School of Computing
FACULTY OF ENGINEERING

Surface Normal Estimation from RGB Images

Mohammad Kaykanloo

Submitted in accordance with the requirements for the degree of
MSc Advanced Computer Science (Intelligent Systems)

2016/2017

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Project Report	Report	SSO (27/09/17)
Dataset	URL	Supervisor, Assessor (27/09/17)
Implementation	Software codes and URL	Supervisor, Assessor (27/09/17)

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) _____

Summary

This project explored a deep learning approach for the task of pixel-wise surface normal estimation from monocular RGB images. At first, an analysis of the previous research was conducted. Then, data required for use in this project was obtained from two publicly available data sets. Software code was developed to compute the ground truth surface normal maps based on two different methods and the results were used to produce the final data sets.

A deep learning pipeline was developed and a baseline network architecture was implemented. Based on the insights gained from the literature review and analysis of the state of the art methods, different modifications to the baseline model were investigated. In particular, improvements over baseline results were explored in respect to two aspects: network architectures, and the quality of the data set.

Finally, well-established evaluation metrics were implemented and the quantitative and qualitative result of evaluation were presented and compared to the state of the art methods.

Contents

List of Tables

Chapter 1

Introduction

1.1 Context

Arguably, humans rely on their sense of sight more than any other sense in understanding the world around them. Finding our way in a busy street, locating a book on a shelf in a library and interacting with hundreds of objects in our everyday life are some of examples that require extensive use of visual cues. We can not only recognise objects and locations, but also create a 3D structure of the scene in our mind. Even by looking at a single image, we can answer complex questions about position, orientation, and relation of objects.

As technologies such as augmented reality, self-driving cars, drones, and robotics in general, progress towards finding more use cases in our every day lives, the need for better scene understanding in real 3D space increases. By advent of intelligent personal assistant softwares and smarter applications, providing more useful features and better user experience depends on understanding the real world that users live in it.

Scene understanding is one of the most fundamental problems in computer vision. It is very challenging to solve this problem by only using 2D visual data captured by ordinary cameras. Thus, hardware manufacturers have tried to make this task easier by providing more information about the 3D space; using a variety of camera arrays, structured light 3D scanners, and time-of-flight (ToF) sensors. 3D vision systems use these sensors to measure, map, locate, and reconstruct the three dimensional structure of the environment for visual perception tasks.

Data provided by these sensors, plays a crucial role in many currently popular products in fields of augmented reality, robotics, biometric facial recognition, and gesture detection (e.g. Microsoft Kinect, Apple FaceID, and Tesla Autopilot). But in many applications, access to these sensors is not simply possible. For example, the vast amount of images and videos currently available on the internet, provide no extra 3D information about their content. On the other hands, in comparison to humans, software systems are unable to fully exploit the visual information that 2D images contain. Therefore, there is still a strong need for developing more advanced scene understanding and reconstruction systems that are based on 2D input data.

Estimating the orientation of surfaces in an image, is an important step in reconstructing a 3D model of the scene. Recently, the surface normal estimation from monocular RGB images, has been an active area of research in computer vision. Tackling this problem is the main topic of this M.Sc. project.

1.2 Problem Statement

Before any further discussion, a precise definition of the problem is needed. In this project, the problem of pixel-wise surface normal estimation from monocular RGB images of the indoor scenes is addressed.

The RGB images are captured from indoor environments with realistic scene layout such as bedrooms, living rooms, kitchens, and offices. Figure 1.1 illustrates a typical scene layout and viewpoint.

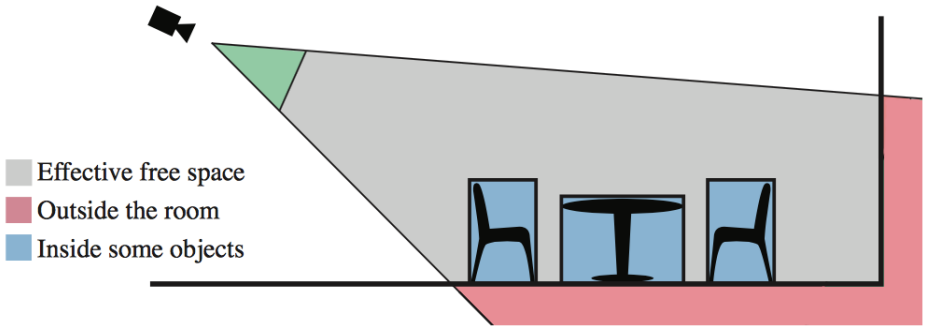

Figure 1.1: Schematic view of a scene, adapted from [1, p. 5]

Given an RGB image as input, the estimated surface normal map is expected as output. Samples of RGB images and the corresponding normal maps are illustrated in figure 1.2.

Figure 1.2: Top: RGB images of indoor scenes, Bottom: corresponding surface normal maps
Normal legend: $blue \rightarrow X(horizontal)$; $green \rightarrow Y(vertical)$; $red \rightarrow Z(perpendicular\ to\ image)$.

1.3 Proposed Solution

In last few years, deep learning has made major breakthroughs in field of computer vision. Deep neural networks (DNNs) have achieved impressive performance in challenging tasks such as image classification, semantic segmentation, and image compression. DNNs have started to even exceed human performance in tasks such as image classification [2]. Therefore it is not surprising that current state of the art solutions for the task of surface normal estimation are all based on deep learning.

Based on the promising achievements of deep neural networks, and the huge gap between the results yielded by using these networks in comparison to traditional methods in similar problems, a solution based on DNNs was chosen as the preferred approach for this project. At first, a baseline network architecture was implemented and after training the model, the estimated surface normal maps were evaluated. Afterwards, based on the insights gained from the literature review and analysis of the state of the art methods, different modifications to the baseline model were investigated. In particular, improvements over baseline results were explored in respect to two aspects: network architectures, and the quality of data set.

