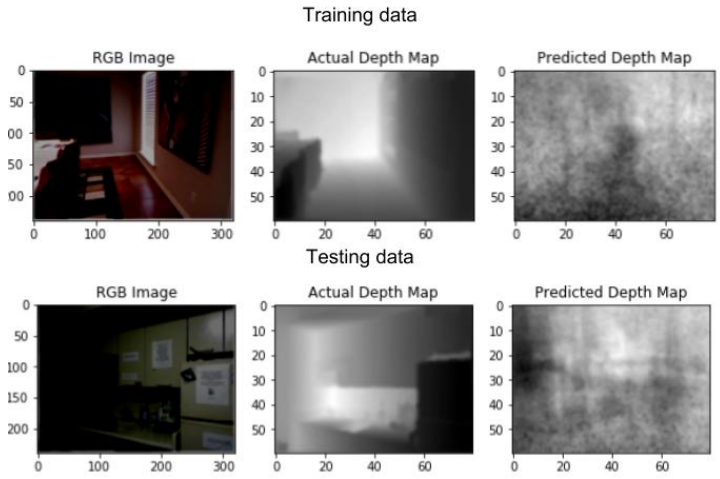

Figure 2.9: Model-3 loss

Training data	1305
Testing data	145
Training MSE	0.004
Testing MSE	0.029

Table 2.3: Model-3 Performance metrics

2.9 Model-4

In this model batch normalization and dropout is used after each convolutional layer as shown in figure 2.10. Also in this model while pre-processing depth was locally scaled between 0 to 1 as compared to global scaling in the previous models. This resulted in drastic improvements in predictions as shown in figure 2.12.

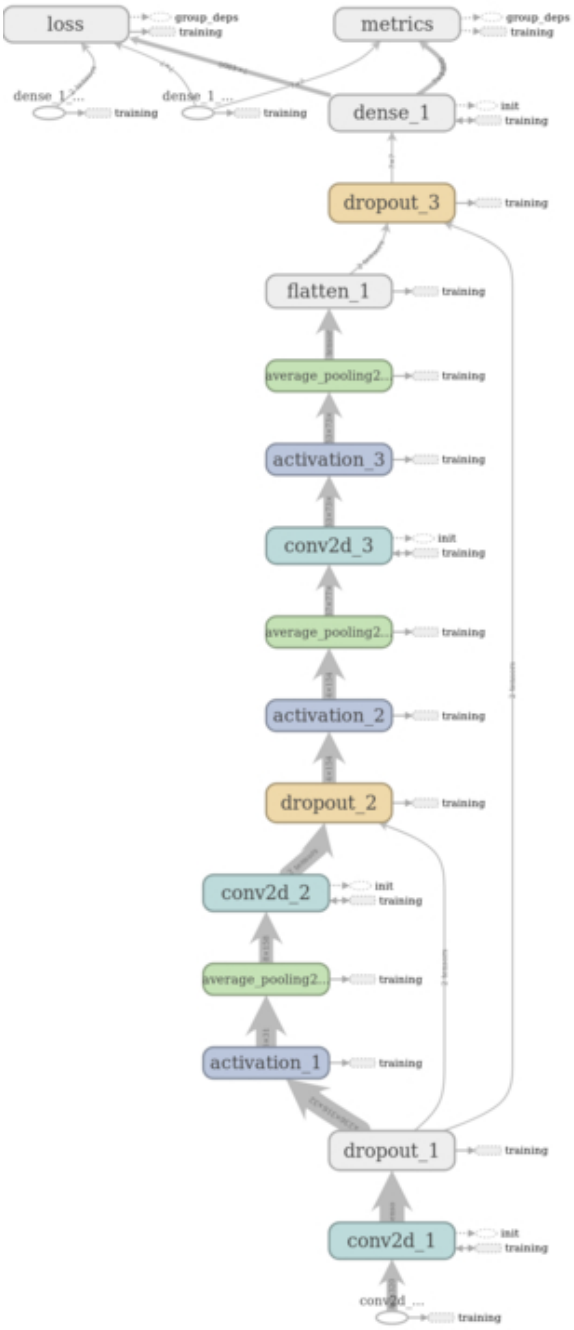

Figure 2.10: Model-4 architecture

Training data	1305
Testing data	145
Training MSE	0.003
Testing MSE	0.023

Table 2.4: Model-4 Performance metrics

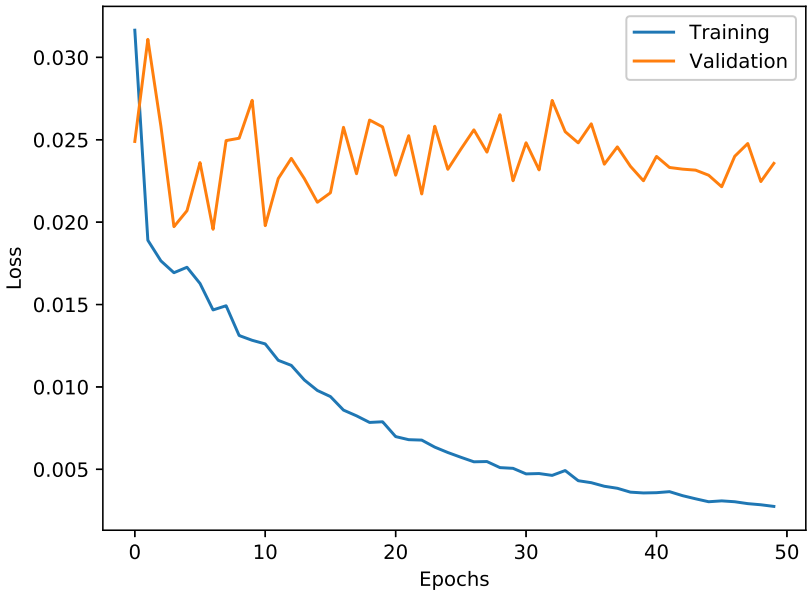

Figure 2.11: Model-4 Loss

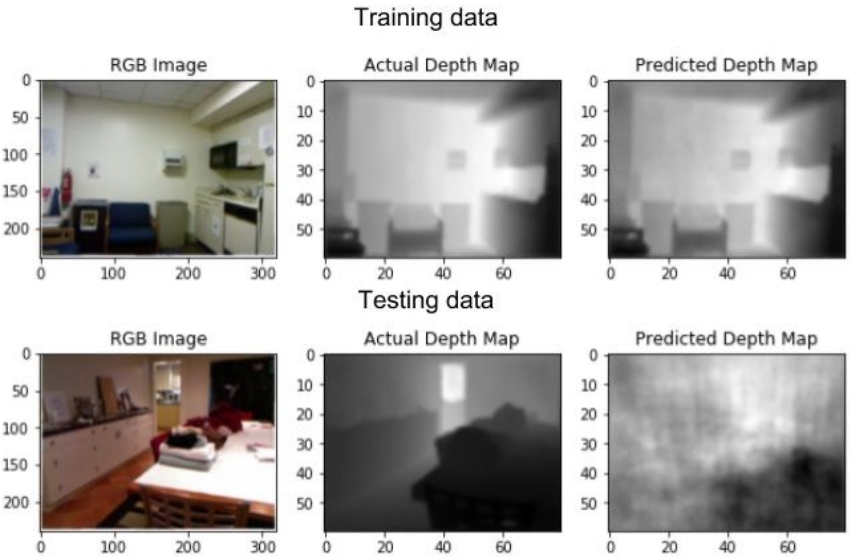

Figure 2.12: Model-4 predictions

2.10 Model-5

In this model, fully convolutional network is used which retains the spatial location of each pixel until the last layer and reduces the number of parameters drastically. This kind of architecture is commonly used in problems like image segmentation. It can be observed from figure 2.15 that network is able to segment the images but underperforms in predicting depth when compare to model 4

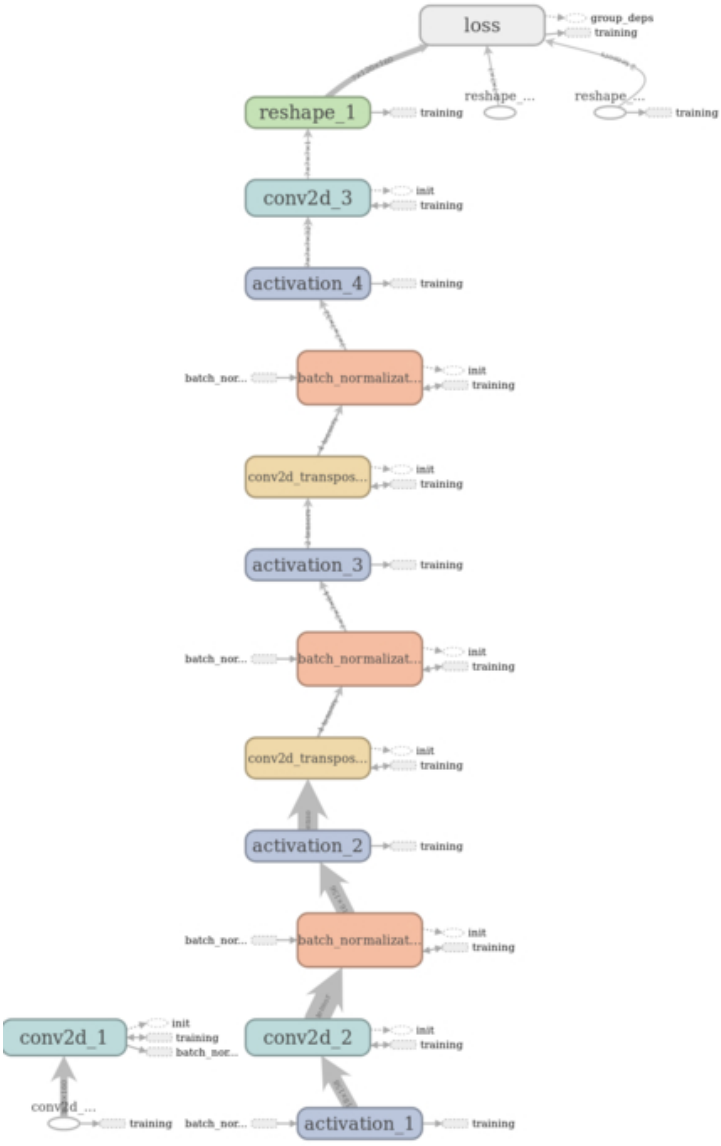

Figure 2.13: Model-5 architecture

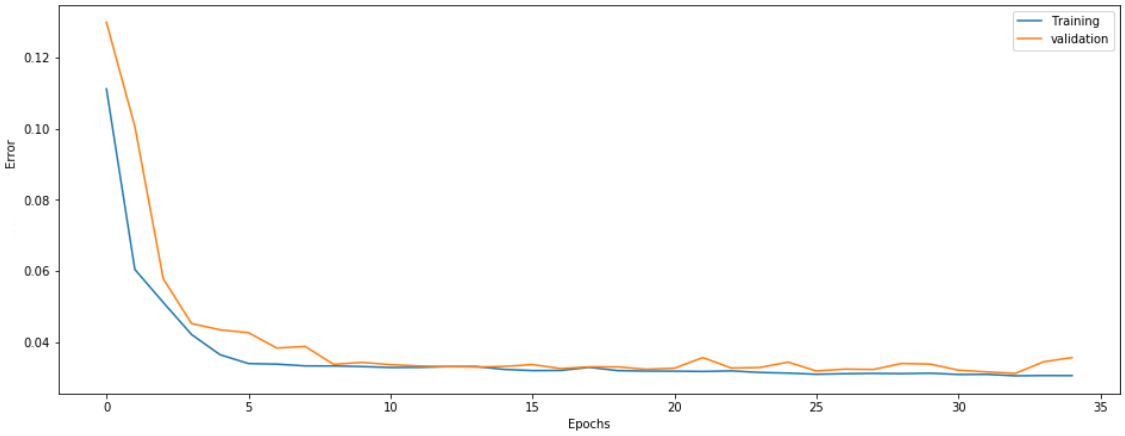

Figure 2.14: Model-5 Loss

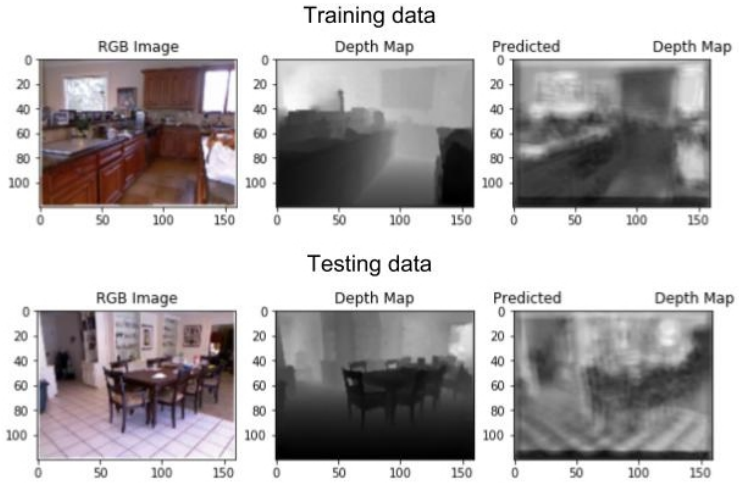

Figure 2.15: Model-5 prediction

Training data	1305
Testing data	145
Training MSE	0.02
Testing MSE	0.03

Table 2.5: Model-5 Performance metrics

2.11 Results

After training weights of the first layer filter of each network were visualized and all of them showed pretty similar pattern as shown in figure 2.16. The significance of these visualization is that they try to mimic the features a filter is trying to learn for example if a filter tries to learn features like orientation then one can easily see that pattern in the visualization. In this case it appears that model is trying to segment different objects at the first layer which should not come as surprise because human eyes also segment the environment into different objects and try to estimate relative depth based on occlusion.

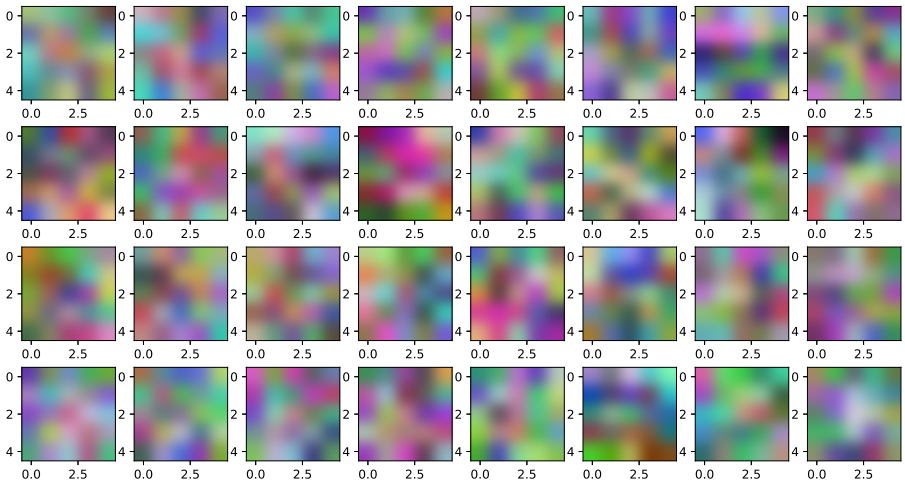

Figure 2.16: Visualization of first layer filters

It was observed that model- 4 gave a decent estimate of depth when tested on the dataset limited to NYU depth v2 but it fails to generalize in other indoor environments. Hence, RGB-D data containing images of corridor is collected in next chapter which can be used to train a network for depth estimation in corridor environment.

CHAPTER 3

Data Collection

3.1 Introduction

Even in this age of data explosion, RGB-D data is difficult to find because of the challenges and cost involved in getting true depth map from a scene. Two of the most commonly used open source RGB-D data are KITTI (Geiger *et al.*, 2013) and NYU v2 (Nathan Silberman and Fergus, 2012).

KITTI data (figure 3.1) was collected using Volkswagen Passat B6 with following equipments mounted on it:

- 1 Inertial Navigation System (GPS/IMU): OXTS RT 3003
- 1 Laserscanner: Velodyne HDL-64E
- 2 Grayscale cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3M-C)
- 2 Color cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3C-C)
- 4 Varifocal lenses, 4-8 mm: Edmund Optics NT59-917

The NYU-Depth V2 data set (3.2) is comprised of video sequences from a variety of indoor scenes as recorded by both the RGB and Depth cameras from the Microsoft Kinect.

The problem with these datasets is that they don't contain images of corridor and models built in the last chapter do not generalize well to images which are not similar to training data. Hence in the following sections methodology for collecting data using Microsoft kinect is layed out.

3.2 Microsoft Kinect

Microsoft's kinect sensor has probably been the sensor with the highest influence on the robotics and mechatronics communities in the last three years. Originally proposed as a

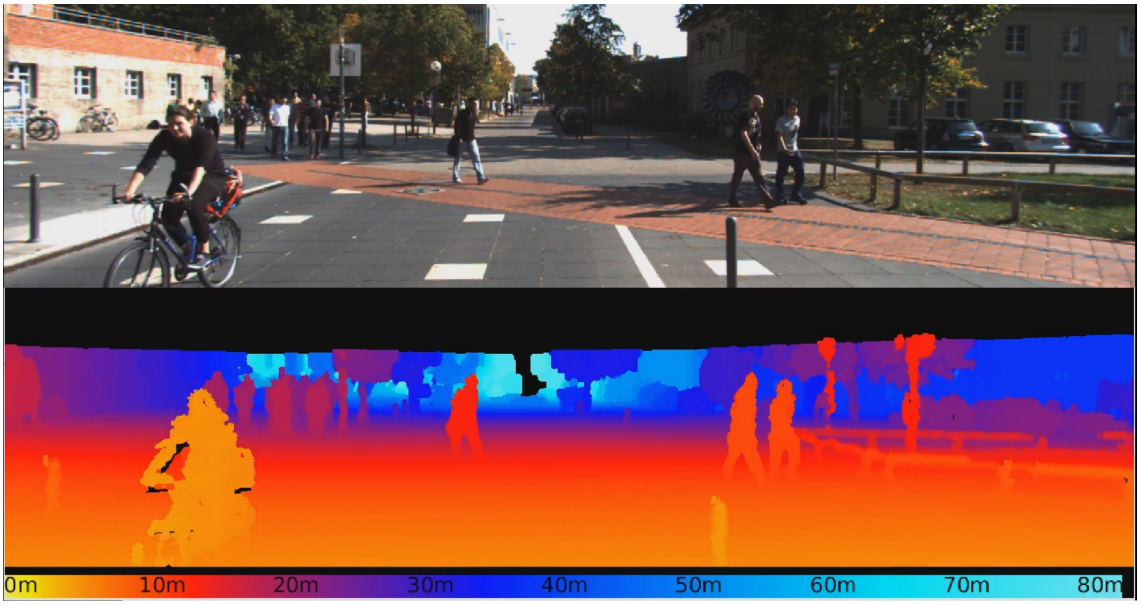

Figure 3.1: KITTI RGB-D data

Figure 3.2: NYU RGB-D data

game controller, many researchers have explored additional applications in health-care (Ning and Guo, 2013), robotics (Machida *et al.*, 2012), video surveillance (Zhang *et al.*, 2012), and the facilitation of communication (Sun *et al.*, 2013). It is primarily the low price of the device that motivates researchers to explore the kinect in applications that have formerly been solved even by non-optical sensors.