1.4 Project Aim

This project explored a deep learning approach for the task of surface normal estimation. The data needed for training and evaluation of the system, was acquired by using publicly available data sets. Different network architectures were implemented and finally, well-established evaluation metrics were used to compare the results with the previous research in this field.

1.4.1 Objectives

- Data set and ground truth acquisition
- Analysing the state-of-the-art methods
- Proposing a new method or improvements over current methods and implementing it
- Evaluating the method and comparing with other methods

1.4.2 Deliverables

- Data set
- Analysis of the state-of-the-art methods
- Source code and algorithms
- Evaluation results

1.5 Planning and Project Management

Before starting the work on this project, a period of two months was spent on background study and literature review. During this period, works relevant to the subject of this project

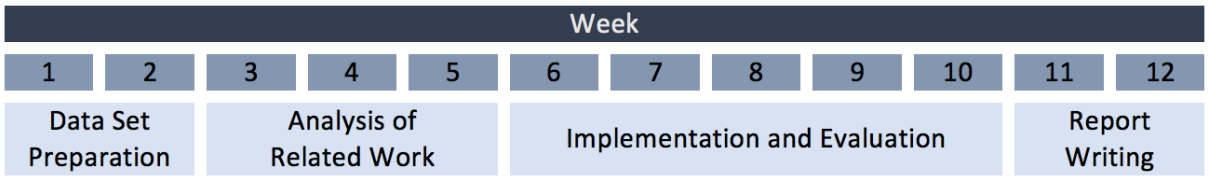

Figure 1.3: Initial project plan

were identified, and the theoretical aspects of the solutions discussed in existing research, were studied. A better understanding of the problem, and a proposed solution were the outcomes of this process.

1.5.1 Initial Plan

To guide the project execution, an initial plan was devised and the project was broken down to four main tasks:

- *Data Acquisition and Preparation*: preparing the data sets required for training and evaluation of the system
- *Analysis of Related Work*: studying the source codes released by relevant works and reproducing their results
- *Implementation and Evaluation*: implementing the system and evaluating the results
- *Report Writing*: writing the project report

Based on these tasks, an initial project schedule for a period of three months was created (figure 1.3).

1.5.2 Revised Plan

In reality, the time that was spent on different tasks was different with what initially was planned. Because of difficulties in data set preparation task, one more week than expected was spent on this task. Also, analysis of related work took one week less than expected. After two months, in a progress meeting with supervisor and assessor of the project, the project plan was reconsidered.

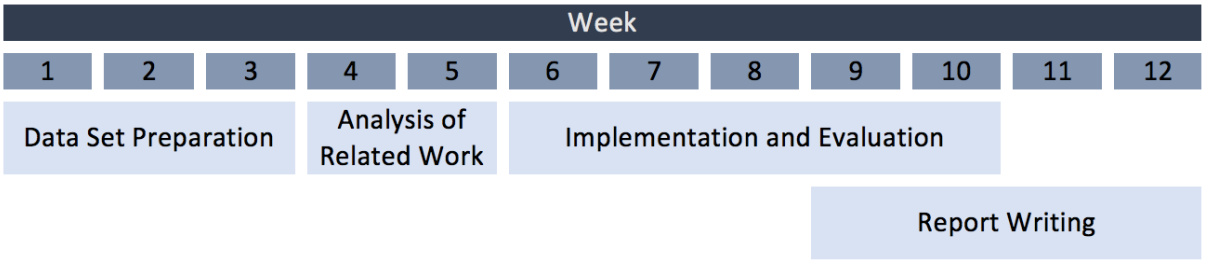

Figure 1.4: Revised project plan

Figure 1.4 illustrates the revised project plan. Based on the provided suggestions, the time considered for writing the project report was extended to four weeks, starting in parallel to last weeks of project implementation and evaluation.

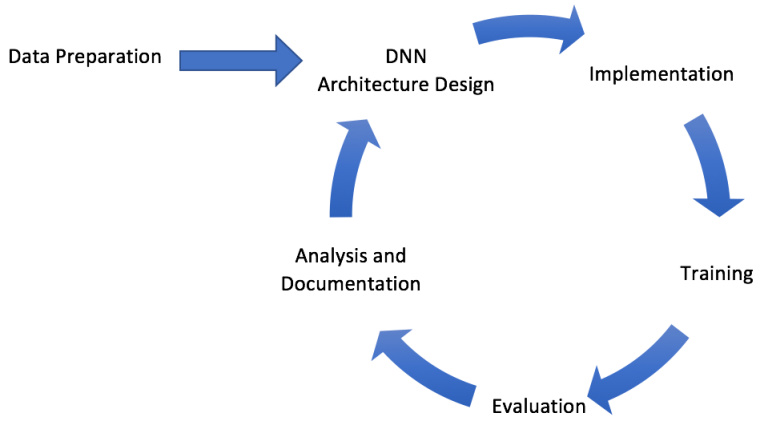

Figure 1.5: Project Workflow

1.5.3 Methodology

The methodology used for this project was mainly based on performing a series of experiments to gradually design better network architectures. The implementation and evaluation task was conducted based on the workflow illustrated in figure 1.5. After preparing the data sets, for each experiment, a sequence of activities was carried out: designing a network architecture, implementation, training the network, evaluation of the predicted outputs, analysis of the results and documentation. After each experiment, based on the outcome, a new experiment was designed to investigate a hypothesis or improve the network architecture. This process was repeated for the limited time that was available to this phase of project.

1.5.4 Risk Assessment

After literature review, an assessment of the risks associated with this project was carried out. Consequently, difficulty in reproducing the results of the previous research, was identified as one of the main risks in this project. Therefore, it was decided that in case of facing such problem, instead of reproducing the results, the reported results would be used as a basis for comparing the outcome of this project with the state of the art.