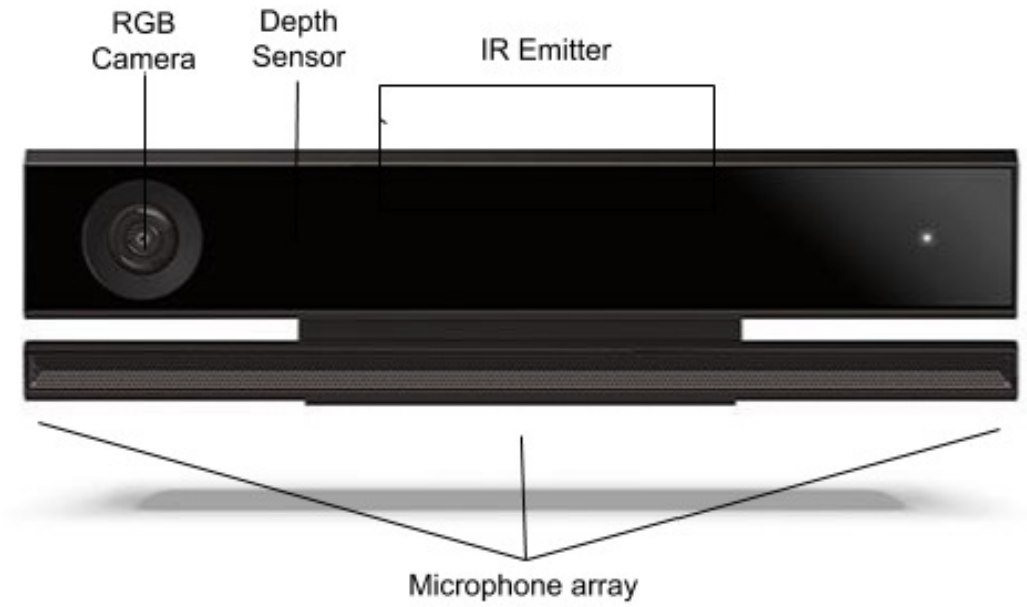

Figure 3.3: Microsoft Kinect 2.0

Working Principle

Kinect 2.0 works on the principle of time of flight. A time-of-flight camera emits light signals and then measures how long it takes them to return. The measurements are as acurate as $1/10,000,000,000$ so as to capture the minuscule time difference. With such measurements, the camera is able to differentiate light reflecting from objects in a room and the surrounding environment. That provides an accurate depth estimation that enables the shape of those objects to be computed. Specifications of Kinect 2.0 are mention in the table 3.1

IR camera resolution	512×424 pixels
RGB camera resolution	1920×1080 pixels
Field of view (RGB)	84×53 degrees
Field of view (depth)	70×60 degrees
Frame rate	30 fps
Operative measuring range	from 0.5 m to 4.5 m
Object pixel size	1.4 mm (@0.5m range) & 12 mm (@4.5m range)

Table 3.1: Kinect 2.0 specifications (Pagliari and Pinto, 2015)

3.3 Camera calibration

Camera calibration estimates the parameters of a lens and image sensor of an image or video camera. One can use these parameters to correct for lens distortion, measure the size of an object in world units, or determine the location of the camera in the scene. These tasks are used in applications such as machine vision to detect and measure objects. They are also used in robotics, for navigation systems, and 3-D scene reconstruction. In the context of this project camera calibration parameters will be used for RGB and depth camera registration as they don't have the same field of view and are not present at the same position as shown in the figure 3.4

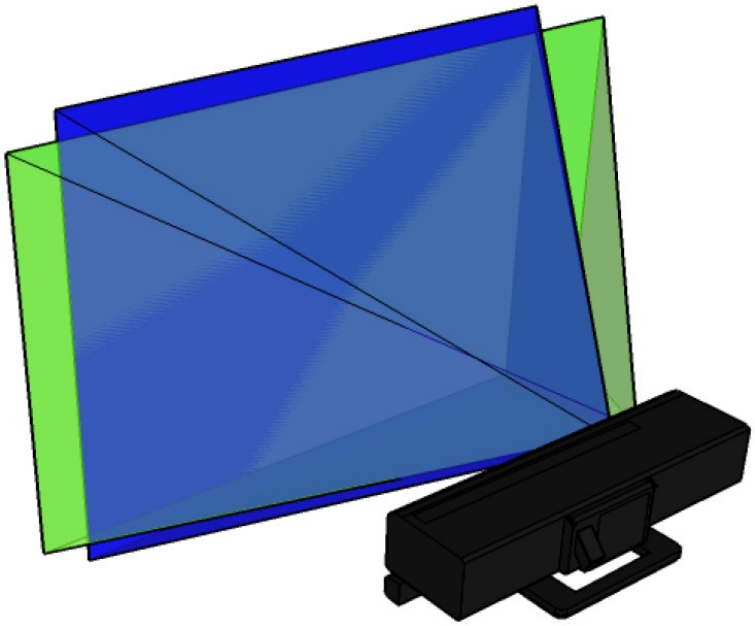

Figure 3.4: RGB (green) and depth camera (blue) FOV (Pagliari and Pinto, 2015)

Camera parameters include intrinsics, extrinsics, and distortion coefficients. To estimate the camera parameters, one need to have 3-D world points and their corresponding

2-D image points. These correspondences can be extracted using multiple images of a calibration pattern, such as a checkerboard. Using the correspondences, one can solve for the camera parameters.

3.3.1 Camera calibration parameters

The calibration algorithm calculates the camera matrix using the extrinsic and intrinsic parameters. The extrinsic parameters represent a rigid transformation from 3-D world coordinate system to the 3-D camera's coordinate system. The intrinsic parameters represent a projective transformation from the 3-D camera's coordinates into the 2-D image coordinates.

3.3.2 Extrinsic parameters

The extrinsic parameters consist of a rotation, R, and a translation, t. The origin of the camera's coordinate system is at its optical center and its x- and y-axis define the image plane.

3.3.3 Intrinsic parameters

The intrinsic parameters include the focal length, the optical center, also known as the principal point, and the skew coefficient. The camera intrinsic matrix, K, is defined as:

$$\begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix}$$

3.3.4 Distortion in camera calibration

The camera matrix does not account for lens distortion because an ideal pinhole camera does not have a lens. To accurately represent a real camera, the camera model includes the radial and tangential lens distortion.

Radial distortion

Radial distortion occurs when light rays bend more near the edges of a lens than they do at its optical center. The smaller the lens, the greater the distortion.

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.1}$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2}$$

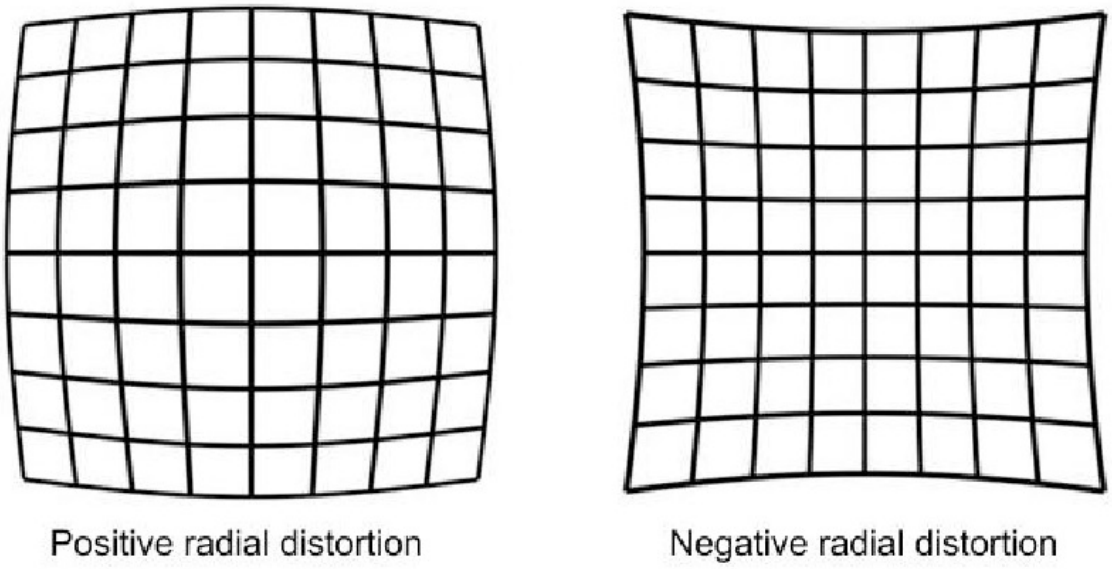

Figure 3.5: Radial distortion (Rojas Vidovic *et al.*, 2012)

Tangential distortion

Tangential distortion occurs when the lens and the image plane are not parallel. The tangential distortion coefficients model this type of distortion.

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.3}$$

$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3.4}$$

For the purpose of removing distortion and RGB-D registration, camera parameters provided by python binding of Microsoft kinect v2.0 pylibfreenect2 is used.

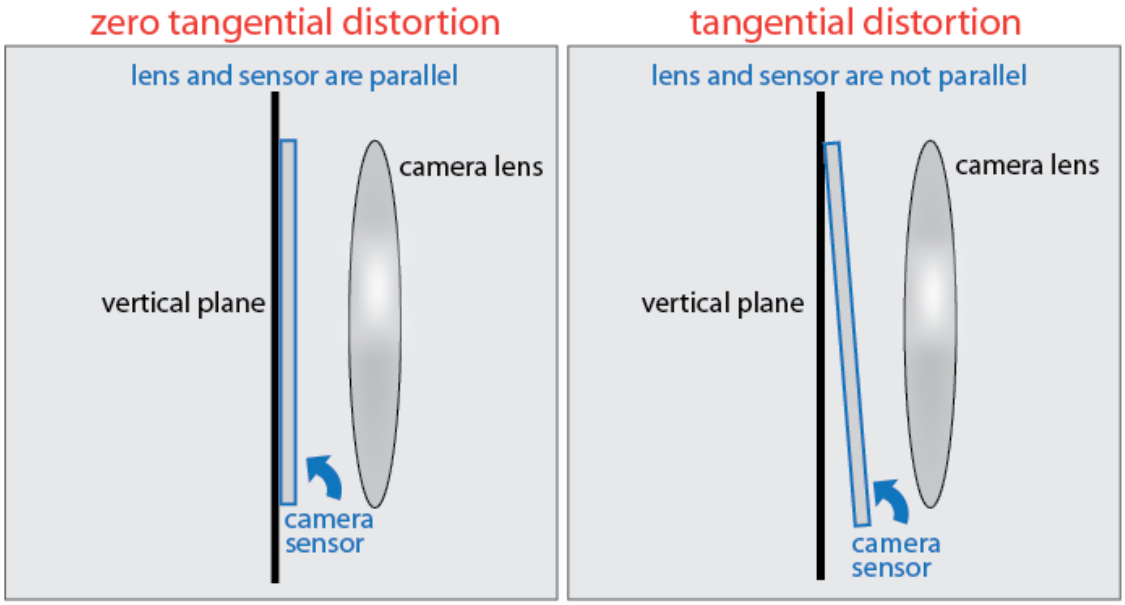

Figure 3.6: Tangential distortion

3.4 Experimental setup

Kinect and workstation are set up on two different platform as shown in figure 3.7. It is ensured that platform of kinect is not fixed and cables have sufficent length so that kinect can be moved freely.

Figure 3.7: Experimental setup

3.5 Procedure

1. Connect kinect USB cable to the system and wait for 5-10 seconds.
2. Start the python code for data collection. (B)
3. Move kinect to span the area and collect as many data points as possible.
4. Terminate the program.

3.6 Results

Figure 3.8 shows raw data. It can be seen that both the images have different dimensions because of different FOV of RGB and depth cameras. Hence, both images are registered using pylibfreenect2 and the resulting images are shown in figure 3.9.

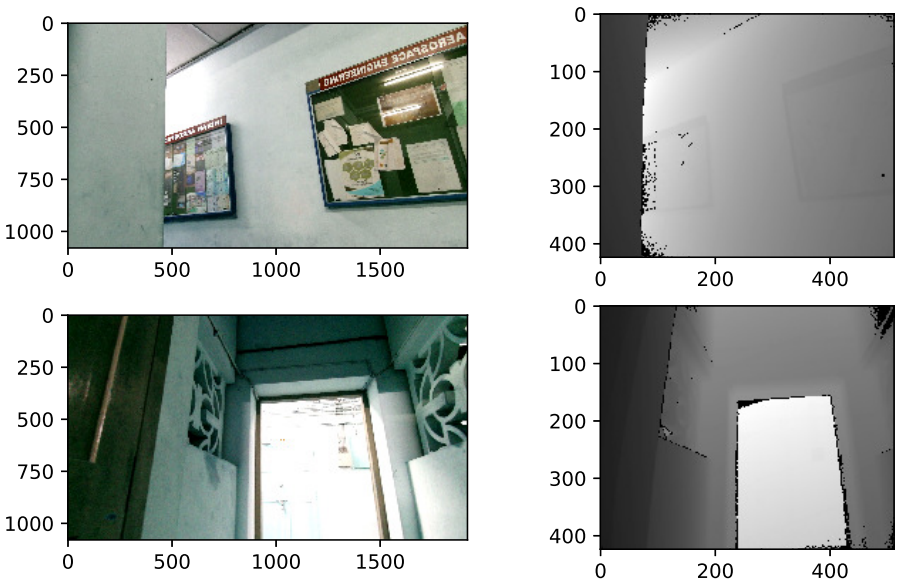

Figure 3.8: Sample Data

3.7 Missing Value Imputation

Missing value may arise due to one of the following reasons:

- Depth discontinuities in the scene.

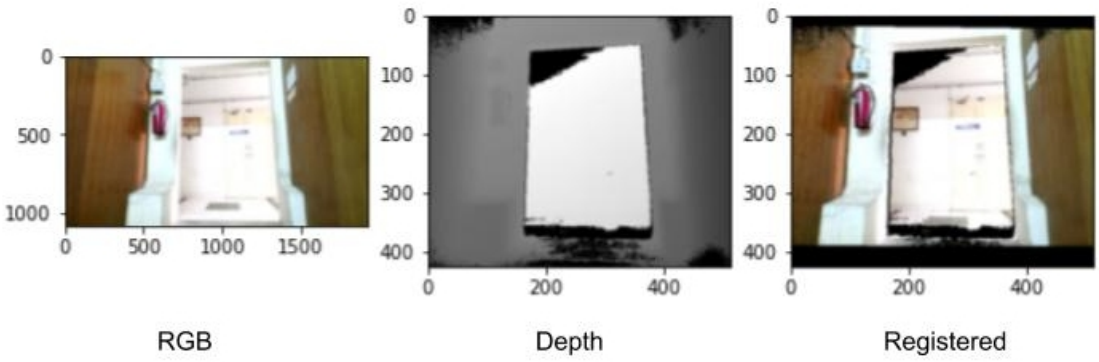

Figure 3.9: RGB-D registration

- Specular surfaces like glass windows deflect infrared light causing missing values
- Light absorbing materials such as black cloth can also cause gaps.

For inpainting missing values code from (Nathan Silberman and Fergus, 2012) is used which based on the colorization scheme provided by Levin *et al.* (2004).

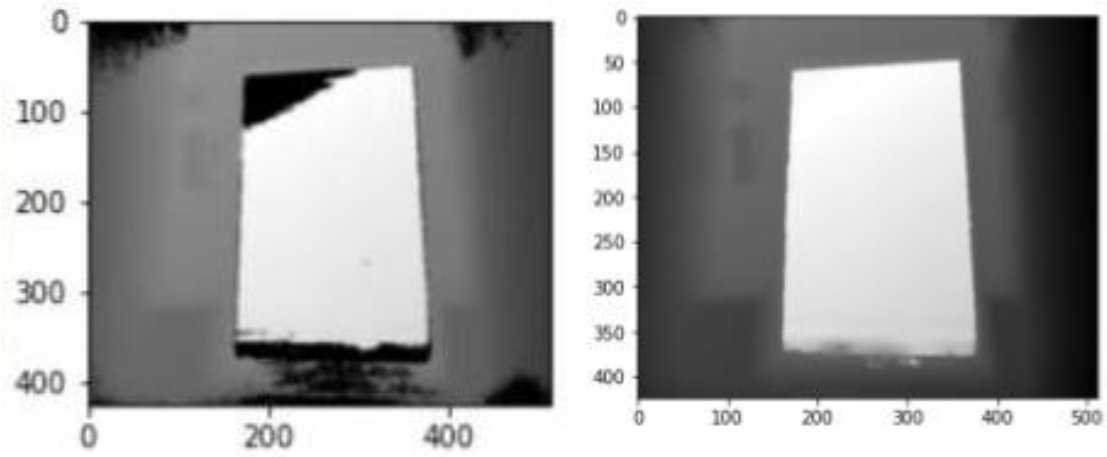

Figure 3.10: Missing value imputation

CHAPTER 4

Autonomous Corridor Navigation

4.1 Introduction

Flying through narrow environments like corridor is challenging task. The margin for error is really small in such tight environments and even a small mistake from a human operator may cause damage to UAV. In this chapter, an algorithm for autonomous corridor navigation has been formulated and tested in a corridor of width 1.5 m.

4.2 Experimental setup

Environment

As already emphasized in introduction test environment for algorithm will be corridor as shown in figure 4.1. It is important to note that corridor should have good lighting conditions so as to ensure proper estimation of the depth map.

AR Drone

For validating the navigation algorithm AR.Drone v2 platform (figure 4.2) is used. It is equipped with a front camera capable of providing video feed at 30 fps and $1280 \times 720px$ resolution and other necessary sensors for autonomous flight stabilization. Other important specifications of drone are mentioned in the table 4.1

Dimensions	$23" \times 23"$
Weight	4 lbs
Battery	1000 mAh
Endurance	12 min

Table 4.1: AR.Drone 2.0 specifications

Figure 4.1: Corridor environment

Figure 4.2: Parrot AR Drone 2.0

Ground station

Core i5 system with 4 GB ram is used for image processing and estimating depth. It gives a frame rate of approximately 0.5 fps. This frame rate is too low for avoiding moving obstacle hence, there are no moving obstacles in the testing environment. However, this algorithm will work for moving obstacle as well if image processing and depth estimation are done using GPU.

4.3 Information flow

Automating AR Drone broadly involves 3 steps:

- Transmitting video data from drone to the system.
- Computing control commands on the system.
- Transmitting control commands back to the drone.

For the case of autonomous corridor navigation, this has been shown as a flowchart in figure 4.3.

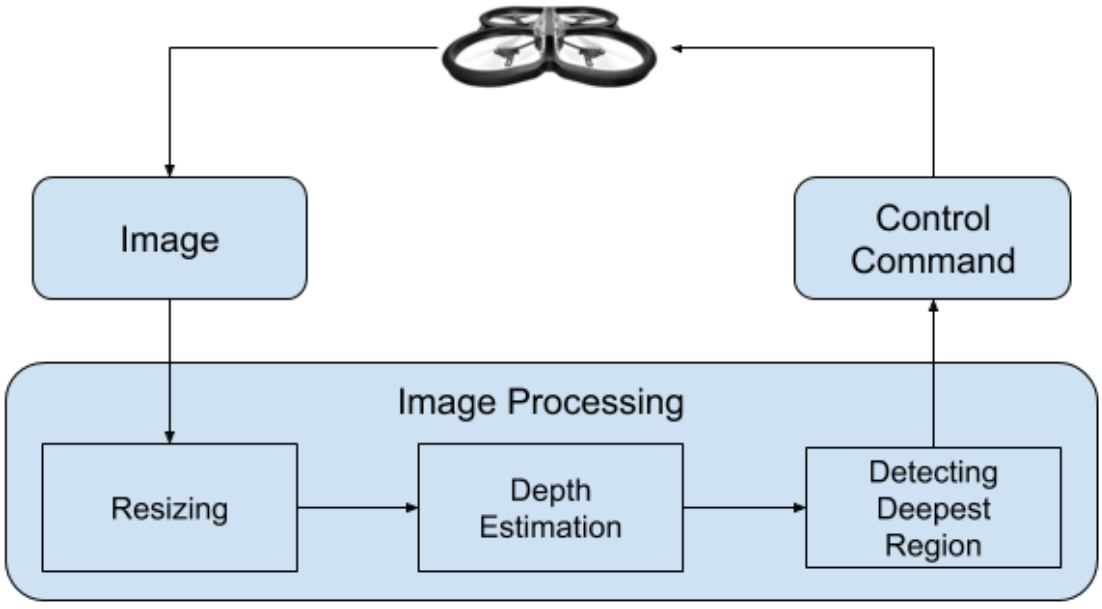

Figure 4.3: Control flow Chart

4.4 Algorithm

4.4.1 Detecting deepest region

Algorithm for corridor navigation is divided into two sub-algorithms. Algorithm 1 computes the deepest region in an image. It initially estimates depth map using neural network models developed in chapter 2. It then slides a rectangular window of fixed dimension across the image calculating average depth of pixels inside it. Finally, the position of window with maximum average depth is returned. Figure 4.4 shows the step by step transformation of an image passed through this algorithm.