Chapter 2

Background

2.1 Deep Learning

In last few years, Deep Learning has been the state of the art approach for solving most computer vision problems. All related work to this project that has been carried out in recent years, are based on Deep Learning. Also, based on the promising results of this approach, it was chosen as the main method for this project. Therefore, before discussing any related work, it is best to have a brief introduction to Deep Learning.

2.1.1 Overview

Deep Learning is a sub-field of machine learning concerned with Artificial Neural Networks (ANNs) with more than a few layers. ANNs are computing systems inspired by biological neural networks in the brain. ANNs, like brain, are based on collection of connected units called artificial neurons. Typically, these neurons are organised in layers. Figure 2.1 demonstrates the architecture of a simple ANN.

Data is fed to the network by input neurons in form of numerical vectors. Input neurons are connected to other neurons in the network and pass the data through these connections. Each connection has a weight associated with it. Any vector of data that passes through a connection is multiplied by the weight of that connection. At each neuron, a function (e.g. hyperbolic tangent) is applied to all the input vectors that reach the neuron through different connections, and the result is passed to the other connected neurons. Data flows though the network by input neurons; each layer of neurons performs some kind of transformation on it, and the end result becomes available at output neurons.

The ANN in figure 2.1 is a very simple network with just three layers. Deep neural networks usually consist of many more layers with different types of neurons. For example GoogLeNet [3], which was the winner of The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014, is a 22 layer neural network with 7 different types of layers. Some of the current state of the art deep neural networks have more than a hundred layers.

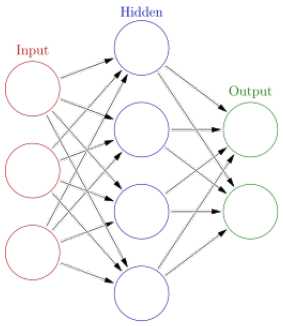

Figure 2.1: A simple Artificial Neural Network (ANN)
By Glosser.ca [CC BY-SA 3.0], via Wikimedia Commons

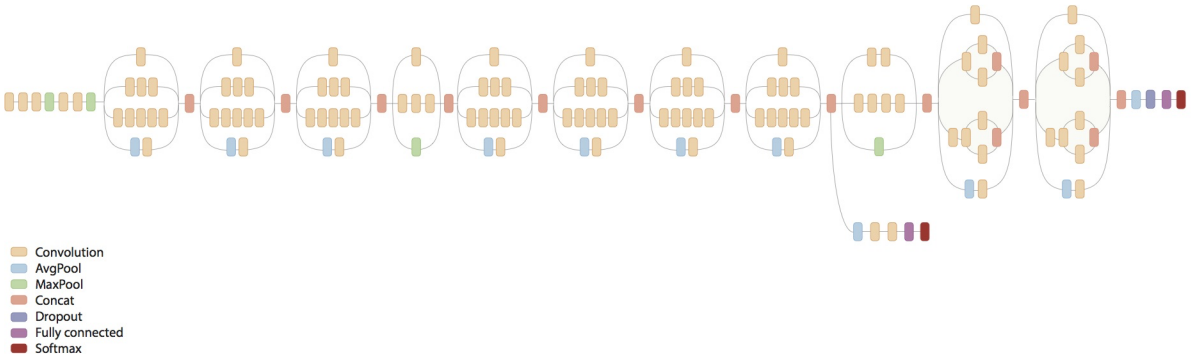

Figure 2.2: GoogLeNet [3], a deep neural network
Picture courtesy of `https://github.com/tensorflow/models/tree/master/inception`

Two main applications of deep neural networks are in classification and regression problems. In classification problems, the aim is to predict the correct category for each input sample from a set of predefined classes. For example, predicting the specie of animals by looking at their pictures. On the other hand, in regression problems, the goal is to estimate the relationship between variables. For example, estimating the age of people (dependent variable) by looking at a photo of their face (facial structure as independent variable); Or in case of this project, the pixel-wise surface normals based on the RGB image of a scene.

To achieve these goals, a suitable network architecture (based on the problem) should be selected and the optimal weights for its connections have to be determined. Network architecture is usually chosen based on the architectures that are used in similar problems and the prior experience of the expert. But it is not possible to manually determine the optimal weights of a network; because deep neural networks usually have millions of parameters (e.g. 61 million for AlexNet [4] and 138 million for VGG-16 [5] architecture) [6]. In practice, usually these parameters are initialised by random numbers and the optimal value of them is learned in a process called training. In next sections, the overall network architecture, building-blocks of the deep neural networks which are relevant to this project, and the training process are introduced.

2.1.2 Architecture

In last few years, Convolutional Neural Networks (CNNs) have been the most popular type of deep neural networks in field of computer vision [7]. In this chapter, we focus on this type of deep neural networks.

CNNs usually consist of three different types of layers: Convolutional, Pooling and Fully Connected. Figure 2.3 demonstrates an example of such networks. It is worth noting that popular CNN architectures differ in the number, arrangement, and the connection between these layers. In following sections, these layers are discussed in more details.

Convolutional Layer

Convolutional layers are used to extract features from data. In case of images, these features can range from low level details such as edges to very abstract concepts such as human face.