Algorithm 1: detectDeepestRegion

```
   Input  : depth, box_shape
   Output: deepest_region
 1 max_depth ← 0;
 2 deepest_region ← 0 ;
 3 while Depth map is parsed do
 4 |   calculate average depth in box;
 5 |   if avg_depth > max_depth then
 6 |   |   max_depth ← avg_depth ;
 7 |   |   deepest_region ← curr_region ;
 8 |   else
 9 |   |   pass;
10 |   end
11 |   shift the box by one unit ;
12 end
```

Figure 4.4: Output of Algorithm 1

Time complexity analysis

For the online implementation of an algorithm, time is a major factor as it will govern the maximum speed of UAV. Two bottlenecks in the algorithm proposed in previous sections are estimating depth map and finding deepest region. As already mentioned

time taken to estimate depth depends on the model architecture and can be improved by using GPU. However, the time complexity of the algorithm to find the deepest region depends on the window size. Figure 4.5 shows time taken to find the deepest region as a function of window size. Y-axis shows the time in seconds and X-axis shows the units by which width and height are reduced starting from (80, 250).

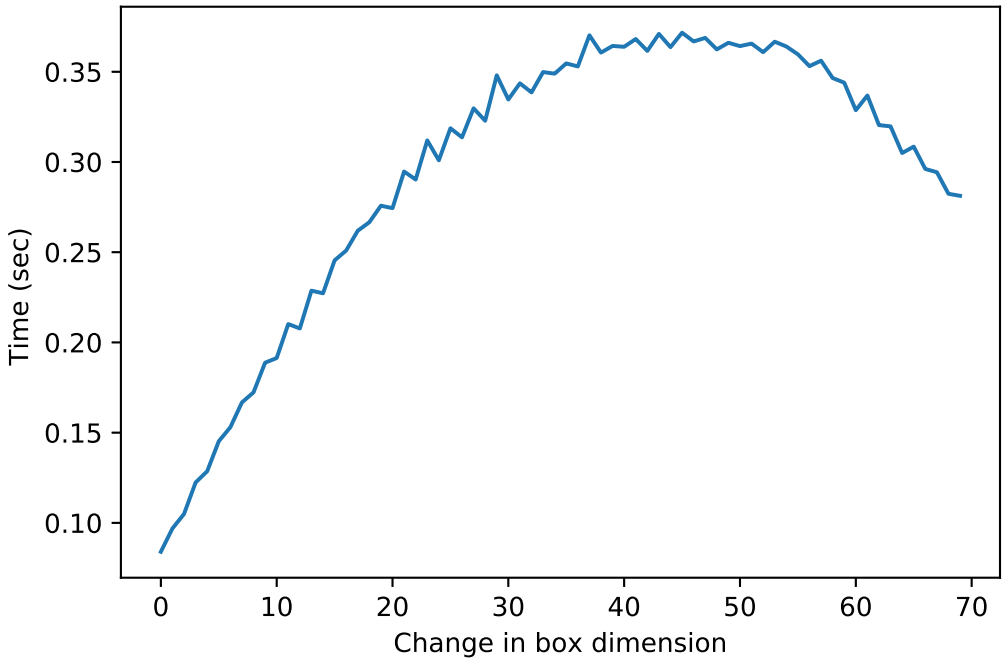

Figure 4.5: Time complexity plot of Deepest region detection

To understand the behavior of plot consider an image of fixed width W and height H and a rectangular window of width w and height h as shown in figure 4.6. Now the number of regions for which average depth will be computed for a window of given size will be equal to $(W - w)(H - h)$ and time taken to compute average depth inside a window will be of the order wh. Hence, the total time taken to find the deepest region for an image as a function of window size will be of the order $(W-w)(H-h)wh$. Now it can be seen that time taken increases initially because time complexity is dominated by $(W - w)(H - h)$ factor i.e. no of times average depth has to be calculated but after a point w and h become very small and pull down the time take to calculate average depth inside the window, reducing the overall time complexity of the algorithm.

Finally, it is important to mention that as the window size is decreased, the position of deepest region will fluctuate more frequently within frames which will make the con-

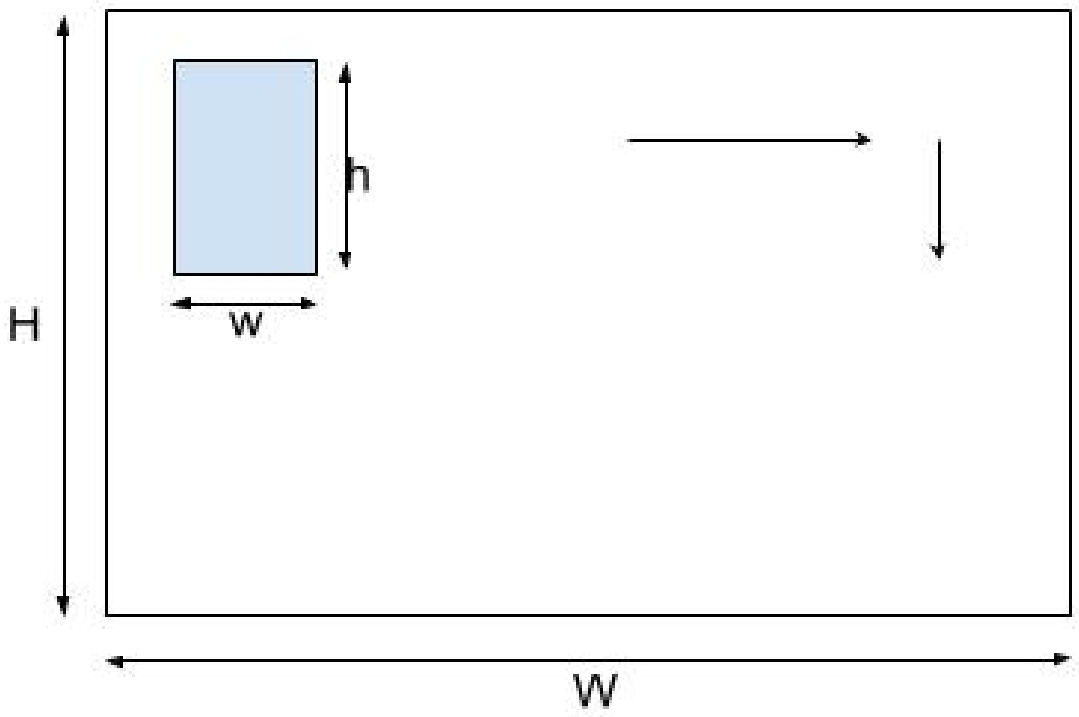

Figure 4.6: Schematic of sliding window algorithm

trol difficult. Hence, window size must be selected depending upon the environment, maximum speed required and manoeuvrability of UAV.

4.4.2 Control decision

Algorithm 2 is used for providing control commands to drone once the deepest region is detected using algorithm 1. If the drone is in the deepest region then it moves forward else it moves left or right depending on the relative location of drone with respect to the deepest region.

4.5 Testing

Figure 4.7 shows successive frames from the front camera of AR Drone while navigating autonomously through the corridor.Green dot represents the location of drone, red box represents the deepest region and on top left, the control command is shown in pink color. It is important to note that there is a significant time difference between each frame due to computational limitations but one can always use GPU to achieve higher

Algorithm 2: corridorNavigation

Input : Drone camera feed
Output: Direction of motion

```
while Drone is online do
    image ← get current image ;
    depth ← predict_depth(image) ;
    deepest_region ← detectDeepestRegion(depth) ;
    if drone is to left of deepest_region then
        move right;
    else if drone is to right of deepest_region then
        move left;
    else
        move forward;
    end
end
```

frame rates.

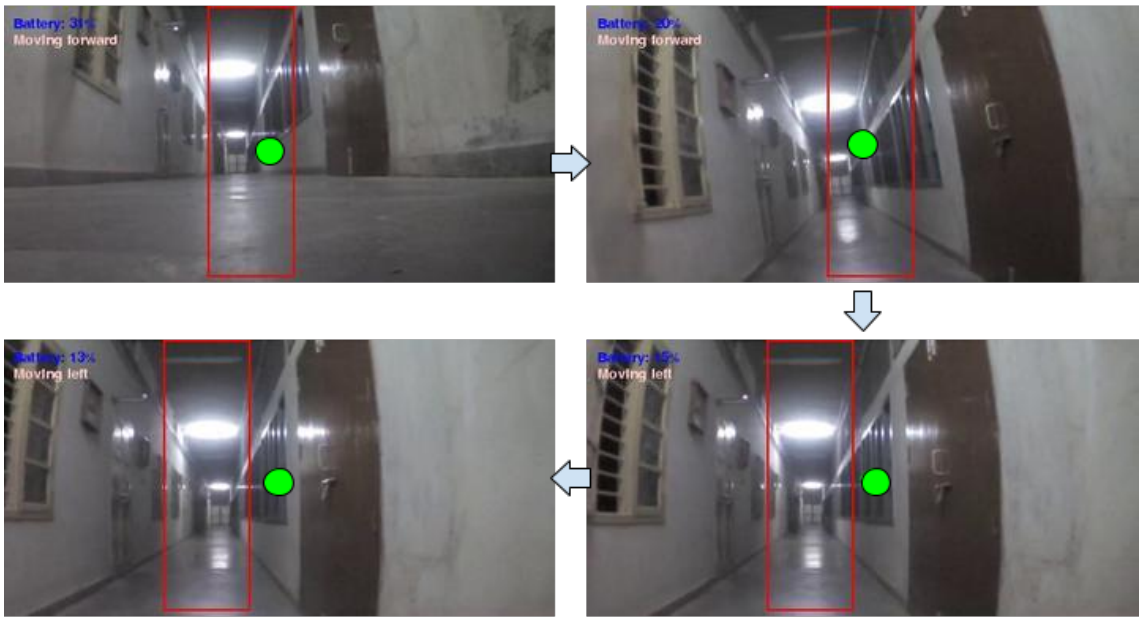

Figure 4.7: Autonomous corridor navigation

4.6 Exponential smoothing

So far no temporal information has been used in the corridor navigation algorithm. The deepest region is estimated independently for each frame. This might result in unsteady control commands due to the presence of noise in the estimated depth. To overcome this limitation exponential smoothing operation can be used on the depth map as shown

in equation 4.1.

$$F_t = \alpha D_t + (1 - \alpha) F_{t-1} \tag{4.1}$$

Here F_t is exponentially smoothed depth at time t, D_t is depth predicted at time t, F_{t-1} is exponentially smoothed depth at time $t - 1$ and α is smoothing factor. α can take values between 0 and 1. The closer α is to 1, current frame contributes more to the depth and closer it is to 0 greater is the contribution of previous frames.

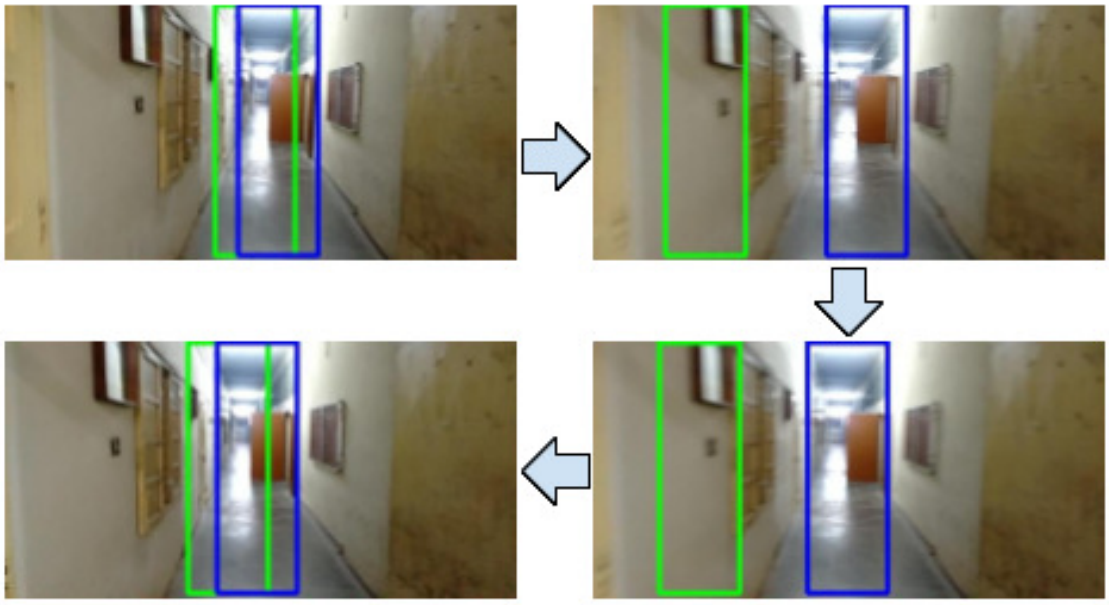

Figure 4.8: Exponential smoothing with $\alpha = 0.5$

Figure 4.8 shows the effect of exponential smoothing on 4 consecutive frames. Blue window shows the deepest region with exponential smoothing and green without exponential smoothing. It is evident that blue window is much more stable as compared to the green window, which fluctuates drastically between frames due to noisy depth estimates.

CHAPTER 5

Graphical User Interface

5.1 Introduction

A graphical user interface (GUI) is a human-computer interface that uses windows, icons and menus and which can be manipulated by a mouse (or any other input event).

A major advantage of GUIs is that they make computer operation more intuitive, and thus easier to learn and use. For example, it is much easier for a new user to move a file from one directory to another by dragging its icon with the mouse than by having to remember and type seemingly arcane commands to accomplish the same task.

Adding to this intuitiveness of operation is the fact that GUIs generally provide users with immediate, visual feedback about the effect of each action. For example, when a user deletes an icon representing a file, the icon immediately disappears, confirming that the file has been deleted (or at least sent to the trash can). This contrasts with the situation for a CLI, in which the user types a delete command (inclusive of the name of the file to be deleted) but receives no automatic feedback indicating that the file has actually been removed.

5.2 Features

- Different neural network architectures can be designed and trained on RGB-D dataset.
- In built tools for basic pre-processing are also given.
- Model weights and architectures can be saved and reloaded.
- Error can be visualized while training.
- Predictions on new out of sample data can also be done.

Figure 5.1: Graphical User Interface

5.3 Layout

Figure 5.1 shows the main window of graphical user interface. It is made from 4 small sub frames. Frame on top left side is used for reading and pre-processing data. Frame on bottom left side is used for designing network. Frame on right hand side is for visualization and bottom most frame is for tracking progress. Functionality of all these frames is described in detail in the coming sections.

5.3.1 Data input frame

This frame takes the path of RGB and depth data as input and reads them into the memory. It also provides basic pre-processing features like scaling, shuffling and resizing.

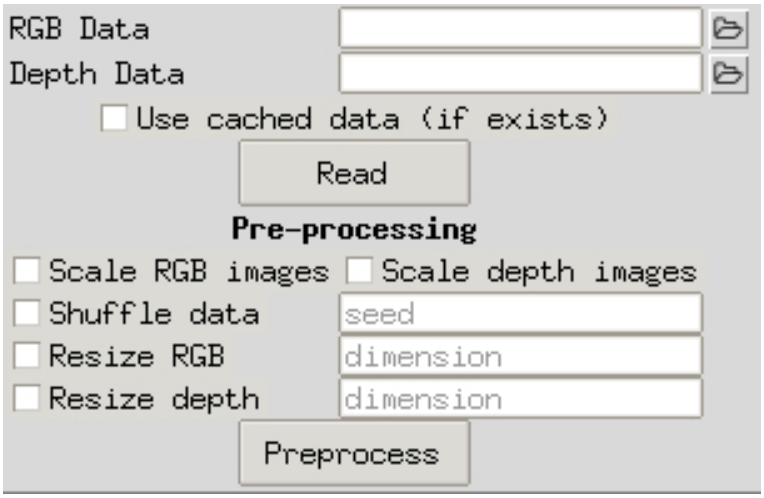

Figure 5.2: Data input frame

5.3.2 Network architecture frame

This frames allows user to select different layers to design a neural network architecture. User also has option to load a predefined network and modify it. Other parameters like learning rate, batch size etc. are also given in this frame only.

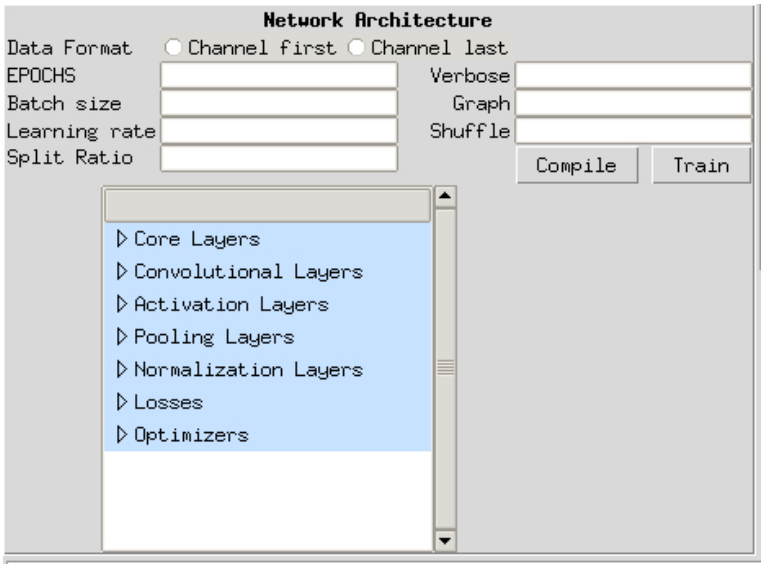

Figure 5.3: Network architecture frame

5.3.3 Visualization frame

In this frame loss can be visualized while training and once training is completed it provides tools to save and modify the plot. One can also see the predicted output in this frame

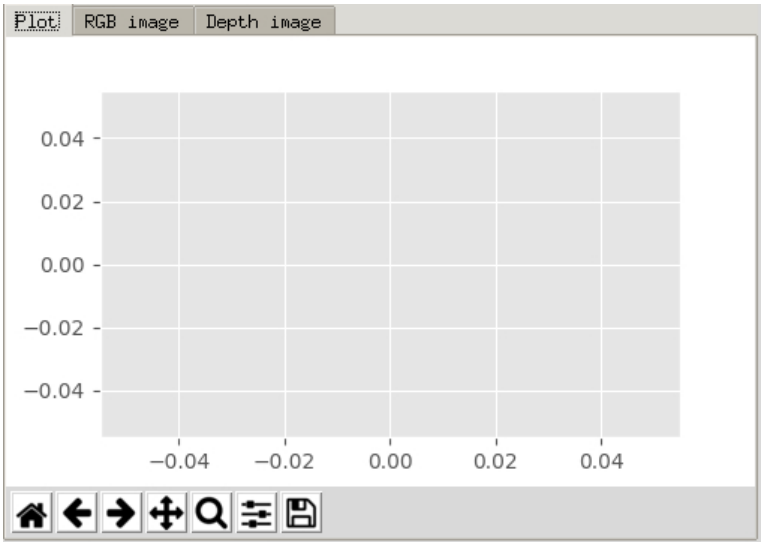

Figure 5.4: Visualization frame

CHAPTER 6

Conclusions

6.1 Depth estimation

In this project different convolutional neural network models are proposed to estimate depth. Even without using GPU, a decent estimate of depth is obtained which is good enough for the navigation of drone in corridor environment.

6.2 Corridor navigation

A very simple algorithm is proposed for navigation of drone in the corridor environment. Algorithm is capable of handling moving obstacles if ground station is powerful enough to provide high output frame rate. This algorithm improves upon the previously proposed navigation algorithms using deep learning in which motion of drone was limited to fixed number of directions (Tai *et al.*, 2016) by allowing the drone to autonomously chose direction instead of providing predefined set of directions.

6.3 Graphical User Interface

A GUI for training neural network models on RGB-D data is built. This GUI simplifies the task of designing and training neural networks to estimate depth. It provides tools to visualize error during training. One can also save predictions and use predefined model using this GUI.

6.4 Future work

Although convolutional neural networks have enjoyed lot of success in the recent past, they have a major drawback that is they are not able to encode relative spatial relation-

ship between features. To overcome this limitation Sabour *et al.* (2017) came up with a new architecture called capsule net. In future this architecture can be explored for the task of depth estimation.

Now a days ensemble methods are becoming increasingly popular. Ensemble methods are nothing but a mixture of weak learners which are combined to achieve better results. In the case of depth estimation different models can be trained and average of the depth from each model can be used as a final estimate.

In future, corridor navigation algorithm can be improved to detect walls and to intelligently navigate in case of turns or allow for landing in case of dead end. Also, the functionality of GUI can be extended by providing an interface to ROS so as to simulate the results of the model on a drone.

APPENDIX A

Code for training Neural Network model

```
"""
Author : Ashutosh Singh
Roll no: AE13B051

This is a sample code to design a neural  network and visualize the
    results

"""

from random import shuffle
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import gc
import time
import h5py
import pandas as pd
from tqdm import tqdm

from keras.models import Sequential
from keras.layers import Activation, Dropout, Flatten, Dense,
    BatchNormalization, Conv2D
from keras.utils.np_utils import to_categorical
from keras.callbacks import TensorBoard
from keras.layers.convolutional import ZeroPadding2D
from keras.optimizers import Adam
from keras.engine.topology import Input

IMG_SIZE  = 40
split_ratio = 0.80
EPOCHS = 50
BATCH_SIZE = 128
LR = 1.0e-3
```

```
split_ratio = 0.10

path_c1   = "less_1m"
path_c2 =  "more_1m"

# Reading data & Image Pre-processing

def img_preprocess(file_dir) :
    file_list = os.listdir(file_dir)
    data = []
    for img_lbl in tqdm(file_list) :
        img_lb = file_dir+"/"+img_lbl
        img = cv2.imread(img_lb , cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img , (IMG_SIZE , IMG_SIZE))
        img = np.transpose(img)
        data.append(img)
    return np.array(data)

##### Preparing, Merging and shuffling data

## using 1-hot encoding to encode the two classes

data_less_1m  = img_preprocess(path_c1)    # [1,0] for less than 1m
data_more_1m = img_preprocess(path_c2)      # [0,1] for more than 1m

label_less_1m = np.ones(data_less_1m.shape[0])
label_more_1m = np.zeros(data_more_1m.shape[0])

X = np.concatenate([data_less_1m , data_more_1m] , axis =  0)
y = np.concatenate([label_less_1m , label_more_1m] , axis =  0)

X = np.divide(X,255.0)

y = to_categorical(y)

y.shape

dummy = []
for i in tqdm(range(X.shape[0])) :
    dummy.append(X[i].flatten())
```

```
73 X = np.array(dummy)
74
75 c = np.c_[X.reshape(len(X), -1), y.reshape(len(y), -1)]
76
77 np.random.seed(1)
78 np.random.shuffle(c)
79
80 X2 = c[:, :X.size//len(X)].reshape(X.shape)
81 y2 = c[:, X.size//len(X):].reshape(y.shape)
82
83
84 X = X2.copy()
85 y = y2.copy()
86
87 print "Input : {} Output : {}".format(X.shape, y.shape)
88
89 # Deep Neural Network Classifier
90
91 ###### Defining the architectire of the neural network
92
93 model  = Sequential()
94
95 # First conv + activation + pooling layer
96
97 model.add(Dense(100 , input_dim = X.shape[1]))
98 model.add(Activation('sigmoid'))
99
100 model.add(Dense(100))
101 model.add(Activation('sigmoid'))
102
103
104 model.add(Dense(2))
105 model.add(Activation('softmax'))
106
107 model.summary()
108
109 adam = Adam(lr = LR, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-08,
      decay = 0.0)
110 model.compile(loss='binary_crossentropy', optimizer = adam)
111 tensorboard = TensorBoard(log_dir="./logs_5" , write_graph = True,
```

```
    write_images = True)

history = model.fit(X, y, epochs = EPOCHS, batch_size = BATCH_SIZE,
    verbose = 1, validation_split = split_ratio
                    ,callbacks = [tensorboard], shuffle = True)

model.save_weights('weights_1.hdf5')

##### Predicting

train_pred = model.predict(X)

y_classes = train_pred.argmax(axis=-1)

y_classes

y_pred_train = y_classes[:4595]
y_pred_valid = y_classes[4595:]

y_true = y.argmax(axis = -1)
y_train = y_true[:4595]
y_valid = y_true[4595:]

train_accuracy  = sum(y_pred_train == y_train)/4595.0
test_accuracy  = sum(y_pred_valid == y_valid)/511.0

print("Training set accuracy of the model is {}%".format(
    train_accuracy*100) )
print("Testing set accuracy of the model is {}%".format(test_accuracy
    *100) )

plt.figure()
plt.plot(range(EPOCHS) ,history.history['loss'])
plt.plot(range(EPOCHS) ,history.history['val_loss'])
plt.xlabel('EPOCHS')
plt.ylabel('Loss (Binary_crossentropy)')
plt.legend(['Train' , 'Test'])
plt.show()

# Data Visualization
```

```
148
149 fig=plt.figure()
150
151 for num,data in enumerate(X[0:20]):
152     # less than 1 m: [1,0]
153     # more than 1 m: [0,1]
154
155     img_num = num
156     img_data = data
157
158     y = fig.add_subplot(5,4,num+1)
159     #orig = img_data
160     data = img_data.reshape(IMG_SIZE,IMG_SIZE)
161     model_out = y_train[num]
162
163     if model_out == 1: str_label='Less  1m'
164     else: str_label='Greater 1m'
165     plt.subplots_adjust(left= 0.125 , bottom=0.1, right=0.9, top=1.5,
    wspace=0.5, hspace= 0.2)
166     y.imshow(data,cmap='gray')
167     plt.title(str_label)
168     y.axes.get_xaxis().set_visible(False)
169     y.axes.get_yaxis().set_visible(False)
170 plt.show()
```

APPENDIX B

Code for extracting and registering kinect data

```
"""
  Extract and registers RGB-D data from kinect
  Author  : Ashutosh Singh
  Roll no : AE13B051
"""
import numpy as np
import cv2
import sys
from pylibfreenect2 import Freenect2, SyncMultiFrameListener
from pylibfreenect2 import FrameType, Registration, Frame
from pylibfreenect2 import createConsoleLogger, setGlobalLogger
from pylibfreenect2 import LoggerLevel

# Directory Management
save_data = True                                # set to False if don't
    want to save data
path_name = 'path'                  # folder for saving data
path_depth  = path_name + '_depth'
path_rgb = path_name + '_rgb'
path_reg = path_name + '_reg'
path_bigDepth = path_name + '_bigDepth'

try:
    from pylibfreenect2 import OpenCLPacketPipeline
    pipeline = OpenCLPacketPipeline()
except:
    try:
        from pylibfreenect2 import OpenGLPacketPipeline
        pipeline = OpenGLPacketPipeline()
    except:
        from pylibfreenect2 import CpuPacketPipeline
        pipeline = CpuPacketPipeline()
print("Packet pipeline:", type(pipeline).__name__)

```

```
34 # Create and set logger
35 logger = createConsoleLogger(LoggerLevel.Debug)
36 setGlobalLogger(logger)
37
38 fn = Freenect2()
39 num_devices = fn.enumerateDevices()
40 if num_devices == 0:
41     print("No device connected!")
42     sys.exit(1)
43
44 serial = fn.getDeviceSerialNumber(0)
45 device = fn.openDevice(serial, pipeline=pipeline)
46
47 listener = SyncMultiFrameListener(
48     FrameType.Color | FrameType.Ir | FrameType.Depth)
49
50 # Register listeners
51 device.setColorFrameListener(listener)
52 device.setIrAndDepthFrameListener(listener)
53
54 device.start()
55
56 # NOTE: must be called after device.start()
57 registration = Registration(device.getIrCameraParams(),
58                             device.getColorCameraParams())
59
60 undistorted = Frame(512, 424, 4)
61 registered = Frame(512, 424, 4)
62
63 # Optimal parameters for registration
64 # set True if you need
65
66 need_bigdepth = True
67 need_color_depth_map = False
68
69 i,j = 0,0
70 skip_frame = 15
71 while True:
72     frames = listener.waitForNewFrame()
73
```

```
74      color = frames["color"]
75      ir = frames["ir"]
76      depth = frames["depth"]
77
78      registration.apply(color, depth, undistorted, registered)
79
80      cv2.imshow("depth", depth.asarray() / 4500.)
81      cv2.imshow("color", cv2.resize(color.asarray(),(int(1920 / 3),
        int(1080 / 3))))
82      cv2.imshow("registered", registered.asarray(np.uint8))
83      if i%skip_frame == 0 and save_data :
84          np.save('/home/ashu/DDP/Kinect/RGB/'+ path_rgb +'/rgb_{}'.
        format(j),color.asarray())
85          np.save('/home/ashu/DDP/Kinect/Depth/'+ path_depth +'/depth_
        {}'.format(j),depth.asarray())
86          np.save('/home/ashu/DDP/Kinect/registered/'+ path_reg +'/reg_
        {}'.format(j) ,registered.asarray(np.uint8))
87          j += 1
88          print j
89
90      i += 1
91      listener.release(frames)
92      key = cv2.waitKey(delay=1)
93      if key == ord('q'):
94          break
95  device.stop()
96  device.close()
97
98  sys.exit(0)
```

APPENDIX C

Code for corridor navigation

Main

```
"""
# Copyright (c) 2011 Bastian Venthur
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
```

```
# THE SOFTWARE.
"""

from navigation_utils import *
from termcolor import colored
import pygame
import pygame.surfarray
import pygame.transform
import libardrone as ardrone
from scipy.misc import imresize, imrotate

print colored("Setting up drone..." , 'yellow')
pygame.init()
W, H = 512, 256
FILTER_SHAPE = (250,80)

red = (255,0,0)
green = (0,255,0)
blue = (0,0,255)
darkBlue = (0,0,128)
white = (255,255,255)
black = (0,0,0)
pink = (255,200,200)


screen = pygame.display.set_mode((W, H))
drone = ardrone.ARDrone(True)
drone.reset()
#speed = raw_input("Enter the speed (0,1) : ")
#drone.set_speed(speed)
clock = pygame.time.Clock()
print colored("Starting mission ..." , "green")

i = 0
running = True
show_depth = raw_input("Show depth ? : ")
save_raw_image = raw_input("Save raw rgb image ? : ")
save_processed_image = raw_input("Save processed rgb image ? : ")
save_depth = raw_input("Save depth ? : ")
while running:
```

```
60      for event in pygame.event.get():
61          if event.type == pygame.QUIT:
62              running = False
63          elif event.type == pygame.KEYUP:
64              drone.hover()
65          elif event.type == pygame.KEYDOWN:
66              if event.key == pygame.K_ESCAPE:
67                  drone.reset()
68                  running = False
69              # takeoff / land
70              elif event.key == pygame.K_RETURN:
71                  drone.takeoff()
72              elif event.key == pygame.K_SPACE:
73                  drone.land()
74              # emergency
75              elif event.key == pygame.K_BACKSPACE:
76                  drone.reset()
77              # forward / backward
78              elif event.key == pygame.K_w:
79                  drone.move_forward()
80              elif event.key == pygame.K_s:
81                  drone.move_backward()
82              # left / right
83              elif event.key == pygame.K_a:
84                  drone.move_left()
85              elif event.key == pygame.K_d:
86                  drone.move_right()
87              # up / down
88              elif event.key == pygame.K_UP:
89                  drone.move_up()
90              elif event.key == pygame.K_DOWN:
91                  drone.move_down()
92              # turn left / turn right
93              elif event.key in [pygame.K_LEFT, pygame.K_q]:
94                  drone.turn_left()
95              elif event.key in [pygame.K_RIGHT, pygame.K_e]:
96                  drone.turn_right()
97              # speed
98              elif event.key == pygame.K_1:
99                  drone.speed = 0.1
```

```
100             elif event.key == pygame.K_2:
101                 drone.speed = 0.2
102             elif event.key == pygame.K_3:
103                 drone.speed = 0.3
104             elif event.key == pygame.K_4:
105                 drone.speed = 0.4
106             elif event.key == pygame.K_5:
107                 drone.speed = 0.5
108             elif event.key == pygame.K_6:
109                 drone.speed = 0.6
110             elif event.key == pygame.K_7:
111                 drone.speed = 0.7
112             elif event.key == pygame.K_8:
113                 drone.speed = 0.8
114             elif event.key == pygame.K_9:
115                 drone.speed = 0.9
116             elif event.key == pygame.K_0:
117                 drone.speed = 1.0
118             elif event.key == pygame.K_r:
119                 drone.set_camera_view(True)
120             elif event.key == pygame.K_f:
121                 drone.set_camera_view(False)
122     try:
123         pixelarray = drone.get_image()
124         if save_raw_image :
125             cv2.imwrite("raw_image/img_{}.jpg".format(i) , pixelarray
    )
126         if (not pixelarray is None) and pixelarray.any():
127             print "Processing Video....."
128             depth = get_depth(pixelarray)
129             depth = np.subtract(1,depth)
130             if show_depth :
131                 d = np.multiply(np.divide(np.subtract(depth,np.amin(
    depth)) , np.amax(depth)-np.amin(depth)),255)
132                 cv2.imshow('Depth', d)
133                 cv2.waitKey(1)
134             if save_depth :
135                 cv2.imwrite("Depth/depth_{}.jpg".format(i) , d)
136             x_max , y_max = deepest_region_detect(depth ,
    FILTER_SHAPE)
```

```
137             y_drone , x_drone = depth.shape
138             y_drone , x_drone = y_drone/2 , x_drone/2
139             ## Control
140             if x_drone > x_max + FILTER_SHAPE[1] :
141                 print "moving left"
142                 direction = "Moving left"
143                 drone.move_left()
144                 #time.sleep(4)
145                 drone.hover()
146             elif  x_drone < x_max :
147                 print "moving right"
148                 direction = "Moving right"
149                 drone.move_right()
150                 #time.sleep(4)
151                 drone.hover()
152             else :
153                 print "moving forward"
154                 direction = "Moving forward"
155                 drone.move_forward()
156                 #time.sleep(10)
157                 drone.hover()
158             ## Live Video
159             img = imresize(pixelarray , (256,512))
160             surface = pygame.surfarray.make_surface(img)
161             rotsurface = pygame.transform.rotate(surface, 270)
162             rotsurface = pygame.transform.flip(rotsurface, True,
    False)
163             pygame.draw.rect(rotsurface , red , [x_max, y_max,
    FILTER_SHAPE[1] , FILTER_SHAPE[0]] , 2)
164             pygame.draw.circle(rotsurface , green , [x_drone, y_drone
    ] , 2)
165             screen.blit(rotsurface, (0, 0))
166
167         # battery status
168         hud_color = (255, 0, 0) if drone.navdata.get('drone_state',
    dict()).get('emergency_mask', 1) else (10, 10, 255)
169         bat = drone.navdata.get(0, dict()).get('battery', 0)
170         f = pygame.font.Font(None, 20)
171         f2 = pygame.font.Font(None, 20)
172         hud = f.render('Battery: %i%%' % bat, True, hud_color)
```

```
173             control = f2.render(direction, True, pink)
174             screen.blit(control, (10, 25))
175             screen.blit(hud, (10, 10))
176             if save_processed_image :
177                 pygame.image.save(screen, "Image/img_{}.jpg".format(i))
178             i += 1
179         except KeyboardInterrupt:
180             break
181         except:
182             pass
183 
184         pygame.display.flip()
185         clock.tick(50)
186         pygame.display.set_caption("FPS: %.2f" % clock.get_fps())
187 
188 print("Shutting down...")
189 drone.halt()
190 print("Ok.")
```

Utility functions

```
1 """
2   Utility functions like predicting depth
3 """
4 from __future__ import absolute_import, division, print_function
5 import os
6 os.environ['TF_CPP_MIN_LOG_LEVEL']='0'
7 
8 import cv2
9 import numpy as np
10 import gc, time
11 from tqdm import tqdm
12 import time
13 from math import sqrt
14 import re
15 import tensorflow as tf
16 import scipy.misc
17 import matplotlib.pyplot as plt
18 from monodepth_model import *
19 from monodepth_dataloader import *
```

```
20
21
22 FILTER_SHAPE = (718,80)
23 green = (0,255,0)
24 blue = (0,0,255)
25 red = (255,0,0)
26 depth_shape = (256, 512)
27
28
29 def post_process_disparity(disp):
30     _, h, w = disp.shape
31     l_disp = disp[0,:,:]
32     r_disp = np.fliplr(disp[1,:,:])
33     m_disp = 0.5 * (l_disp + r_disp)
34     l, _ = np.meshgrid(np.linspace(0, 1, w), np.linspace(0, 1, h))
35     l_mask = 1.0 - np.clip(20 * (l - 0.05), 0, 1)
36     r_mask = np.fliplr(l_mask)
37     return r_mask * l_disp + l_mask * r_disp + (1.0 - l_mask - r_mask
    ) * m_disp
38
39 def get_depth(img , encoder = 'vgg'):
40     """Test function."""
41     input_height = 256
42     input_width = 512
43     checkpoint_path = 'models/model_cityscapes'
44     params = monodepth_parameters(
45         encoder = encoder,
46         height= input_height,
47         width= input_width,
48         batch_size=2,
49         num_threads=4,
50         num_epochs=1,
51         do_stereo=False,
52         wrap_mode="border",
53         use_deconv=False,
54         alpha_image_loss=0,
55         disp_gradient_loss_weight=0,
56         lr_loss_weight=0,
57         full_summary=False)
58     tf.reset_default_graph()
```

```
59      left  = tf.placeholder(tf.float32, [2, input_height, input_width,
        3])
60      model = MonodepthModel(params, "test", left, True)
61
62      input_image = img
63      original_height, original_width, num_channels = input_image.shape
64      input_image = scipy.misc.imresize(input_image, [input_height,
       input_width], interp='lanczos')
65      input_image = input_image.astype(np.float32) / 255
66      input_images = np.stack((input_image, np.fliplr(input_image)), 0)
67
68      # SESSION
69      config = tf.ConfigProto(allow_soft_placement=True)
70      with tf.Session(config=config , graph = None) as sess:
71
72          # SAVER
73          train_saver = tf.train.Saver()
74
75          # INIT
76          sess.run(tf.global_variables_initializer())
77          sess.run(tf.local_variables_initializer())
78          coordinator = tf.train.Coordinator()
79          threads = tf.train.start_queue_runners(sess=sess, coord=
       coordinator)
80
81          # RESTORE
82          restore_path = checkpoint_path.split(".")[0]
83          train_saver.restore(sess, restore_path)
84          disp = sess.run(model.disp_left_est[0], feed_dict={left:
       input_images})
85
86      disp_pp = post_process_disparity(disp.squeeze()).astype(np.
       float32)
87
88      return disp_pp
89
90
91  def create_filter(filter_shape) :
92      """ Creates weighted filter"""
93      m,n = filter_shape
```

```
94      k = n//2
95      f = np.array(range(1,k+1)).reshape(1,k)
96      f = np.concatenate([f]*m , axis = 0)
97      if n%2 == 0 :
98          f = np.concatenate([f, np.flip(f , axis = 1)] , axis = 1)
99          return f
100     else :
101         f = np.concatenate([f, (k+1)*np.ones([m,1]),np.flip(f , axis
        = 1)] , axis = 1)
102         return f
103
104
105 def deepest_region_detect(img , filter_shape , filter_type = '
        avg_filter') :
106     """ Average filter is used to find deepest region"""
107     if filter_type == 'avg_filter' :
108         avg_filter = np.ones(filter_shape)
109     elif filter_type == 'wt_avg_filter' :
110         avg_filter = create_filter(filter_shape)
111     else : raise ValueError('filter_type got an unexpected value')
112
113     x_max , y_max = 0,0
114     curr_max = 0
115     for i in range(img.shape[0] - filter_shape[0]) :
116         for j in range(img.shape[1] - filter_shape[1]) :
117             img_seg  = img[i:i+filter_shape[0] ,j:j + filter_shape
        [1]]
118             avg  = np.sum(np.multiply(avg_filter ,img_seg ))
119             if avg > curr_max :
120                 curr_max = avg
121                 x_max, y_max =  j,i
122     return (x_max, y_max)
123
124 def create_box(img , (x_max,y_max) ,filter_shape , color = (0,255,0))
        :
125     """Creates a box given the top right corner and box shape along
        with color"""
126     return cv2.rectangle(img,(x_max, y_max),(x_max + FILTER_SHAPE[1],
        y_max + FILTER_SHAPE[0]),color ,3)
```