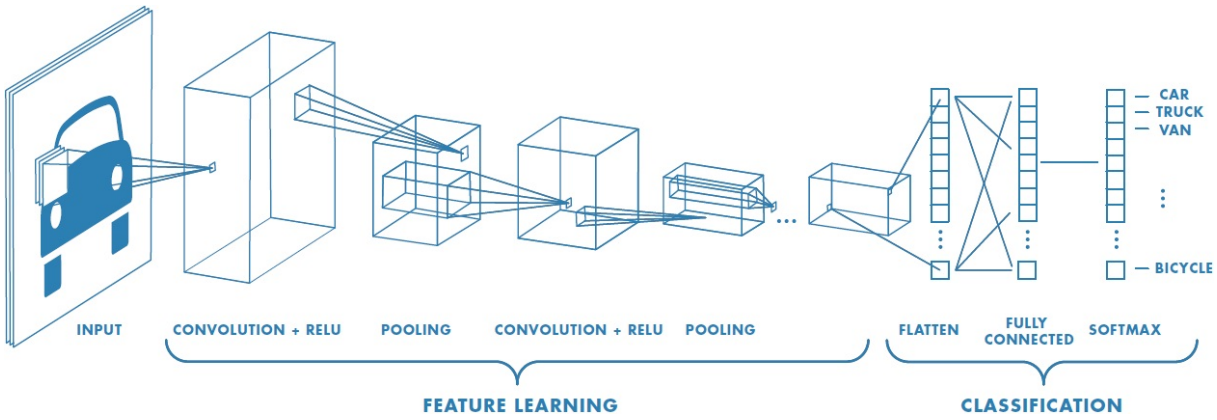

Figure 2.3: Example of a CNN architecture for image classification
Diagram courtesy of The MathWorks, Inc.
https://uk.mathworks.com/discovery/convolutional-neural-network.html

Convolutional layers work based on convolution operation. In each layer, a set of features in form of convolution kernels (also called filters) are applied over the input. The output of these operations is called feature map and specifies the regions in the input that the corresponding feature is present. By stacking convolutional layers in a network, input data is represented in form of feature maps in different levels of abstraction.

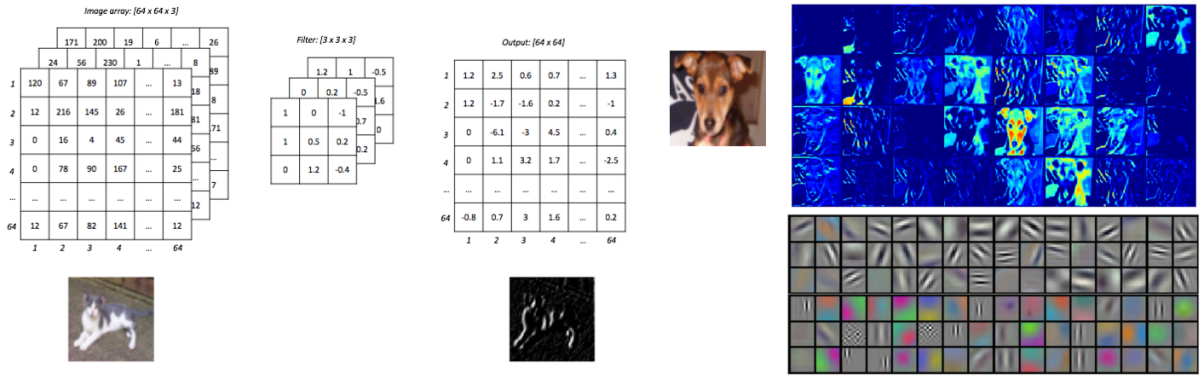

Figure 2.4: Left: Applying a filter (convolution kernel) on input image and the resulting feature map.
Right top: Heat-map visualisation of feature maps corresponding to 32 different filters.
Right bottom: Visualisation of 96 filters of first convolution layer in AlexNet [4]
Left and Right top images are courtesy of
http://www.subsubroutine.com/sub-subroutine/2016/9/30/cats-and-dogs-and-convolutional-neural-networks

In traditional methods of computer vision, the filters for feature extraction were used to be designed by the experts. The main advantage of CNNs is in their ability to automatically learn the suitable filters for each problem in the training process. By learning the best features to represent the data, CNNs get much better results in classification and regression analysis of data in comparison to traditional methods.

To design the architecture of a CNN, for each convolutional layer, value of several parameters are needed to be decided: the number of filters in a layer, the size (height and width) of these filters, strides, and padding. To apply a filter on input matrix, the filter window shifts over the input data; computing a value for each position in the feature map. Strides parameter specifies how many pixels the filter window should move in each dimension over the input to compute a value in the feature map. For example, a stride of (2,2) moves the filter windows two pixels in each dimension, resulting in a smaller feature map than the input. Even with a stride of (1,1),

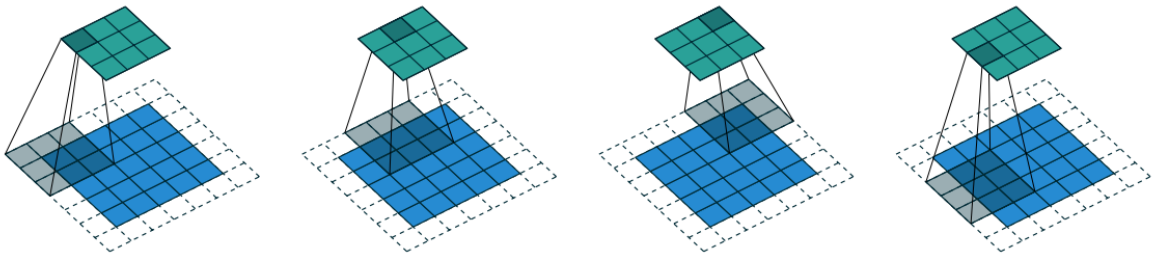

Figure 2.5: Convolution on an input with 5x5 size (Blue), by a kernel size of 3x3 (Gray), in 2x2 strides with zero padding; resulting in a 3x3 feature map (Green) [8].

the size of feature map shrinks. To keep the feature map the same size as input, a zero-padding can be applied on input data.

Activation Layer

It is convention to apply a nonlinear activation layer after each convolutional layer. The main purpose of this layer is to add non-linearity to the network. The activation layer applies a function to all the values of input data from last layer. The most common activation function in CNNs, is the Rectifier: $f(x) = max(0, x)$. In network architecture diagrams, usually the activation layer following a convolutional layer is not depicted.

Pooling Layer

Usually after some convolutional layers in a CNN, a pooling layer is applied. There are several different types of pooling layers such as max pooling, average pooling, and L2-Norm pooling. In this layer a filter (usually of size 2x2) is applied to the input with strides of same length (2x2). In contrast to convolutional layers that compute the weighted sum over the region covered by the filter, the pooling layers compute the max, average or L2-Norm of that region. Pooling (also referred to as down-sampling) layers play two important roles in a network. First, it reduces the size of the data in the network; thus lessening the computational cost. Second, because it preserves the relative position of features in the feature maps but not the exact locations, the network learns to be more general and robust to noise.

Fully Connected Layer

The collection of convolutional, activation, and pooling layers, provides a representation of input data based on the learnt features. Another type of layers is needed to actually perform the classification or regression analysis based on the represented data. The fully connected layers provide a universal function that can learn (by adjusting the weights of their connections) to estimate the relation between variables or predict the class of the input data. Figure 2.6 demonstrates such layers. The fully connected layers work in the same way as traditional neural networks (as described in page 7). It is common to apply the fully connected layers at the end of the CNNs to produce the output of the network. To connect the other layers to the fully connected ones, the multi-dimensional output of those layers is first flattened into a 1D vector, then it is passed to the all input neurons of the fully connected layers (figure 2.3).

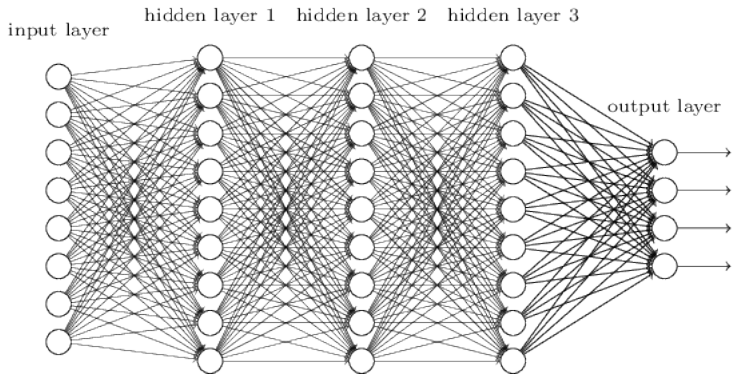

Figure 2.6: Example of fully connected layers of a CNN
Picture courtesy of `http://neuralnetworksanddeeplearning.com/chap6.html`

In classification problems, each output neuron of the fully connected layers estimates the probability of belonging to a specific class (e.g. cat, dog, and so on.). In regression problems, each output neuron estimates the value of a dependent variable. For example in case of this project, the x, y, or z component of a normal vector, corresponding to a specific pixel in the input image.

2.1.3 Training

In CNNs, usually the filters of the convolutional layers and the weights of the fully connected layers are initialised with random numbers. But, to produce a meaningful output, these parameters needs to be adjusted based on the problem. Training is a process in supervised learning to determine the near-optimal values of these parameters. In supervised learning, pairs of input samples and the corresponding desired outputs for a problem are needed. For example, in the problem of age estimation based on the face photo, the pairs of photos of different people and their ages are needed. Based on these (input,desired output) samples, an optimiser can calculate the appropriate values for the parameters of a network. After training, by having a network that its parameters are customised to the problem, it is possible to estimate the output for input samples that are new to the network.

To train a neural network, in addition to an optimiser, a cost function is also needed. A cost function is any function that accepts the computed output of the network and the desired output as arguments, and returns a cost value that is an indicator of error between these values.

Training a network is performed in iterations. In each step, a sample (or a small batch of samples) is chosen from the data set. First, this input is passed to the network and the predicted output of the network is computed. Then, the predicted output and the desired output in the data set is passed to the cost function and the cost value is computed. Finally, the optimiser calculates the adjustments to the parameters of the network based on the cost value. Most of the common optimisers, calculate these adjustments based on the gradient of cost function in respect to all parameters of the network. After each iteration, the parameters of the network are updated.

Usually, to have an idea of the accuracy of a network, the data set is divided into two separate data sets: training and testing. The network is trained on the training data set; then, its estimated output for the samples of the testing data set is computed. The difference between

these values and the desired outputs of the testing data set is an indicator of how good the network is expected to perform for samples out of the training data set.

In the past, the most frequently used optimiser for training the deep neural networks was the Stochastic Gradient Descent (SGD). This optimiser approximates the gradient at each sample (or small batch), then updates the parameters of the network based on this value multiplied by a learning rate. The learning rate is a number usually close but less than one, that determines how fast the parameters of the network should change in training process. Having a single learning rate for all the parameters of the network (while different parameters need different rates) that is fixed during training is one of the main disadvantages of this optimiser.

In this project the Adam optimiser [9] is used for training the network. The Adam optimiser, maintains a learning rate for each network parameter which separately adapts as learning unfolds. Based on the findings of a recent survey [10], Adam is currently the best optimiser for use in projects such as this one.

Likewise, there are many different choices available for the cost function. Some of the most common ones are Mean Squared Error, Mean Absolute Error, and Categorical Cross Entropy. Besides popular cost functions, it is also common to use a custom cost function for a problem.

Finally, the last term relevant to training process that is worth noting is the epoch. An epoch consists of one complete training cycle on the samples in the data set. Normally, more than one epoch is needed to train a network.

2.1.4 Transfer Learning

As mentioned earlier, normally the parameters of a network is initialised by random values. In order to learn the best parameters for the network, a lot of time and a large data set for training is needed. As network architecture gets more sophisticated, the need for larger data sets and more training time increases. Therefore, if the training time or the size of the data set is a constraint, getting acceptable results is less likely. An effective method to solve this problem is the use of the transfer learning. In transfer learning, a pre-trained network (or a part of it) is fine-tuned for use in current problem. For example the lower layers of an image classification network such as VGG-16 [5], can be used for feature extraction in a regression problem.