APPENDIX D

Code for creating GUI

```
"""
    This is a code for generating Graphical User Interface
    Author  : Ashutosh Singh
    Roll no : AE13B051
"""

from Tkinter import *
import ScrolledText
import sys
import os
import numpy as np
import cv2
import scipy.io as sio
import warnings
import ttk
from PIL import ImageTk, Image
from keras.models import Sequential, model_from_json
from keras.layers import Conv2D, Dense, Dropout, Reshape, Flatten,
    Conv2DTranspose, UpSampling2D, Activation
from keras.layers import MaxPooling2D, AveragePooling2D,
    BatchNormalization
from keras.optimizers import SGD, RMSprop, Adam
from keras.callbacks import Callback
from tkFileDialog import askopenfilename, asksaveasfilename,
    askdirectory
import tkMessageBox
from messages import *
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
    NavigationToolbar2TkAgg
from matplotlib.figure import Figure
from matplotlib import style

```

```
31
32 warnings.simplefilter("ignore")
33 style.use('ggplot')
34
35 root = Tk()
36 f = Figure(figsize=(5, 3), dpi=100)
37 c_plot = f.add_subplot(111)
38 c_plot.plot()
39
40
41 class PlotLosses(Callback):
42
43     def __init__(self, graph_page):
44
45         super(PlotLosses, self).__init__()
46         self.graph_page = graph_page
47         self.i, self.x, self.losses, self.val_losses, self.logs = [
    None]*5
48
49     def on_train_begin(self, logs={}):
50         self.i = 0
51         self.x = []
52         self.losses = []
53         self.val_losses = []
54         self.logs = []
55
56     def on_epoch_end(self, epoch, logs={}):
57         self.i += 1
58         self.logs.append(logs)
59         self.x.append(self.i)
60         self.losses.append(logs.get('loss'))
61         self.val_losses.append(logs.get('val_loss'))
62
63         Application.clear_frame(self.graph_page)
64         Application.init_dynamic_plot(self.graph_page)
65
66         c_plot.plot(self.x, self.losses, color='red', label='Training
    ')
67         c_plot.plot(self.x, self.val_losses, color='blue', label='
    Validation')
```

```
68
69
70 class RedirectText:
71     """Redirects the stdoutput and stderr to tkinter textbox"""
72
73     def __init__(self, text_ctrl):
74         self.output = text_ctrl
75
76     def write(self, string):
77         self.output.configure(state=NORMAL)
78         self.output.insert(END, string)
79         self.output.update_idletasks()
80         self.output.see(END)
81         self.output.configure(state=DISABLED)
82
83     @staticmethod
84     def flush():
85         pass
86
87     @staticmethod
88     def isatty():
89         return False
90
91
92 class Application:
93     n_layers = 0
94
95     def __init__(self, master):
96
97         # Important milestone markers
98         self.preprocess_flag, self.read_flag, self.network_flag = [
    False]*3
99         self.compile_flag, self.train_flag = [None]*2
100        # Dataset Properties
101        self.n_data, self.depth_shape, self.rgb_shape = [None]*3
102
103        # Dividing screen into frames
104        frame_l = Frame(master, bd=2)
105        self.frame_r = Frame(master, bd=2)
106        self.frame_l2 = Frame(master, bd=2, relief=GROOVE)
```

```
107         # self.frame_r2 = Frame(master, bd=2)
108         frame_b = Frame(master, bd=2)
109         frame_t = Frame(master, bg='gray45', bd=1, relief=RAISED)
110 
111         # Placing frames in the master window
112         frame_t.grid(row=0, columnspan=2, sticky=W+E+N)
113         frame_l.grid(row=1, column=0)
114         self.frame_r.grid(row=1, column=1, rowspan=2, sticky=N)
115         self.frame_l2.grid(row=2, column=0)
116         # self.frame_r2.grid(row=2, column=1, sticky="nsew")
117         frame_b.grid(row=4, columnspan=2, sticky=W+E)
118 
119         # Creating Log Window and redirecting stdout to log window
120         self.text_log = ScrolledText.ScrolledText(frame_b, state=
    DISABLED,  width=50, height=8,
121                                                   font=("Comic Sans
    MS", 10, 'bold'))
122         self.text_log.pack(fill=BOTH, expand=True)
123         redir = RedirectText(self.text_log)
124         sys.stdout = redir
125         sys.stderr = redir
126 
127         # Menu Bar
128         menu = Menu(master, bg="gray35", tearoff=0, relief=RAISED)
129         master.config(menu=menu)
130 
131         sub_menu = Menu(menu, tearoff=0, relief=RAISED)
132         menu.add_cascade(label="File", menu=sub_menu)
133         sub_menu.add_command(label="Reset", command=self.reset)
134         sub_menu.add_command(label="Clear log", command=self.
    clear_log)
135         sub_menu.add_command(label="Clear cache", command=self.
    clear_cache)
136         sub_menu.add_separator()
137         sub_menu.add_command(label="Save model", command=self.
    save_model)
138         sub_menu.add_command(label="Save weights", command=self.
    save_weights)
139         sub_menu.add_command(label="Save model history", command=self
    .save_model_history)
```

```
140         sub_menu.add_separator()
141         sub_menu.add_command(label="Load model", command=self.
    load_model)
142         sub_menu.add_command(label="Load weights", command=self.
    load_weights)
143         sub_menu.add_command(label="Load model history", command=self
    .load_model_history)
144         sub_menu.add_separator()
145         sub_menu.add_command(label="Exit", command=self.close_app)
146 
147         # Edit Menu
148         edit_menu = Menu(menu, tearoff=0, relief=RAISED)
149         menu.add_cascade(label='Edit', menu=edit_menu)
150         edit_menu.add_command(label="Copy       ", command=self.
    clear_log)
151         edit_menu.add_command(label="Cut", command=master.quit)
152         edit_menu.add_command(label="Paste", command=master.quit)
153 
154         # View Menu
155         view_menu = Menu(menu, tearoff=0, relief=RAISED)
156         menu.add_cascade(label='View', menu=view_menu)
157         view_menu.add_command(label="Data Summary", command=self.
    data_summary)
158         view_menu.add_command(label="Depth Dimension", command=self.
    get_depth_shape)
159         view_menu.add_command(label="RGB Dimension", command=self.
    get_rgb_shape)
160         view_menu.add_command(label="Dataset size", command=self.
    get_dataset_size)
161         sub_menu.add_separator()
162         view_menu.add_command(label='Model Summary', command=self.
    network_summary)
163 
164         # Network Menu
165         network_menu = Menu(menu, tearoff=0, relief=RAISED)
166         menu.add_cascade(label='Network', menu=network_menu)
167         network_menu.add_command(label="Reset Network", command=self.
    reset_model)
168         network_menu.add_command(label="Delete last layer", command=
    self.remove_last_layer)
```

```
169
170         # Predict Menu
171         predict_menu = Menu(menu, tearoff=0, relief=RAISED)
172         menu.add_cascade(label='Predict', menu=predict_menu)
173         predict_menu.add_command(label="Single Image", command=self.
    read_predict)
174         predict_menu.add_command(label="Directory", command=self.
    predict_folder)
175
176         # Toolbar
177         # Reset Button
178         reset_image = ImageTk.PhotoImage(Image.open("images/
    reset_icon.png"))
179         reset_button = Button(frame_t, image=reset_image, command=
    self.reset, anchor=W)
180         reset_button.image = reset_image
181         reset_button.grid(row=0, column=1, padx=2, pady=2)
182
183         # Clear log button
184         clear_image = ImageTk.PhotoImage(Image.open("images/
    clear_log_icon.png"))
185         clear_button = Button(frame_t, image=clear_image, command=
    self.clear_log, anchor=W)
186         clear_button.image = clear_image
187         clear_button.grid(row=0, column=2, padx=2, pady=2)
188
189         # Clear cache button
190         clear_cache_image = ImageTk.PhotoImage(Image.open("images/
    clear_cache_icon.png"))
191         clear_cache_button = Button(frame_t, image=clear_cache_image,
     command=self.clear_cache, anchor=W)
192         clear_cache_button.image = clear_cache_image
193         clear_cache_button.grid(row=0, column=3, padx=2, pady=2)
194
195         # Status Bar
196         self.status_msg = StringVar()
197         self.status = Label(master, textvariable=self.status_msg, bd
    =2, relief=SUNKEN, anchor=W, bg="gray")
198         self.status.grid(row=5, columnspan=2, sticky=W+E)
199
```

```
200         # Left frame Layout
201
202         # Label and ttk.Entry of RGB and Depth data
203         self.rgb_lbl = Label(frame_l, text='RGB Data')
204         self.rgb_entry = ttk.Entry(frame_l)
205         self.depth_lbl = Label(frame_l, text='Depth Data')
206         self.depth_entry = ttk.Entry(frame_l)
207
208         # Selecting path from directory
209         browse_img = ImageTk.PhotoImage(Image.open('images/
    browse_icon12.png'))
210         self.explore_rgb_b = Button(frame_l, image=browse_img,
    command=self.browse_folder_rgb)
211         self.explore_rgb_b.image = browse_img
212         self.explore_depth_b = Button(frame_l, image=browse_img,
    command=self.browse_folder_depth)
213         self.explore_depth_b.image = browse_img
214
215         # Cache Data usage checkbutton
216         self.state_cache_cbox = IntVar()
217         self.cache_cbox = ttk.Checkbutton(frame_l, text="Use cached
    data (if exists)",
218                                           state=NORMAL, variable=self
    .state_cache_cbox)
219
220         # Layout of Label and ttk.Entry of RGB and Depth data
221         self.rgb_lbl.grid(row=0, sticky=W)
222         self.depth_lbl.grid(row=1, sticky=W)
223         self.rgb_entry.grid(row=0, column=1)
224         self.depth_entry.grid(row=1, column=1)
225         self.cache_cbox.grid(row=2, columnspan=2)
226         self.explore_rgb_b.grid(row=0, column=2)
227         self.explore_depth_b.grid(row=1, column=2)
228
229         # Read button logic and connection
230         self.read_button = ttk.Button(frame_l, text="Read")
231         self.read_button.grid(row=3, columnspan=2)
232         self.read_button.bind("<Button-1>", self.read_b_signal)
233
234         # Data processing labels and widgets
```

```
235         pp_lbl = Label(frame_1, text="Pre-processing", font="Verdana
    10 bold", anchor=CENTER)
236         pp_lbl.grid(row=4, columnspan=2)
237 
238         # Scaling checkboxes
239         self.scale_rgb_cb_var = IntVar()
240         self.scale_depth_cb_var = IntVar()
241         scale_rgb_cb = ttk.Checkbutton(frame_1, text="Scale RGB
    images", variable=self.scale_rgb_cb_var)
242         scale_depth_cb = ttk.Checkbutton(frame_1, text="Scale depth
    images", variable=self.scale_depth_cb_var)
243         scale_rgb_cb.grid(row=6, column=0, sticky=W)
244         scale_depth_cb.grid(row=6, column=1, sticky=W)
245 
246         # Shuffling Checkbox and entry
247         self.state_shuffle = IntVar()
248         shuffle_cb = ttk.Checkbutton(frame_1, text="Shuffle data",
    variable=self.state_shuffle, command=self.seed_check)
249         shuffle_cb.grid(row=7, column=0, sticky=W)
250 
251         self.seed_entry = ttk.Entry(frame_1)
252         self.seed_entry.grid(row=7, column=1, sticky=W)
253         self.seed_entry.insert(END, 'seed')
254         self.seed_entry.configure(state=DISABLED)
255 
256         # Resizing RGB checkbox and entry
257         self.state_resize_rgb = IntVar()
258         resize_rgb_cb = ttk.Checkbutton(frame_1, text="Resize RGB",
    variable=self.state_resize_rgb,
259                                         command=self.rgb_check)
260         resize_rgb_cb.grid(row=8, column=0, sticky=W)
261 
262         self.resize_rgb_entry = ttk.Entry(frame_1)
263         self.resize_rgb_entry.grid(row=8, column=1, sticky=W)
264         self.resize_rgb_entry.insert(END, 'dimension')
265         self.resize_rgb_entry.configure(state=DISABLED)
266 
267         # Resizing depth checkbox and entry
268         self.state_resize_depth = IntVar()
269         resize_depth_cb = ttk.Checkbutton(frame_1, text="Resize depth
```

```
", variable=self.state_resize_depth,
                                                command=self.depth_check)
        resize_depth_cb.grid(row=9, column=0, sticky=W)

        self.resize_depth_entry = ttk.Entry(frame_1)
        self.resize_depth_entry.grid(row=9, column=1, sticky=W)
        self.resize_depth_entry.insert(END, 'dimension')
        self.resize_depth_entry.configure(state=DISABLED)

        # Pre-processing button
        self.pp_button = ttk.Button(frame_1, text="Preprocess")
        self.pp_button.grid(row=10, columnspan=2)
        self.pp_button.bind("<Button-1>", self.pp_b_signal)

        # Neural Network architecture
        top_frame = Frame(self.frame_12)
        left_frame = Frame(self.frame_12)
        self.right_frame = Frame(self.frame_12, relief=GROOVE)

        top_frame.grid(row=0, column=0, columnspan=2)
        left_frame.grid(row=1, column=0)
        self.right_frame.grid(row=1, column=1)

        # Initializing the network
        network_lbl = Label(top_frame, text="Network Architecture",
font="Verdana 10 bold",
                            anchor=CENTER)
        network_lbl.grid(row=0, columnspan=4)

        data_format_lbl = Label(top_frame, text='Data Format')
        data_format_lbl.grid(row=1, column=0, sticky=W)

        self.chnl_var = IntVar()
        chnl_first_rb = ttk.Radiobutton(top_frame, text="Channel
first", variable=self.chnl_var, value=1)
        chnl_last_rb = ttk.Radiobutton(top_frame, text="Channel last"
, variable=self.chnl_var, value=2)
        chnl_first_rb.grid(row=1, column=1)
        chnl_last_rb.grid(row=1, column=2)

```

```
306         epoch_lbl = Label(top_frame, text='EPOCHS')
307         batch_lbl = Label(top_frame, text='Batch size')
308         lr_lbl = Label(top_frame, text='Learning rate')
309         ratio_lbl = Label(top_frame, text='Split Ratio')
310 
311         epoch_lbl.grid(row=2, column=0, sticky=W)
312         batch_lbl.grid(row=3, column=0, sticky=W)
313         lr_lbl.grid(row=4, column=0, sticky=W)
314         ratio_lbl.grid(row=5, column=0, sticky=N+W)
315 
316         self.epoch_entry = ttk.Entry(top_frame)
317         self.batch_entry = ttk.Entry(top_frame)
318         self.lr_entry = ttk.Entry(top_frame)
319         self.ratio_entry = ttk.Entry(top_frame)
320 
321         self.epoch_entry.grid(row=2, column=1, columnspan=2, sticky=W
    )
322         self.batch_entry.grid(row=3, column=1, columnspan=2, sticky=W
    )
323         self.lr_entry.grid(row=4, column=1, columnspan=2, sticky=W)
324         self.ratio_entry.grid(row=5, column=1, columnspan=2, sticky=N
    +W)
325 
326         verbose_lbl = Label(top_frame, text='Verbose')
327         graph_lbl = Label(top_frame, text='Graph')
328         shuffle_lbl = Label(top_frame, text='Shuffle')
329 
330         verbose_lbl.grid(row=2, column=2, sticky=E)
331         graph_lbl.grid(row=3, column=2, sticky=E)
332         shuffle_lbl.grid(row=4, column=2, sticky=E)
333 
334         self.e_verbose = ttk.Entry(top_frame)
335         self.e_graph = ttk.Entry(top_frame)
336         self.e_shuffle = ttk.Entry(top_frame)
337 
338         self.e_verbose.grid(row=2, column=3)
339         self.e_graph.grid(row=3, column=3)
340         self.e_shuffle.grid(row=4, column=3)
341 
342         # Compiling network button
```

```
343         self.compile_b = Button(top_frame, text='Compile', command=
    self.compile_model)
344         self.compile_b.grid(row=5, column=3, sticky=W)
345 
346         self.train_b = Button(top_frame, text='Train', command=self.
    train_model)
347         self.train_b.grid(row=5, column=3, sticky=E)
348 
349         # Creating self.tree of all avialable layers and displaying
    layout
350         self.tree = ttk.Treeview(left_frame, selectmode='browse')
351         self.tree.pack(side="left", fill="both", expand=True)
352 
353         self.tree.insert('', 'end', 'core', text="Core Layers")
354         self.tree.insert('', 'end', 'conv', text="Convolutional
    Layers")
355         self.tree.insert('', 'end', 'act', text="Activation Layers")
356         self.tree.insert('', 'end', 'pool', text="Pooling Layers")
357         self.tree.insert('', 'end', 'norm', text="Normalization
    Layers")
358         self.tree.insert('', 'end', 'loss', text="Losses")
359         self.tree.insert('', 'end', 'opt', text="Optimizers")
360 
361         # Core layer sub-items
362         self.tree.insert('core', 'end', 'dense', text="Fully
    connected")
363         self.tree.insert('core', 'end', 'drop', text="Dropout")
364         self.tree.insert('core', 'end', 'reshape', text="Reshape")
365         self.tree.insert('core', 'end', 'flatten', text='Flatten')
366 
367         # Convolutional layer sub-items
368         self.tree.insert('conv', 'end', 'conv2d', text="Conv2D")
369         self.tree.insert('conv', 'end', 'conv2dt', text="Conv2D
    Transpose")
370         self.tree.insert('conv', 'end', 'padding', text='
    ZeroPadding2D')
371         self.tree.insert('conv', 'end', 'upsample', text='
    UpSampling2D')
372 
373         # Pooling layer sub-items
```

```
374         self.tree.insert('pool', 'end', 'maxpool', text="Max Pooling
    2D")
375         self.tree.insert('pool', 'end', 'avgpool', text="Average
    Pooling 2D")
376 
377         # Normalization layer sub-items
378         self.tree.insert('norm', 'end', 'bnorm', text="Batch
    Normalization")
379 
380         # Activation layer sub-items
381         self.tree.insert('act', 'end', 'softmax', text="Softmax")
382         self.tree.insert('act', 'end', 'sigmoid', text="Sigmoid")
383         self.tree.insert('act', 'end', 'tanh', text="Tanh")
384         self.tree.insert('act', 'end', 'relu', text="Relu")
385 
386         # Losses sub-items
387         self.tree.insert('loss', 'end', 'mse', text="Mean Squared
    Error")
388         self.tree.insert('loss', 'end', 'mae', text="Mean Absolute
    Error")
389         self.tree.insert('loss', 'end', 'mse_log', text="Logarithmic
    MSE")
390 
391         # Optimizers sub-items
392         self.tree.insert('opt', 'end', 'sgd', text="SGD")
393         self.tree.insert('opt', 'end', 'rmsprop', text="RMSprop")
394         self.tree.insert('opt', 'end', 'adam', text="Adam")
395 
396         # Setting tags to each heading of self.tree
397         self.tree_headings = ['core', 'conv', 'act', 'pool', 'norm',
    'loss', 'opt']
398 
399         for item_ in self.tree_headings:
400             self.tree.item(item_, tags='heading')
401 
402         self.tree.tag_configure('heading', background='SlateGray1')
403 
404         # Adding scrollbar to self.tree
405         self.tree_scroll = ttk.Scrollbar(left_frame, orient="vertical
    ", command=self.tree.yview)
```

```
406         self.tree_scroll.pack(side="right", fill="y", expand=False)
407
408         # Adding control logic to network architecture tree
409         self.tree.bind("<Double-1>", self.tree_control)
410
411         # Initializing a lot of variables
412         self.e_dense_units, self.add_b = [None]*2
413
414         self.e_conv_filters, self.e_conv_strides, self.
    e_conv_kernel_size = [None]*3
415         self.e_conv_padding = "valid"
416
417         self.e_drop_rate, self.e_reshape_dim, self.e_padding, self.
    e_upsample = [None]*4
418
419         self.data_format = "channels_last"
420         self.model, self.epochs, self.batch_size, self.lr, self.
    split_ratio = [None]*5
421         self.loss, self.optimizer = [None]*2
422         self.e_decay, self.e_momentum, self.e_nesterov = [None]*3
423         self.e_rmsprop_decay, self.e_rho, self.e_epsilon = [None]*3
424         self.e_beta1, self.e_beta2, self.e_adam_epsilon = [None]*3
425         self.e_adam_decay = None
426         self.history = None
427
428         # Visualization notebook
429         self.nb = ttk.Notebook(self.frame_r)
430         self.nb.grid(sticky=N)
431
432         self.graph_page = Frame(self.nb)
433         self.images_page = Frame(self.nb)
434         self.depth_page = Frame(self.nb)
435
436         self.nb.add(self.graph_page, text='Plot')
437         self.nb.add(self.images_page, text='RGB image')
438         self.nb.add(self.depth_page, text='Depth image')
439
440         # Graph frame callback
441         self.init_dynamic_plot(self.graph_page)
442
```

```
443         # Prediction result display
444 
445         # Master window layout
446         master_icon = ImageTk.PhotoImage(Image.open('images/
    application_icon.png'))
447         master.tk.call('wm', 'iconphoto', master._w, master_icon)
448         master.grid_rowconfigure(0, weight=1)
449         master.grid_columnconfigure(0, weight=1)
450         master.resizable(True, True)
451         s = ttk.Style()
452         s.theme_use('clam')
453         master.protocol('WM_DELETE_WINDOW', self.close_app)
454         master.title("Corridor Navigator")
455 
456     @staticmethod
457     def str_to_bool(s):
458         if s == 'True':
459             return True
460         elif s == 'False':
461             return False
462         else:
463             raise ValueError
464 
465     @staticmethod
466     def clear_frame(frame):
467         for child in frame.winfo_children():
468             child.destroy()
469 
470     @staticmethod
471     def str_to_none(s):
472         if s == 'None':
473             return None
474         else:
475             return float(s)
476 
477     def progress_bar(self, iteration, total, prefix='', suffix='',
    decimals=1, length=100, fill='='):
478         """
479         Call in a loop to create terminal progress bar
480         @params:
```

```
481             iteration   - Required  : current iteration (Int)
482             total       - Required  : total iterations (Int)
483             prefix      - Optional  : prefix string (Str)
484             suffix      - Optional  : suffix string (Str)
485             decimals    - Optional  : positive number of decimals in
    percent complete (Int)
486             length      - Optional  : character length of bar (Int)
487             fill        - Optional  : bar fill character (Str)
488         """
489         percent = ("{0:." + str(decimals) + "f}").format(100 * (
    iteration / float(total)))
490         filled_length = int(length * iteration // total)
491         bar = fill * filled_length + '-' * (length - filled_length)
492         self.clear_log()
493         print '%s |%s| %s%% %s ' % (prefix, bar, percent, suffix)
494         # Print New Line on Complete
495         if iteration == total:
496             print ''
497 
498     @staticmethod
499     def shuffle_data(rgb_data, depth_data, seed=0):
500         np.random.seed(seed)
501         np.random.shuffle(rgb_data)
502         np.random.seed(seed)
503         np.random.shuffle(depth_data)
504 
505     @staticmethod
506     def close_app():
507         if tkMessageBox.askokcancel("Close", "Are you sure...?"):
508             sys.exit()
509 
510     def cache_data(self):
511         """Stores data in "Data" folder"""
512         rgb_path = self.rgb_entry.get().strip(' ')
513         depth_path = self.depth_entry.get().strip(' ')
514 
515         try:
516             rgb_data = self.read_data(rgb_path)
517             depth_data = self.read_data(depth_path, depth=True)
518             self.read_flag = True
```

```
519         except:
520             return
521 
522         self.n_data = rgb_data.shape[0]
523         self.rgb_shape = rgb_data.shape[1:]
524         self.depth_shape = depth_data.shape[1:]
525         sio.savemat('temp/data_description.mat', {'n_data': self.
    n_data, 'rgb_shape': self.rgb_shape,
526                                                   'depth_shape': self
    .depth_shape})
527         np.save("Data/rgb_data.npy", rgb_data)
528         np.save("Data/depth_data.npy", depth_data)
529 
530     def read_b_signal(self, event):
531         """Function gives signal to Read button to read RGB and Depth
     Data"""
532 
533         self.status_msg.set("Reading data ...")
534         if self.state_cache_cbox.get() == 0:
535             self.cache_data()
536         else:
537             if len(os.listdir('Data')) < 2:
538                 self.cache_data()
539             else:
540                 data_desc = sio.loadmat('temp/data_description.mat')
541                 self.n_data = data_desc['n_data'][0][0]
542                 self.rgb_shape = data_desc['rgb_shape'][0]
543                 self.depth_shape = data_desc['depth_shape'][0]
544                 self.status_msg.set("Using Cached data ...")
545                 self.read_flag = True
546                 return
547         self.status_msg.set("Finished reading data ...")
548 
549     def pp_b_signal(self, event):
550         """Gives signal to preprocessing button"""
551         if not self.read_flag:
552             print "No data found"
553             return
554         self.status_msg.set("Preprocessing ...")
555         pp_rgb = np.load("Data/rgb_data.npy")
```

```
556         pp_depth = np.load("Data/depth_data.npy")
557         try:
558             if self.scale_depth_cb_var.get():
559                 print "Scaling depth images ... \n"
560                 pp_depth = self.scale_depth_data(pp_depth)
561 
562             if self.scale_rgb_cb_var.get():
563                 print "Scaling rgb images ... \n"
564                 pp_rgb = self.scale_rgb_data(pp_rgb)
565 
566             if self.state_shuffle.get():
567                 print "Shuffling ...\n"
568                 self.shuffle_data(pp_rgb, pp_depth, int(self.
    seed_entry.get()))
569 
570             if self.state_resize_rgb.get():
571                 print "Resizing RGB image ...\n"
572                 dim = self.resize_rgb_entry.get()
573                 dim = tuple(map(int, dim.strip(' ').split(',')))
574                 pp_rgb = self.resize_data(pp_rgb, dim)
575                 self.rgb_shape = pp_rgb.shape[1:]
576 
577             if self.state_resize_depth.get():
578                 print "Resizing depth ...\n"
579                 dim = self.resize_depth_entry.get()
580                 dim = tuple(map(int, dim.strip(' ').split(',')))
581                 pp_depth = self.resize_data(pp_depth, dim)
582                 self.depth_shape = pp_depth.shape[1:]
583 
584             if self.scale_rgb_cb_var.get() == 0 and self.
    scale_depth_cb_var.get() == 0 and self.state_shuffle == 0:
585                 print "Please select at least one checkbox"
586                 self.status_msg.set("")
587                 return
588             np.save("ppData/pp_depth_data", pp_depth)
589             np.save("ppData/pp_rgb_data", pp_rgb)
590             self.preprocess_flag = True
591         except ValueError:
592             print "Something went wrong"
593             self.status_msg.set("")
```

```
594             return
595         self.status_msg.set("Preprocessing complete ...")
596 
597     def seed_check(self):
598         if self.state_shuffle.get():
599             self.seed_entry.configure(state='normal')
600             self.seed_entry.delete(0, END)
601         else:
602             self.seed_entry.insert(0, "seed")
603             self.seed_entry.configure(state='disabled')
604 
605     def rgb_check(self):
606         if self.state_resize_rgb.get():
607             self.resize_rgb_entry.configure(state='normal')
608             self.resize_rgb_entry.delete(0, END)
609         else:
610             self.resize_rgb_entry.insert(0, "dimension")
611             self.resize_rgb_entry.configure(state='disabled')
612 
613     def depth_check(self):
614         if self.state_resize_depth.get():
615             self.resize_depth_entry.configure(state='normal')
616             self.resize_depth_entry.delete(0, END)
617         else:
618             self.resize_depth_entry.insert(0, "dimension")
619             self.resize_depth_entry.configure(state='disabled')
620 
621     def browse_folder_rgb(self):
622         dirname = askdirectory(parent=root, initialdir=os.getcwd(),
    title='Please select a directory')
623         try:
624             self.rgb_entry.delete('0', END)
625             self.rgb_entry.insert(END, dirname)
626         except:
627             return
628 
629     def browse_folder_depth(self):
630         dirname = askdirectory(parent=root, initialdir=os.getcwd(),
    title='Please select a directory')
631         try:
```

```
632             self.depth_entry.delete('0', END)
633             self.depth_entry.insert(END, dirname)
634         except:
635             return
636 
637     def reset(self):
638         """Takes the window to default state"""
639 
640         self.rgb_entry.delete(0, END)
641         self.depth_entry.delete(0, END)
642         self.status_msg.set("")
643         self.state_cache_cbox.set(0)
644         self.scale_rgb_cb_var.set(0)
645         self.scale_depth_cb_var.set(0)
646 
647         self.state_shuffle.set(0)
648         self.state_resize_rgb.set(0)
649         self.state_resize_depth.set(0)
650 
651         self.seed_entry.configure(state='normal')
652         self.seed_entry.delete(0, END)
653         self.seed_entry.insert(0, "seed")
654         self.seed_entry.configure(state='disabled')
655 
656         self.resize_rgb_entry.configure(state='normal')
657         self.resize_rgb_entry.delete(0, END)
658         self.resize_rgb_entry.insert(0, "dimension")
659         self.resize_rgb_entry.configure(state='disabled')
660 
661         self.resize_depth_entry.configure(state='normal')
662         self.resize_depth_entry.delete(0, END)
663         self.resize_depth_entry.insert(0, "dimension")
664         self.resize_depth_entry.configure(state='disabled')
665 
666         self.lr_entry.delete(0, END)
667         self.batch_entry.delete(0, END)
668         self.ratio_entry.delete(0, END)
669         self.epoch_entry.delete(0, END)
670 
671         self.model = None
```

```
672         self.clear_frame(self.right_frame)
673 
674         self.chnl_var.set(0)
675         self.clear_tree_selection()
676         self.collapse_tree()
677 
678 
679         self.clear_log()
680         self.status_msg.set("Reset complete ...")
681 
682     def clear_log(self):
683         """Clears the log written in text widget"""
684 
685         self.text_log.configure(state=NORMAL)
686         self.text_log.delete(1.0, END)
687         self.text_log.configure(state=DISABLED)
688         self.status_msg.set("Log cleared ...")
689 
690     def clear_cache(self):
691         """Deletes the cached data if present"""
692 
693         for file_ in os.listdir("Data"):
694             os.remove("Data"+'/'+file_)
695         for file_ in os.listdir("temp"):
696             os.remove("temp"+'/'+file_)
697         for file_ in os.listdir("ppData"):
698             os.remove("ppData"+'/'+file_)
699         self.status_msg.set("Cache cleared ...")
700 
701     def data_summary(self):
702         """Gives the summary of dataset"""
703         if not self.read_flag:
704             print "No data found"
705         else:
706             print "--------------------Summary--------------------
    "
707             print "Dataset Size : {}".format(self.n_data)
708             print "RGB image dimension : {}".format((self.rgb_shape
    [1], self.rgb_shape[0]))
709             print "Depth image dimension : {}".format((self.
```

```
        depth_shape[1], self.depth_shape[0]))
710             print "-------------------------------------------------------
        "
711
712     def get_rgb_shape(self):
713         """Gives the dimension of rgb image"""
714
715         self.clear_log()
716         if not self.read_flag:
717             print "No data found"
718         else:
719             print "RGB image dimension : {}x{}".format(self.rgb_shape
        [1], self.rgb_shape[0])
720
721     def get_depth_shape(self):
722         """Gives the dimension of rgb image"""
723
724         self.clear_log()
725         if not self.read_flag:
726             print "No data found"
727         else:
728             print "Depth image dimension : {}x{}".format(self.
        depth_shape[1], self.depth_shape[0])
729
730     def get_dataset_size(self):
731         """Gives the number of data points"""
732
733         self.clear_log()
734         if not self.read_flag:
735             print "No data found"
736         else:
737             print "Dataset size : {}".format(self.n_data)
738
739     def collapse_tree(self):
740         """Collapse the network architecture tree"""
741
742         for item_ in self.tree_headings:
743             self.tree.item(item_, open=False)
744
745     def clear_tree_selection(self):
```

```
746         if len(self.tree.selection()) > 0:
747             self.tree.selection_remove(self.tree.selection()[0])
748 
749     def init_network(self):
750         try:
751             self.model = Sequential()
752             self.network_flag = True
753         except:
754             print "Unable to initialize network. Input all the
    constants !!!"
755 
756     def add_conv2d_frame(self):
757         self.clear_frame(self.right_frame)
758         lbl = Label(self.right_frame, text='Conv2D Layer Parameters',
     font="Verdana 10 bold")
759         lbl2 = Label(self.right_frame, text='Filters')
760         lbl3 = Label(self.right_frame, text='Kernel size')
761         lbl4 = Label(self.right_frame, text='Strides')
762         lbl5 = Label(self.right_frame, text="Padding")
763 
764         self.e_conv_filters = ttk.Entry(self.right_frame)
765         self.e_conv_kernel_size = ttk.Entry(self.right_frame)
766         self.e_conv_strides = ttk.Entry(self.right_frame)
767         self.e_conv_padding = ttk.Entry(self.right_frame)
768 
769         self.add_b = Button(self.right_frame, text='Add', command=
    self.add_conv2d_layer)
770         lbl.grid(row=0, columnspan=2)
771         lbl2.grid(row=1, column=0, sticky=E)
772         lbl3.grid(row=2, column=0, sticky=E)
773         lbl4.grid(row=3, column=0, sticky=E)
774         lbl5.grid(row=4, column=0, sticky=E)
775         self.e_conv_filters.grid(row=1, column=1)
776         self.e_conv_kernel_size.grid(row=2, column=1)
777         self.e_conv_strides.grid(row=3, column=1)
778         self.e_conv_padding.grid(row=4, column=1)
779 
780         self.add_b.grid(row=5, columnspan=2)
781 
782     def add_conv2d_layer(self):
```

```
783         if not self.read_flag:
784             print "No data found"
785             return
786         filters = int(self.e_conv_filters.get())
787         strides = tuple(map(int, self.e_conv_strides.get().strip(' ')
    .split(',')))
788         kernel_size = tuple(map(int, self.e_conv_kernel_size.get().
    strip(' ').split(',')))
789         padding = self.e_conv_padding.get()
790 
791         if not self.n_layers:
792             self.init_network()
793             self.model.add(Conv2D(filters=filters, kernel_size=
    kernel_size, strides=strides,
794                                   input_shape=self.rgb_shape,
    data_format=self.data_format,
795                                   padding=padding))
796             self.n_layers += 1
797         else:
798             self.model.add(Conv2D(filters=filters, kernel_size=
    kernel_size, strides=strides,
799                                   padding=padding))
800             self.n_layers += 1
801 
802         print "Conv2D layer added"
803 
804     def add_dense_frame(self):
805         """Adds dense layer to neural network"""
806 
807         self.clear_frame(self.right_frame)
808         lbl = Label(self.right_frame, text='Fully Connected Layer
    Parameters', font="Verdana 10 bold")
809         lbl2 = Label(self.right_frame, text='Units')
810         self.e_dense_units = ttk.Entry(self.right_frame)
811         self.add_b = Button(self.right_frame, text='Add', command=
    self.add_dense_layer)
812         lbl.grid(row=0, columnspan=2, sticky=N)
813         lbl2.grid(row=1, column=0, sticky=E)
814         self.e_dense_units.grid(row=1, column=1, sticky=N)
815         self.add_b.grid(row=2, columnspan=2, sticky=N)
```

```
816
817     def add_dense_layer(self):
818         if not self.read_flag:
819             print "No data found"
820             return
821         units = self.e_dense_units.get()
822         self.model.add(Dense(int(units)))
823         self.n_layers += 1
824         print "Fully connected layer added"
825
826     def add_drop_frame(self):
827         self.clear_frame(self.right_frame)
828         lbl = Label(self.right_frame, text='Dropout Layer Parameters'
    , font="Verdana 10 bold")
829         lbl2 = Label(self.right_frame, text='Rate')
830
831         self.e_drop_rate = ttk.Entry(self.right_frame)
832         self.add_b = Button(self.right_frame, text='Add', command=
    self.add_drop_layer)
833
834         lbl.grid(row=0, columnspan=2, sticky=N)
835         lbl2.grid(row=1, column=0, sticky=E)
836         self.e_drop_rate.grid(row=1, column=1, sticky=N)
837         self.add_b.grid(row=2, columnspan=2, sticky=N)
838
839     def add_drop_layer(self):
840         if not self.read_flag:
841             print "No data found"
842             return
843         rate = self.e_drop_rate.get()
844         self.model.add(Dropout(int(rate)))
845         self.n_layers += 1
846         print "Dropout layer added"
847
848     def add_reshape_frame(self):
849         self.clear_frame(self.right_frame)
850         lbl = Label(self.right_frame, text='Reshape Layer Parameters'
    , font="Verdana 10 bold")
851         lbl2 = Label(self.right_frame, text='Shape')
852
```

```
853         self.e_reshape_dim = ttk.Entry(self.right_frame)
854         self.add_b = Button(self.right_frame, text='Add', command=
    self.add_reshape_layer)
855 
856         lbl.grid(row=0, columnspan=2, sticky=N)
857         lbl2.grid(row=1, column=0, sticky=E)
858         self.e_reshape_dim.grid(row=1, column=1, sticky=N)
859         self.add_b.grid(row=2, columnspan=2, sticky=N)
860 
861     def add_reshape_layer(self):
862         if not self.read_flag:
863             print "No data found"
864             return
865         dim = tuple(map(int, self.e_reshape_dim.get().strip(' ').
    split(',')))
866         self.model.add(Reshape(dim))
867         self.n_layers += 1
868         print "Reshape layer added"
869 
870     def add_flatten_frame(self):
871         self.clear_frame(self.right_frame)
872         lbl = Label(self.right_frame, text='Flatten Layer Parameters'
    , font="Verdana 10 bold")
873 
874         self.add_b = Button(self.right_frame, text='Add', command=
    self.add_flatten_layer)
875 
876         lbl.grid(row=0, column=0, sticky=N)
877         self.add_b.grid(row=2, column=0, sticky=N)
878 
879     def add_flatten_layer(self):
880         if not self.read_flag:
881             print "No data found"
882             return
883         self.model.add(Flatten())
884         self.n_layers += 1
885         print "Flatten layer added"
886 
887     def add_padding_frame(self):
888         self.clear_frame(self.right_frame)
```

```
889         lbl = Label(self.right_frame, text='Padding Layer Parameters'
    , font="Verdana 10 bold")
890         lbl2 = Label(self.right_frame, text='Padding')
891 
892         self.e_padding = ttk.Entry(self.right_frame)
893         self.add_b = Button(self.right_frame, text='Add', command=
    self.add_padding_layer)
894 
895         lbl.grid(row=0, columnspan=2, sticky=N)
896         lbl2.grid(row=1, column=0, sticky=E)
897         self.e_padding.grid(row=1, column=1, sticky=N)
898         self.add_b.grid(row=2, columnspan=2, sticky=N)
899 
900     def add_padding_layer(self):
901         if not self.read_flag:
902             print "No data found"
903             return
904         padding = tuple(map(int, self.e_padding.get().strip(' ').
    split(',')))
905         self.model.add(Reshape(padding))
906         self.n_layers += 1
907         print "Padding added"
908 
909     def add_conv2dt_frame(self):
910         self.clear_frame(self.right_frame)
911         lbl = Label(self.right_frame, text='Conv2DTranspose Layer
    Parameters', font="Verdana 10 bold")
912         lbl2 = Label(self.right_frame, text='Filters')
913         lbl3 = Label(self.right_frame, text='Kernel size')
914         lbl4 = Label(self.right_frame, text='Strides')
915         lbl5 = Label(self.right_frame, text="Padding")
916 
917         self.e_conv_filters = ttk.Entry(self.right_frame)
918         self.e_conv_kernel_size = ttk.Entry(self.right_frame)
919         self.e_conv_strides = ttk.Entry(self.right_frame)
920         self.e_conv_padding = ttk.Entry(self.right_frame)
921 
922         self.add_b = Button(self.right_frame, text='Add', command=
    self.add_conv2dt_layer)
923         lbl.grid(row=0, columnspan=2)
```

```
924         lbl2.grid(row=1, column=0, sticky=E)
925         lbl3.grid(row=2, column=0, sticky=E)
926         lbl4.grid(row=3, column=0, sticky=E)
927         lbl5.grid(row=4, column=0, sticky=E)
928         self.e_conv_filters.grid(row=1, column=1)
929         self.e_conv_kernel_size.grid(row=2, column=1)
930         self.e_conv_strides.grid(row=3, column=1)
931         self.e_conv_padding.grid(row=4, column=1)
932 
933         self.add_b.grid(row=5, columnspan=2)
934 
935     def add_conv2dt_layer(self):
936         if not self.read_flag:
937             print "No data found"
938             return
939         filters = int(self.e_conv_filters.get())
940         strides = tuple(map(int, self.e_conv_strides.get().strip(' ')
    .split(',')))
941         kernel_size = tuple(map(int, self.e_conv_kernel_size.get().
    strip(' ').split(',')))
942         padding = self.e_conv_padding.get()
943 
944         if not self.n_layers:
945             self.init_network()
946             self.model.add(Conv2DTranspose(filters=filters,
    kernel_size=kernel_size, strides=strides,
947                                            input_shape=self.rgb_shape
    , data_format=self.data_format,
948                                            padding=padding))
949             self.n_layers += 1
950         else:
951             self.model.add(Conv2DTranspose(filters=filters,
    kernel_size=kernel_size, strides=strides,
952                                            padding=padding))
953             self.n_layers += 1
954         print "Conv2D Transpose layer added"
955 
956     def add_upsample_frame(self):
957         self.clear_frame(self.right_frame)
958         lbl = Label(self.right_frame, text='Upsampling2D Layer
```

```
Parameters', font="Verdana 10 bold")
959         lbl2 = Label(self.right_frame, text='Size')
960
961         self.e_upsample = ttk.Entry(self.right_frame)
962         self.add_b = Button(self.right_frame, text='Add', command=
self.add_upsample_layer)
963
964         lbl.grid(row=0, columnspan=2, sticky=N)
965         lbl2.grid(row=1, column=0, sticky=E)
966         self.e_upsample.grid(row=1, column=1, sticky=N)
967         self.add_b.grid(row=2, columnspan=2, sticky=N)
968
969     def add_upsample_layer(self):
970         if not self.read_flag:
971             print "No data found"
972             return
973         size = tuple(map(int, self.e_upsample.get().strip(' ').split(
',')))
974         self.model.add(UpSampling2D(size))
975         self.n_layers += 1
976         print "Upsample2D layer added"
977
978     def add_softmax_frame(self):
979         self.clear_frame(self.right_frame)
980         lbl = Label(self.right_frame, text='Softmax activation', font
="Verdana 10 bold")
981
982         self.add_b = Button(self.right_frame, text='Add', command=
self.add_softmax_layer)
983
984         lbl.grid(row=0, column=0, sticky=N)
985         self.add_b.grid(row=2, column=0, sticky=N)
986
987     def add_softmax_layer(self):
988         if not self.read_flag:
989             print "No data found"
990             return
991         self.model.add(Activation('softmax'))
992         self.n_layers += 1
993         print "Softmax activation added"
```

```
994
995     def add_sigmoid_frame(self):
996         self.clear_frame(self.right_frame)
997         lbl = Label(self.right_frame, text='Sigmoid activation', font
        ="Verdana 10 bold")