By initialising the parameters of the network to compatible parameters of another network that is trained on a larger data set and in a relevant problem, it is more likely that the optimal parameters of the network are close to the initial values; thus potentially, less number of training iterations is needed for achieving acceptable outputs.

2.1.5 Data Augmentation

Another technique to compensate for the small training data set is data augmentation. Some data transformations on the data set samples can make considerable changes in the inputs while preserving the desired outputs. For example, moderate changes in contrast or saturation of an image of a dog does not affect the specie of the animal (i.e. the desired output). But, these changes produce different training samples for the network. By using carefully selected data augmentations, the size of a data set increases considerably. Another positive side effect of using data augmentation is decreasing the chance of over-fitting. An over-fitted network losses its generality and robustness to noise.

2.2 Related Work

Before discussing the related work on surface normal estimation from RGB images, it is best to mention a similar problem: estimating the depth map from RGB images. By having the depth map of a scene, it is possible to calculate the pixel-wise surface normals by simply fitting a least-square plane for the neighbouring sets of points in the 3D point cloud (discussed in section 3.3). However, depth map estimation is also a challenging task. In fact, some of the works discussed in this section [11, 12], attempt to jointly estimate depth and surface normal maps. Independent methods for depth map estimation have been proposed, too; methods such as Markov Random Fields [13], focus information and higher order statistics [14], semantic labels [15], deep convolutional neural fields [16], and multi-scale deep neural networks [17].

In recent years, there have been several attempts to tackle the problem of surface normal estimation from RGB images. In this section, some of the better performing works are discussed; interestingly, all of them are based on convolutional neural networks. Although all of these methods use CNNs, the main difference between them lies in their network architecture. They all use the same data sets and metrics for training and evaluation (which are addressed in chapter 3 and chapter 5, respectively). Therefore, the main focus of this section in discussing these methods, is the architecture of underlying CNN models.

Wang et al. [18] propose a network architecture that consists of three sub-networks: a top-down network, a bottom-up network, and a fusion network (figure 2.7). The top-down network takes the RGB image as input and predicts two complementary global interpretations of the scene: a coarse estimation of the surface orientations, and the cuboidal layout of the room. For the bottom-up network, a sliding window on the input image, extracts patches of the image. The bottom-up network takes these patches one after another to generate two types of local image interpretations for the whole image: estimated pixel-wise surface normals, and semantic edge labels (convex, concave, and occlusion).

They also calculate the vanishing points of the input image (based on the non-neural network approach of Hedau et al. [19]) and compute the vanishing point-aligned coarse interpretation of the scene, based on the output of the top-down network. The concatenation of the four outputs of the other sub-networks with the input image and the vanishing point-aligned coarse output, is used as the input to the fusion network. At last, the fusion network estimates the final pixel-wise surface normals, by using a sliding window on the input image and based on the interpretations provided by other sub-networks.

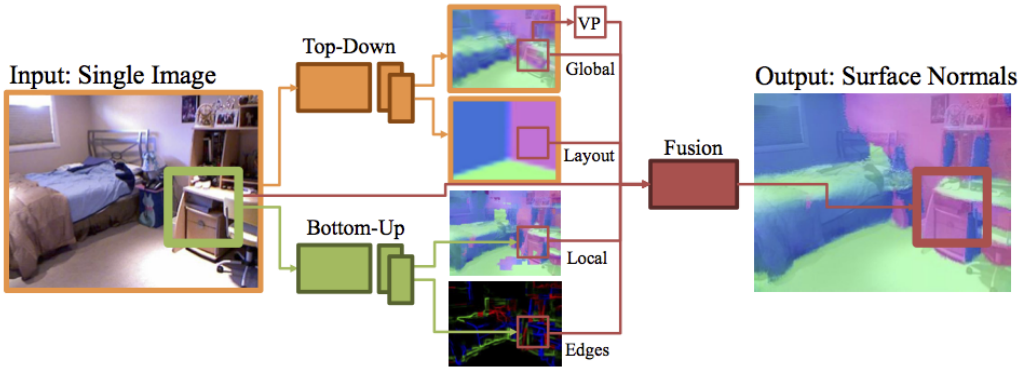

Figure 2.7: An overview of the network architecture designed by Wang et al. [18]

Because of the computational costs and difficulty in training a network that globally estimates the pixel-wise surface normals, Wang et al. use a sliding window to estimate the local surface normals. But, only by looking at local patches of the image, it is difficult to get the results that are consistent within the whole picture. By using a fusion network that considers the coarse global interpretation of the scene, while looking at local patches, they manage to produce 31% better results in comparison to output of the bottom-up network (increase from 27.2% to 39.5% within 11.25° of error).

Moreover, they argue that the design of deep neural networks can benefit greatly by incorporating the insights from past research in 3D scene understanding; notably, knowledge about the room layout and edge labels.

Despite the fact that by adding all the additional interpretations (i.e., room layout, edge labels, and vanishing point-aligned coarse output) to the fusion network, the percentage of the normal vectors within 11.25° error improves by 11.9% (from 39.5% to 44.2%), just adding the vanishing point-aligned coarse output results in 8.1% better outputs (from 39.5% to 42.7%).

Therefore, contrary to the opinion of Wang et al., the small benefits gained by using room layout and edge label interpretations, might not worth the additional complexity that they impose on the network. Furthermore, as recent achievements by convolutional neural networks have shown, focusing on exploiting the feature extraction capabilities of CNNs may be a better choice than using hand-crafted features like room layout or edge labels.

Eigen et al. [11] also estimate the coarse global output and the local surface orientations, but instead of independently computing these outputs and combining them by a separate fusion network, they design a multi-scale deep network (figure 2.8).