998
999         self.add_b = Button(self.right_frame, text='Add', command=
        self.add_sigmoid_layer)
1000
1001        lbl.grid(row=0, column=0, sticky=N)
1002        self.add_b.grid(row=2, column=0, sticky=N)
1003
1004    def add_sigmoid_layer(self):
1005        if not self.read_flag:
1006            print "No data found"
1007            return
1008        self.model.add(Activation('sigmoid'))
1009        self.n_layers += 1
1010        print "Sigmoid activation added"
1011
1012    def add_tanh_frame(self):
1013        self.clear_frame(self.right_frame)
1014        lbl = Label(self.right_frame, text='Tanh activation', font="
        Verdana 10 bold")
1015
1016        self.add_b = Button(self.right_frame, text='Add', command=
        self.add_tanh_layer)
1017
1018        lbl.grid(row=0, column=0, sticky=N)
1019        self.add_b.grid(row=2, column=0, sticky=N)
1020
1021    def add_tanh_layer(self):
1022        if not self.read_flag:
1023            print "No data found"
1024            return
1025        self.model.add(Activation('tanh'))
1026        self.n_layers += 1
1027        print "tanh activation added"
1028
1029    def add_relu_frame(self):
```

```
1030        self.clear_frame(self.right_frame)
1031        lbl = Label(self.right_frame, text='Relu activation', font="
        Verdana 10 bold")
1032
1033        self.add_b = Button(self.right_frame, text='Add', command=
        self.add_relu_layer)
1034
1035        lbl.grid(row=0, column=0, sticky=N)
1036        self.add_b.grid(row=2, column=0, sticky=N)
1037
1038    def add_relu_layer(self):
1039        if not self.read_flag:
1040            print "No data found"
1041            return
1042        self.model.add(Activation('relu'))
1043        print "Relu activation layer added"
1044        self.n_layers += 1
1045        print "relu activation added"
1046
1047    def add_maxpool_frame(self):
1048        self.clear_frame(self.right_frame)
1049        lbl = Label(self.right_frame, text='MaxPooling2D Layer
        Parameters', font="Verdana 10 bold")
1050        lbl2 = Label(self.right_frame, text='Pool size')
1051        lbl4 = Label(self.right_frame, text='Strides')
1052        lbl5 = Label(self.right_frame, text="Padding")
1053
1054        self.e_conv_kernel_size = ttk.Entry(self.right_frame)
1055        self.e_conv_strides = ttk.Entry(self.right_frame)
1056        self.e_conv_padding = ttk.Entry(self.right_frame)
1057
1058        self.add_b = Button(self.right_frame, text='Add', command=
        self.add_maxpool_layer)
1059        lbl.grid(row=0, columnspan=2)
1060        lbl2.grid(row=1, column=0, sticky=E)
1061        lbl4.grid(row=2, column=0, sticky=E)
1062        lbl5.grid(row=3, column=0, sticky=E)
1063        self.e_conv_kernel_size.grid(row=1, column=1)
1064        self.e_conv_strides.grid(row=2, column=1)
1065        self.e_conv_padding.grid(row=3, column=1)
```

```
1066
1067            self.add_b.grid(row=5, columnspan=2)
1068
1069        def add_maxpool_layer(self):
1070            if not self.read_flag:
1071                print "No data found"
1072                return
1073            if self.e_conv_strides.get() == 'None':
1074                strides = None
1075            else:
1076                strides = tuple(map(int, self.e_conv_strides.get().strip(
        ' ').split(',')))
1077            pool_size = tuple(map(int, self.e_conv_kernel_size.get().
        strip(' ').split(',')))
1078
1079            if not Application.n_layers:
1080                self.init_network()
1081                self.model.add(MaxPooling2D(pool_size=pool_size, strides=
        strides,
1082                                            input_shape=self.rgb_shape
        [0], data_format=self.data_format,
1083                                            padding=self.e_conv_padding))
1084                self.n_layers += 1
1085            else:
1086                self.model.add(MaxPooling2D(pool_size=pool_size, strides=
        strides,
1087                                            padding=self.e_conv_padding))
1088                self.n_layers += 1
1089            print "max pooling layer added"
1090
1091        def add_avgpool_frame(self):
1092            self.clear_frame(self.right_frame)
1093            lbl = Label(self.right_frame, text='AveragePooling2D Layer
        Parameters', font="Verdana 10 bold")
1094            lbl2 = Label(self.right_frame, text='Pool size')
1095            lbl4 = Label(self.right_frame, text='Strides')
1096            lbl5 = Label(self.right_frame, text="Padding")
1097
1098            self.e_conv_kernel_size = ttk.Entry(self.right_frame)
1099            self.e_conv_strides = ttk.Entry(self.right_frame)
```

```
1100        self.e_conv_padding = ttk.Entry(self.right_frame)
1101
1102        self.add_b = Button(self.right_frame, text='Add', command=
    self.add_avgpool_layer)
1103        lbl.grid(row=0, columnspan=2)
1104        lbl2.grid(row=1, column=0, sticky=E)
1105        lbl4.grid(row=2, column=0, sticky=E)
1106        lbl5.grid(row=3, column=0, sticky=E)
1107        self.e_conv_kernel_size.grid(row=1, column=1)
1108        self.e_conv_strides.grid(row=2, column=1)
1109        self.e_conv_padding.grid(row=3, column=1)
1110
1111        self.add_b.grid(row=5, columnspan=2)
1112
1113    def add_avgpool_layer(self):
1114        if not self.read_flag:
1115            print "No data found"
1116            return
1117        if self.e_conv_strides.get() == 'None':
1118            strides = tuple(map(int, self.e_conv_strides.get().strip(
    ' ').split(',')))
1119        else:
1120            strides = None
1121        pool_size = tuple(map(int, self.e_conv_kernel_size.get().
    strip(' ').split(',')))
1122
1123        if not Application.n_layers:
1124            self.init_network()
1125            self.model.add(AveragePooling2D(pool_size=pool_size,
    strides=strides,
1126                                            input_shape=self.
    rgb_shape[0], data_format=self.data_format,
1127                                            padding=self.
    e_conv_padding))
1128            self.n_layers += 1
1129        else:
1130            self.model.add(AveragePooling2D(pool_size=pool_size,
    strides=strides,
1131                                            padding=self.
    e_conv_padding))
```

```
1132            self.n_layers += 1
1133        print "average pooling layer added"
1134
1135    def add_mse_frame(self):
1136        self.clear_frame(self.right_frame)
1137        lbl = Label(self.right_frame, text='Mean squared error', font
    ="Verdana 10 bold")
1138
1139        self.add_b = Button(self.right_frame, text='Add', command=
    self.add_mse_layer)
1140
1141        lbl.grid(row=0, column=0, sticky=N)
1142        self.add_b.grid(row=2, column=0, sticky=N)
1143
1144    def add_mse_layer(self):
1145        if not self.read_flag:
1146            print "No data found"
1147            return
1148        self.loss = 'mean_squared_error'
1149        print "mean_squared_error loss added"
1150
1151    def add_mae_frame(self):
1152        self.clear_frame(self.right_frame)
1153        lbl = Label(self.right_frame, text='Mean absolute error',
    font="Verdana 10 bold")
1154
1155        self.add_b = Button(self.right_frame, text='Add', command=
    self.add_mae_layer)
1156
1157        lbl.grid(row=0, column=0, sticky=N)
1158        self.add_b.grid(row=2, column=0, sticky=N)
1159
1160    def add_mae_layer(self):
1161        if not self.read_flag:
1162            print "No data found"
1163            return
1164        self.loss = 'mean_absolute_error'
1165        print "Mean absolute error loss added"
1166
1167    def add_mse_log_frame(self):
```

```
1168        self.clear_frame(self.right_frame)
1169        lbl = Label(self.right_frame, text='Logarithmic mean squared
    error', font="Verdana 10 bold")
1170
1171        self.add_b = Button(self.right_frame, text='Add', command=
    self.add_mse_log_layer)
1172
1173        lbl.grid(row=0, column=0, sticky=N)
1174        self.add_b.grid(row=2, column=0, sticky=N)
1175
1176    def add_mse_log_layer(self):
1177        if not self.read_flag:
1178            print "No data found"
1179            return
1180        self.loss = 'mean_squared_logarithmic_error'
1181        print "logarithmic mean squared loss added"
1182
1183    def add_norm_frame(self):
1184        self.clear_frame(self.right_frame)
1185        lbl = Label(self.right_frame, text='BatchNormalization Layer'
    , font="Verdana 10 bold")
1186
1187        self.add_b = Button(self.right_frame, text='Add', command=
    self.add_norm_layer)
1188
1189        lbl.grid(row=0, column=0, sticky=N)
1190        self.add_b.grid(row=2, column=0, sticky=N)
1191
1192    def add_norm_layer(self):
1193        if not self.read_flag:
1194            print "No data found"
1195            return
1196        self.model.add(BatchNormalization())
1197        print "BatchNormalization layer added"
1198        self.n_layers += 1
1199
1200    def add_sgd_frame(self):
1201        self.clear_frame(self.right_frame)
1202        lbl = Label(self.right_frame, text='SGD optimizer', font="
    Verdana 10 bold")
```

```
1203        lbl2 = Label(self.right_frame, text='Decay')
1204        lbl3 = Label(self.right_frame, text='Momentum')
1205        lbl4 = Label(self.right_frame, text='Nesterov')
1206
1207        self.e_decay = ttk.Entry(self.right_frame)
1208        self.e_momentum = ttk.Entry(self.right_frame)
1209        self.e_nesterov = ttk.Entry(self.right_frame)
1210
1211        self.e_decay.insert(END, '0.0')
1212        self.e_momentum.insert(END, '0.0')
1213        self.e_nesterov.insert(END, 'False')
1214
1215        self.add_b = Button(self.right_frame, text='Add', command=
    self.add_sgd_layer)
1216
1217        lbl.grid(row=0, columnspan=2, sticky=N)
1218        lbl2.grid(row=1, column=0, sticky=E)
1219        lbl3.grid(row=2, column=0, sticky=E)
1220        lbl4.grid(row=3, column=0, sticky=E)
1221
1222        self.e_decay.grid(row=1, column=1, sticky=N)
1223        self.e_momentum.grid(row=2, column=1, sticky=N)
1224        self.e_nesterov.grid(row=3, column=1, sticky=N)
1225
1226        self.add_b.grid(row=4, columnspan=2, sticky=N)
1227
1228    def add_sgd_layer(self):
1229        if not self.read_flag:
1230            print "No data found"
1231            return
1232        decay = float(self.e_decay.get())
1233        momentum = float(self.e_momentum.get())
1234        nesterov = self.str_to_bool(self.e_nesterov.get())
1235        lr = float(self.lr_entry.get())
1236        self.optimizer = SGD(lr=lr, decay=decay, momentum=momentum,
1237                             nesterov=nesterov)
1238
1239        print "SGD optimizer added"
1240
1241    def add_rmsprop_frame(self):
```

```
1242        self.clear_frame(self.right_frame)
1243        lbl = Label(self.right_frame, text='RMSprop optimizer', font=
    "Verdana 10 bold")
1244        lbl2 = Label(self.right_frame, text='Decay')
1245        lbl3 = Label(self.right_frame, text='Rho')
1246        lbl4 = Label(self.right_frame, text='Epsilon')
1247
1248        self.e_rmsprop_decay = ttk.Entry(self.right_frame)
1249        self.e_rho = ttk.Entry(self.right_frame)
1250        self.e_epsilon = ttk.Entry(self.right_frame)
1251
1252        self.e_rmsprop_decay.insert(END, '0.0')
1253        self.e_rho.insert(END, '0.9')
1254        self.e_epsilon.insert(END, 'None')
1255
1256        self.add_b = Button(self.right_frame, text='Add', command=
    self.add_rmsprop_layer)
1257
1258        lbl.grid(row=0, columnspan=2, sticky=N)
1259        lbl2.grid(row=1, column=0, sticky=E)
1260        lbl3.grid(row=2, column=0, sticky=E)
1261        lbl4.grid(row=3, column=0, sticky=E)
1262
1263        self.e_rmsprop_decay.grid(row=1, column=1, sticky=N)
1264        self.e_rho.grid(row=2, column=1, sticky=N)
1265        self.e_epsilon.grid(row=3, column=1, sticky=N)
1266
1267        self.add_b.grid(row=4, columnspan=2, sticky=N)
1268
1269    def add_rmsprop_layer(self):
1270        if not self.read_flag:
1271            print "No data found"
1272            return
1273        decay = float(self.e_rmsprop_decay.get())
1274        rho = float(self.e_rho.get())
1275        epsilon = self.str_to_bool(self.e_epsilon.get())
1276        lr = float(self.lr_entry.get())
1277        self.optimizer = RMSprop(lr=lr, decay=decay, epsilon=epsilon,
    rho=rho)
1278        print "RMSprop optimizer added"
```

```
1279
1280     def add_adam_frame(self):
1281         self.clear_frame(self.right_frame)
1282         lbl = Label(self.right_frame, text='Adam optimizer', font="
         Verdana 10 bold")
1283         lbl2 = Label(self.right_frame, text='beta_1')
1284         lbl3 = Label(self.right_frame, text='beta_2')
1285         lbl4 = Label(self.right_frame, text='Epsilon')
1286         lbl5 = Label(self.right_frame, text='Decay')
1287
1288         self.e_beta1 = ttk.Entry(self.right_frame)
1289         self.e_beta2 = ttk.Entry(self.right_frame)
1290         self.e_adam_epsilon = ttk.Entry(self.right_frame)
1291         self.e_adam_decay = ttk.Entry(self.right_frame)
1292
1293         self.e_beta1.insert(END, '0.9')
1294         self.e_beta2.insert(END, '0.999')
1295         self.e_adam_epsilon.insert(END, '1e-8')
1296         self.e_adam_decay.insert(END, '0.0')
1297
1298         self.add_b = Button(self.right_frame, text='Add', command=
         self.add_adam_layer)
1299
1300         lbl.grid(row=0, columnspan=2)
1301         lbl2.grid(row=1, column=0, sticky=E)
1302         lbl3.grid(row=2, column=0, sticky=E)
1303         lbl4.grid(row=3, column=0, sticky=E)
1304         lbl5.grid(row=4, column=0, sticky=E)
1305
1306         self.e_beta1.grid(row=1, column=1, sticky=N)
1307         self.e_beta2.grid(row=2, column=1, sticky=N)
1308         self.e_adam_epsilon.grid(row=3, column=1, sticky=N)
1309         self.e_adam_decay.grid(row=4, column=1, sticky=N)
1310
1311         self.add_b.grid(row=5, columnspan=2, sticky=N)
1312
1313     def add_adam_layer(self):
1314         if not self.read_flag:
1315             print "No data found"
1316             return
```

```
1317            decay = float(self.e_adam_decay.get())
1318            epsilon = float(self.e_adam_epsilon.get())
1319            beta_1 = float(self.e_beta1.get())
1320            beta_2 = float(self.e_beta2.get())
1321            lr = float(self.lr_entry.get())
1322
1323            self.optimizer = Adam(lr=lr, beta_1=beta_1, beta_2=beta_2,
1324                                  decay=decay, epsilon=epsilon)
1325            print "Adam optimizer added"
1326
1327        def tree_control(self, event):
1328            if self.tree.selection()[0] == 'dense':
1329                self.add_dense_frame()
1330
1331            elif self.tree.selection()[0] == 'drop':
1332                self.add_drop_frame()
1333
1334            elif self.tree.selection()[0] == 'reshape':
1335                self.add_reshape_frame()
1336
1337            elif self.tree.selection()[0] == 'flatten':
1338                self.add_flatten_frame()
1339
1340            elif self.tree.selection()[0] == 'conv2d':
1341                self.add_conv2d_frame()
1342
1343            elif self.tree.selection()[0] == 'conv2dt':
1344                self.add_conv2dt_frame()
1345
1346            elif self.tree.selection()[0] == 'padding':
1347                self.add_padding_frame()
1348
1349            elif self.tree.selection()[0] == 'upsample':
1350                self.add_upsample_frame()
1351
1352            elif self.tree.selection()[0] == 'softmax':
1353                self.add_softmax_frame()
1354
1355            elif self.tree.selection()[0] == 'sigmoid':
1356                self.add_sigmoid_frame()
```

```
1357
1358            elif self.tree.selection()[0] == 'tanh':
1359                self.add_tanh_frame()
1360
1361            elif self.tree.selection()[0] == 'relu':
1362                self.add_relu_frame()
1363
1364            elif self.tree.selection()[0] == 'maxpool':
1365                self.add_maxpool_frame()
1366
1367            elif self.tree.selection()[0] == 'avgpool':
1368                self.add_avgpool_frame()
1369
1370            elif self.tree.selection()[0] == 'mse':
1371                self.add_mse_frame()
1372
1373            elif self.tree.selection()[0] == 'mae':
1374                self.add_mae_frame()
1375
1376            elif self.tree.selection()[0] == 'mse_log':
1377                self.add_mse_log_frame()
1378
1379            elif self.tree.selection()[0] == 'bnorm':
1380                self.add_norm_frame()
1381
1382            elif self.tree.selection()[0] == 'sgd':
1383                self.add_sgd_frame()
1384
1385            elif self.tree.selection()[0] == 'rmsprop':
1386                self.add_rmsprop_frame()
1387
1388            elif self.tree.selection()[0] == 'adam':
1389                self.add_adam_frame()
1390
1391        def network_summary(self):
1392
1393            self.clear_log()
1394            if not self.network_flag:
1395                print "No model found"
1396            else:
```

```
1397            self.model.summary()
1398
1399    def compile_model(self):
1400        if not self.network_flag:
1401            self.clear_log()
1402            print "No network architecture found"
1403            return
1404        elif self.loss is None:
1405            print "No loss function found"
1406            return
1407        elif self.optimizer is None:
1408            print "No optimizer found"
1409            return
1410        else:
1411            loss = self.loss
1412            optimizer = self.optimizer
1413            self.model.compile(loss=loss, optimizer=optimizer)
1414            self.compile_flag = True
1415            print "Model compiled"
1416
1417    def train_model(self):
1418        if not self.compile_flag:
1419            self.clear_log()
1420            print "Compile the model first"
1421            return
1422        else:
1423            graph = self.str_to_bool(self.e_graph.get())
1424            self.status_msg.set("Loading Data ...")
1425            print "Loading data"
1426            train_x = np.load("ppData/pp_rgb_data.npy")
1427            train_y = np.load("ppData/pp_depth_data.npy")
1428
1429            self.status_msg.set("Training ...")
1430
1431            epochs = int(self.epoch_entry.get())
1432            batch_size = int(self.batch_entry.get())
1433            verbose = int(self.e_verbose.get())
1434            validation_split = float(self.ratio_entry.get())
1435            callbacks = [PlotLosses(self.graph_page)]
1436            shuffle = self.str_to_bool(self.e_shuffle.get())
```

```
1437            self.clear_log()
1438
1439            if graph:
1440                self.history = self.model.fit(train_x, train_y,
    epochs=epochs, batch_size=batch_size,
1441                                              verbose=verbose,
    validation_split=validation_split,
1442                                              callbacks=callbacks,
    shuffle=shuffle)
1443            else:
1444                self.history = self.model.fit(train_x, train_y,
    epochs=epochs, batch_size=batch_size,
1445                                              verbose=verbose,
    validation_split=validation_split,
1446                                              shuffle=shuffle)
1447
1448            self.train_flag = True
1449            del train_x, train_y
1450            self.status_msg.set("Training done...")
1451            print "Training completed"
1452
1453    def save_model(self):
1454
1455        if not self.network_flag:
1456            print "No model found"
1457        else:
1458            try:
1459                f_name = asksaveasfilename(defaultextension=".json")
1460                model_json = self.model.to_json()
1461                with open(f_name, "w") as json_file:
1462                    json_file.write(model_json)
1463            except:
1464                pass
1465
1466    def save_weights(self):
1467        if not self.train_flag:
1468            print "Train the model first"
1469        else:
1470            try:
1471                f_name = asksaveasfilename(defaultextension=".h5")
```

```
1472                 self.model.save_weights(f_name)
1473             except:
1474                 pass
1475
1476     def reset_model(self):
1477         self.model = None
1478         self.n_layers = 0
1479         self.network_flag = False
1480
1481     def load_model(self):
1482         if self.model is not None:
1483             msg = tkMessageBox.askokcancel("Warning", "Existing model
     will be overwritten.")
1484             if msg:
1485                 pass
1486             else:
1487                 return
1488         try:
1489             f_name = askopenfilename(filetypes=[('*.yaml', '*.json')
    ])
1490             json_file = open(f_name, 'r')
1491             loaded_model_json = json_file.read()
1492             json_file.close()
1493             self.model = model_from_json(loaded_model_json)
1494             print "Model Loaded"
1495             self.network_flag = True
1496             self.n_layers += 1
1497         except:
1498             pass
1499
1500     def load_weights(self):
1501         if self.network_flag:
1502             if self.train_flag is not None:
1503                 msg = tkMessageBox.askokcancel("Warning", "Existing
    weights will be overwritten.")
1504                 if msg:
1505                     pass
1506                 else:
1507                     return
1508             try:
```

```
1509                f_name = askopenfilename(filetypes=[('HDF5 file', '*.
    h5')])
1510                self.model.load_weights(f_name)
1511                self.train_flag = True
1512                print "Weights loaded"
1513            except:
1514                pass
1515        else:
1516            print err_msg['nmf']
1517
1518    def remove_last_layer(self):
1519        try:
1520            self.model.layers.pop()
1521            print "Last layer removed"
1522        except:
1523            print "Something went wrong"
1524
1525    def save_model_history(self):
1526        if not self.train_flag:
1527            print "Train the model first"
1528        else:
1529            try:
1530                f_name = asksaveasfilename(defaultextension=".mat")
1531                sio.savemat(f_name, self.history.history)
1532            except:
1533                pass
1534
1535    def load_model_history(self):
1536        if self.history is not None:
1537            msg = tkMessageBox.askokcancel("Warning", "Existing
    history will be overwritten.")
1538            if msg:
1539                pass
1540            else:
1541                return
1542        try:
1543            f_name = askopenfilename(filetypes=['*.mat'])
1544            self.history = sio.loadmat(f_name)
1545        except:
1546            pass
```

```
1547
1548        @staticmethod
1549        def init_dynamic_plot(frame):
1550            frame = Frame(frame)
1551            frame.pack()
1552            canvas = FigureCanvasTkAgg(f, frame)
1553            canvas.draw()
1554            canvas.get_tk_widget().pack(side=TOP, fill=BOTH, expand=True)
1555
1556            toolbar = NavigationToolbar2TkAgg(canvas, frame)
1557            toolbar.update()
1558            canvas._tkcanvas.pack(side=TOP, fill=BOTH, expand=False)
1559
1560        def read_data(self, path, depth=False):
1561            """Reads image data from a given directory"""
1562            if depth:
1563                verb = "Reading Depth data"
1564            else:
1565                verb = "Reading rgb data"
1566            try:
1567                file_list = os.listdir(path)
1568                file_list.sort()
1569            except OSError:
1570                print "Directory does not exist"
1571                raise Exception('DirErr')
1572            if len(file_list) == 0:
1573                print "Directory is empty"
1574                raise Exception('DirErr')
1575            images = []
1576            self.progress_bar(0, len(file_list), prefix='Progress:',
    length=50)
1577            for i, image in enumerate(file_list):
1578                try:
1579                    if not depth:
1580                        img = cv2.imread(path + '/' + image)
1581                        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
1582                        images.append(img)
1583                    else:
1584                        img = cv2.imread(path + '/' + image)
1585                        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
1586                    images.append(img)
1587            except OSError:
1588                print "Directory may not contain image files"
1589                raise Exception('DirErr')
1590            self.progress_bar(i + 1, len(file_list), prefix='Progress
    :', suffix=verb, length=50)
1591        return np.array(images)
1592
1593    def scale_depth_data(self, data):
1594        """s2cale all the depth images to values between 0 and 1"""
1595
1596        p_data = []
1597        data = data.astype(np.float32)
1598        self.progress_bar(0, data.shape[0], prefix='Progress:',
    suffix='Scaling depth', length=50)
1599        for i in range(data.shape[0]):
1600            max_depth = np.amax(data[i])
1601            min_depth = np.amin(data[i])
1602            d = np.divide(np.subtract(data[i], min_depth), (max_depth
    - min_depth))
1603            p_data.append(d)
1604            self.progress_bar(i+1, data.shape[0], prefix='Progress:',
    suffix='Scaling depth', length=50)
1605        return np.array(p_data)
1606
1607    def scale_rgb_data(self, data):
1608        """scale all the rgb images to values between 0 and 1"""
1609
1610        p_data = []
1611        self.progress_bar(0, data.shape[0], prefix='Progress:',
    length=50)
1612        for i in range(data.shape[0]):
1613            d = np.divide(data[i], 255.0)
1614            p_data.append(d)
1615            self.progress_bar(i+1, data.shape[0], prefix='Progress:',
    length=50)
1616        return np.array(p_data)
1617
1618    def resize_data(self, data, size):
1619        p_data = []
```

```
1620            self.progress_bar(0, data.shape[0], prefix='Progress:',
    length=50)
1621            for i in range(data.shape[0]):
1622                p_data.append(cv2.resize(data[i], size))
1623                self.progress_bar(i+1, data.shape[0], prefix='Progress:',
     suffix='Resizing', length=50)
1624            return np.array(p_data)
1625
1626        def get_input_shape(self):
1627            layer0 = self.model.layers[0]
1628            input_shape = layer0.input_shape
1629            return input_shape[1:]
1630
1631        def read_predict(self):
1632            if not self.network_flag:
1633                print err_msg['nmf']
1634                return
1635            elif not self.train_flag:
1636                print err_msg['nwf']
1637                return
1638            try:
1639                image_name = askopenfilename()
1640                img = cv2.imread(image_name)
1641                self.plot_predictions(img)
1642                self.nb.select(1)
1643            except:
1644                pass
1645
1646        def predict(self, img):
1647            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
1648            size = self.get_input_shape()
1649            size = size[1], size[0]
1650            img = cv2.resize(img, size)
1651            img = np.expand_dims(img, axis=0)
1652            depth = self.model.predict(img)
1653            d_shape = depth.shape[1:]
1654            depth = depth.reshape(d_shape)
1655            return depth
1656
1657        def plot_predictions(self, img):
```

```
1658
1659        # Plotting RGB image
1660        f1 = Figure(figsize=(5, 3), dpi=100)
1661        plt1 = f1.add_subplot(111)
1662        plt1.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
1663        canvas = FigureCanvasTkAgg(f1, self.images_page)
1664        canvas.draw()
1665        canvas.get_tk_widget().pack(side=TOP, fill=BOTH, expand=True)
1666
1667        toolbar = NavigationToolbar2TkAgg(canvas, self.images_page)
1668        toolbar.update()
1669        canvas._tkcanvas.pack(side=TOP, fill=BOTH, expand=False)
1670
1671        # Plotting depth image
1672        depth = self.predict(img)
1673        print depth.shape
1674        f2 = Figure(figsize=(5, 3), dpi=100)
1675        plt2 = f2.add_subplot(111)
1676        plt2.imshow(depth, cmap='gray')
1677        canvas = FigureCanvasTkAgg(f2, self.depth_page)
1678        canvas.draw()
1679        canvas.get_tk_widget().pack(side=TOP, fill=BOTH, expand=True)
1680
1681        toolbar = NavigationToolbar2TkAgg(canvas, self.depth_page)
1682        toolbar.update()
1683        canvas._tkcanvas.pack(side=TOP, fill=BOTH, expand=False)
1684
1685    def predict_folder(self):
1686        if not self.network_flag:
1687            print err_msg['nmf']
1688            return
1689        elif not self.train_flag:
1690            print err_msg['nwf']
1691            return
1692        dirname = askdirectory(parent=root, initialdir=os.getcwd(),
    title='Please select a directory')
1693        try:
1694            self.rgb_entry.delete('0', END)
1695            self.rgb_entry.insert(END, dirname)
1696        except:
```

```
1697                 return
1698
1699             save_dir = os.path.split(dirname)[0]
1700             file_list = os.listdir(dirname)
1701             np_path = os.path.join(save_dir, "depth_pred_np_arr")
1702             img_path = os.path.join(save_dir, "depth_predictions")
1703             if not os.path.exists(np_path):
1704                 os.makedirs(np_path)
1705             if not os.path.exists(img_path):
1706                 os.makedirs(img_path)
1707
1708             self.progress_bar(0, len(file_list), prefix='Progress:',
    length=50)
1709             for i, file_ in enumerate(file_list):
1710                 img = cv2.imread(os.path.join(dirname, file_))
1711                 depth = self.predict(img)
1712                 np.save(os.path.join(np_path, file_.split('.')[0]), depth
    )
1713                 depth = (depth - np.amin(depth))/(np.amax(depth)-np.amin(
    depth))*255
1714                 cv2.imwrite(os.path.join(img_path, file_), depth)
1715                 self.progress_bar(i+1, len(file_list), prefix='Progress:'
    , length=50)
1716
1717
1718 def main():
1719     Application(root)
1720     root.mainloop()
1721
1722
1723 if __name__ == "__main__":
1724     main()
```