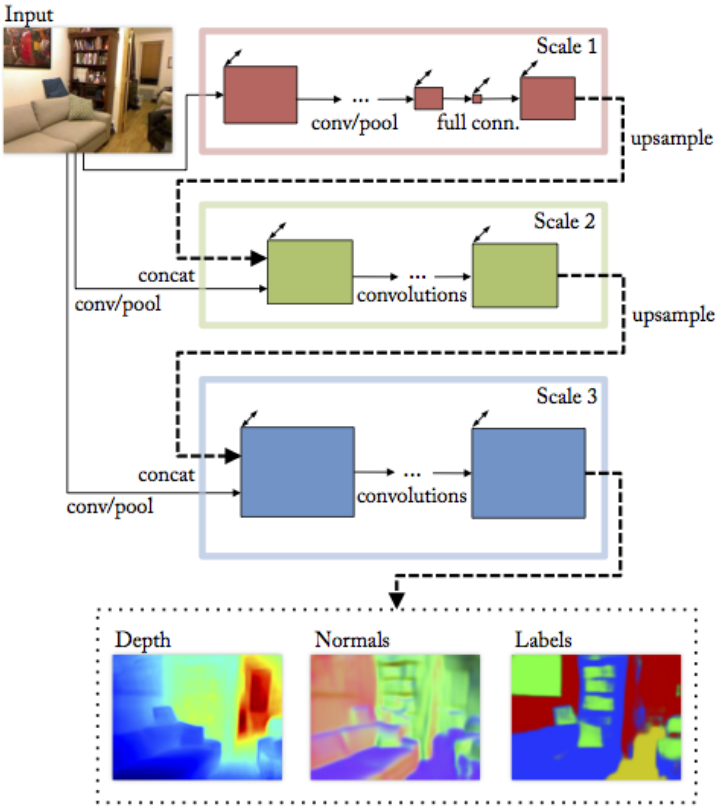

Figure 2.8: An overview of the network architecture designed by Eigen et al. [11]

In their model, first, the input image (of size 320x240) is fed to the Scale 1 network which predicts a coarse global output based on the entire image area. However, instead of directly estimating the coarse surface orientations, this network reshapes the output of the fully connected layers to 64 feature maps of size 19x14. These feature maps are up-sampled by a factor of 4 (i.e., 74x55), and the result is passed to the Scale 2 network.

To produce predictions at a mid-level resolution, a single layer of convolution and pooling with finer strides is applied to the input image and the resulted feature maps are concatenated with the up-sampled feature maps of Scale 1 and are passed to the Scale 2 network. The output of the second scale is a 74x55 prediction of surface orientations which gets up-sampled to size of 147x109.

Finally, by concatenating the up-sampled output of Scale 2 with the feature maps generated by a single layer of convolution and pooling at yet finer stride, the Scale 3 network predicts the surface orientations in more detail (at half resolution of the original image).

Normally in CNNs, the size of the filters in convolutional/pooling layers is relatively small (less than 10x10) in comparison to the input data. Therefore, these layers are suited to extract the local features of the data. In contrast, fully connected layers in a CNN have connections to all the feature maps of the last convolutional/pooling layer of the network. As a result, the fully connected layers have a global view of the input data.

In the network architecture of Eigen et al., the main difference between the Scale 1 and the other scales of the network is in existence of fully connected layers in Scale 1. These layers provide a global view of the scene and enable the Scale 1 network to estimate the coarse global feature maps. The other scales, by using the output of Scale 1 along the feature maps generated from the input image, are able to refine the local details in consistence with the whole scene. It worth noting that the Scale 2 and Scale 3 networks, estimate the pixel-wise normal vectors (x,y,z components), by using 3 filters in their last convolutional layers.

In comparison to work of Eigen et al., Wang et al. use fully connected layers in all of their sub-networks. The local view of their bottom-up and fusion networks, is a result of using a sliding window at the input of these networks. By doing so they can process the input data in higher detail, while keeping the computational costs manageable. In the model of Eigen et al., the resolution of input image to different scales is determined by the length of strides in the first single convolutional/pooling layers in each scale.

For the Scale 1 network of Eigen et al., they try both of AlexNet [4] and VGG-16 [5] network architectures. They initialise the parameters of these networks with parameters of respective models that are trained on ImageNet data set [20].

In training phase, because of the computational costs, first they jointly train both Scale 1 and 2 networks; then, they fix the parameters of these scales and train the Scale 3 network. They show that the more sophisticated model (VGG-16) achieves 13.3% better result (from 39.2% to 44.4% within 11.25° error) in comparison to the other one.

Dharmasiri et al. [12] design a network architecture that is fundamentally very similar to work of Eigen et al. They extend the model of Eigen et al., by adding parallel layers in Scale 2 and 3 of the original network (figure 2.9). These parallel layers are adapted to estimate the pixel-wise depth and surface curvature, based on the input image. The network is configured to jointly estimate the depth, surface normals, and surface curvature.

They argue that by using multi-task learning, each task benefits from having access to the data representation provided by other similar tasks. For example, the normal estimation layers in Scale 3, receive the concatenated outputs of all the layers in Scale 2; thus, theoretically they have access to more extracted features about the input image. This idea is similar to work of Wang et al. in terms of combining different types of extracted features to enhance the performance of the network. Likewise, the enhancements gained by adding these additional representations to the network (from 43.6% to 44.9% within 11.25° error), may not worth the complexity that they introduce to the network.