REFERENCES

1. **Chollet, F.** (2015). keras. `https://github.com/fchollet/keras`.

2. **E. Rumelhart, D.**, **G. E. Hinton**, and **R. J. Williams** (1986). Learning representations by back propagating errors. **323**, 533–536.

3. **Eigen, D.**, **C. Puhrsch**, and **R. Fergus**, Depth map prediction from a single image using a multi-scale deep network. *In IEEE*. 2013.

4. **Geiger, A.**, **P. Lenz**, **C. Stiller**, and **R. Urtasun** (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.

5. **Godard, C.**, **O. Mac Aodha**, and **G. J. Brostow**, Unsupervised monocular depth estimation with left-right consistency. *In CVPR*. 2017.

6. **Kingma, D. P.** and **J. Ba** (2014). Adam: A method for stochastic optimization. *CoRR*, **abs/1412.6980**. URL `http://arxiv.org/abs/1412.6980`.

7. **Krajnik, T.**, **V. Vonásek**, **D. Fišer**, and **J. Faigl** (2011). Ar-drone as a platform for robotic research and education, 172–186.

8. **Krizhevsky, A.**, **I. Sutskever**, and **G. E. Hinton**, Imagenet classification with deep convolutional neural networks. *In Advances in Neural Information Processing Systems*.

9. **Levin, A.**, **D. Lischinski**, and **Y. Weiss**, Colorization using optimization. *In ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04. ACM, New York, NY, USA, 2004. URL `http://doi.acm.org/10.1145/1186562.1015780`.

10. **Machida, E.**, **M. Cao**, **T. Murao**, and **H. Hashimoto**, Human motion tracking of mobile robot with kinect 3d sensor. *In 2012 Proceedings of SICE Annual Conference (SICE)*. 2012. ISSN pending.

11. **Maturana, D.** and **S. Scherer**, 3d convolutional neural networks for landing zone detection from lidar. *In 2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015. ISSN 1050-4729.

12. **Nathan Silberman, P. K., Derek Hoiem** and **R. Fergus**, Indoor segmentation and support inference from rgbd images. *In ECCV*. 2012.

13. **Ning, X.** and **G. Guo** (2013). Assessing spinal loading using the kinect depth sensor: A feasibility study. *IEEE Sensors Journal*, **13**(4), 1139–1140. ISSN 1530-437X.

14. **Pagliari**, **D. Pinto**, and **Livio** (2015). Calibration of kinect for xbox one and comparison between the two generations of microsoft sensors. **11**, 27569–27589.

15. **Pagliari, D.** and **L. Pinto** (2015). Calibration of kinect for xbox one and comparison between the two generations of microsoft sensors. *Sensors*, **15**(11), 27569–27589. ISSN 1424-8220. URL `http://www.mdpi.com/1424-8220/15/11/27569`.

16. **Rojas Vidovic, J.**, **D. Gallipoli**, and **S. J. Wheeler** (2012). Image analysis of strains in soils subjected to wetting and drying. **35**.

17. **Rumelhart, D. E.**, **G. E. Hinton**, and **R. J. Williams** (1986). Learning representations by back-propagating errors. **323**, 533–536.

18. **Sabour, S.**, **N. Frosst**, and **G. E. Hinton** (2017). Dynamic routing between capsules. *CoRR*, **abs/1710.09829**. URL `http://arxiv.org/abs/1710.09829`.

19. **Sun, C.**, **T. Zhang**, **B. K. Bao**, **C. Xu**, and **T. Mei** (2013). Discriminative exemplar coding for sign language recognition with kinect. *IEEE Transactions on Cybernetics*, **43**(5), 1418–1428. ISSN 2168-2267.

20. **Tai, L.**, **S. Li**, and **M. Liu**, A deep-network solution towards model-less obstacle avoidance. *In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016.

21. **Zhang, Z.**, **W. Liu**, **V. Metsis**, and **V. Athitsos**, A viewpoint-independent statistical method for fall detection. *In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. 2012. ISSN 1051-4651.