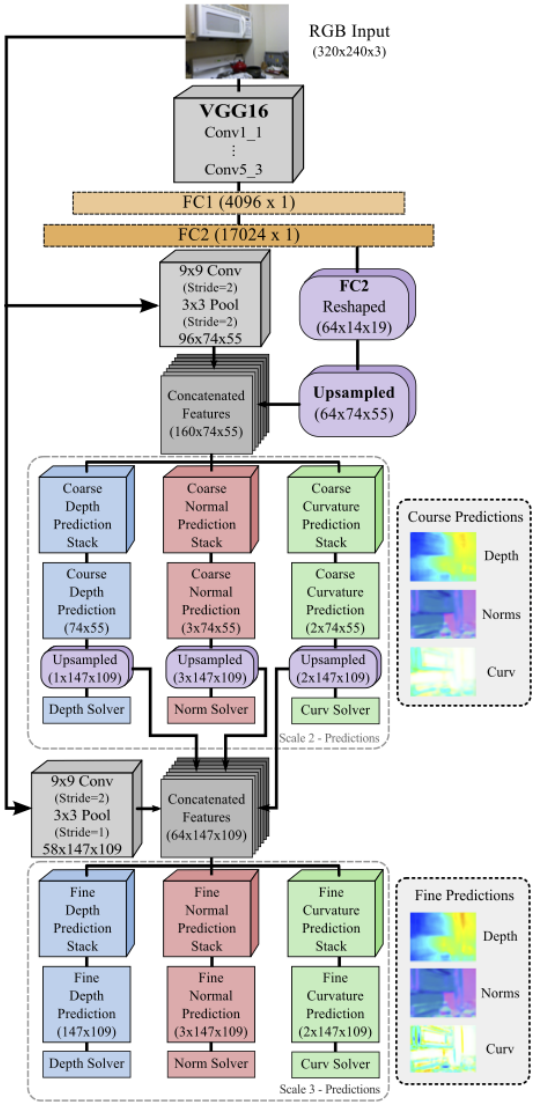

Figure 2.9: An overview of the network architecture designed by Dharmasiri et al. [12]

Finally, Bansal et al. [21] propose a novel network architecture, based on hypercolumn representation, that achieves state of the art results in task of surface normal estimation.

In conventional CNNs, the fully connected layers receive their input only from the last convolutional/pooling layer of the network, which extracts the features with highest level of abstraction. This high level feature representation of input data is exactly what is needed for classification tasks such as classifying the furniture in a room ; after all, the overall shape of an object is a more relevant factor than low level details such as pose or illumination. But in finer grade tasks such as normal estimation, these low level details are exactly what matters most. Therefore, in such applications, the last convolutional/pooling layer is not the optimal representation.

Based on this reasoning, Hariharan et al. [22] introduce hypercolumn representation in CNNs. They define the hypercolumn at a pixel as a feature vector formed by concatenating the convolutional responses of a CNN, corresponding to the location of that pixel. In other words, in each feature map in a convolutional layer, the value corresponding to the location of the input pixel is identified. All these values across the layers of the network, are concatenated to form a hypercolumn feature vector. In contrast to the feature maps of the last convolutional/pooling layer, a hypercolumn feature vector can capture coarse, mid, and fine-level details.

Bansal et al. [21] use the convolutional/pooling layers of VGG-16 [5] architecture (figure 2.11) in their network. On top of VGG-16 layers, two additional convolutional layers are applied. The hypercolumn features are extracted from the $1_2, 2_2, 3_3, 4_3, 5_3$ layers of the VGG-16 model along with the last additional convolutional layer. The fully connected layers accept the hypercolumn feature vector of a pixel as input, and predict the surface normal vector for that pixel (figure 2.10).

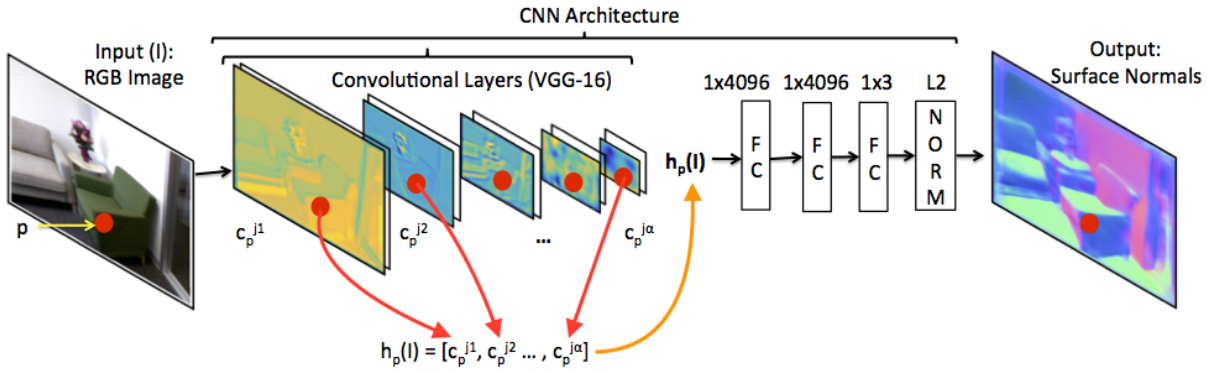

Figure 2.10: An overview of the network architecture designed by Bansal et al. [21]

In training, instead of using all the pixels of an image, 1000 pixels per image are randomly sampled and the cost function calculates the error between the predicted and the desired normal vectors for these pixels. By doing so, the memory usage remains in bound and also the chance of over-fitting is reduced.

Although at first glance, it may look that by randomly sampling the pixels, the spatial information is lost, but in practice the hypercolumn vectors contain high level features from the upper layers of the network.

In fact, they show that the features of the hypercolumn vector should be selected from a combination of low, mid, and high parts of the network; otherwise, the performance of the network suffers considerably. For example, using only low and mid parts of the network ($1_1, 1_2, 3_3$ layers) degrades the performance by 45% (from 42.0% to 23.1% within 11.25° error) in comparison to using low, mid, and high parts ($1_2, 3_3, 5_3$ layers).

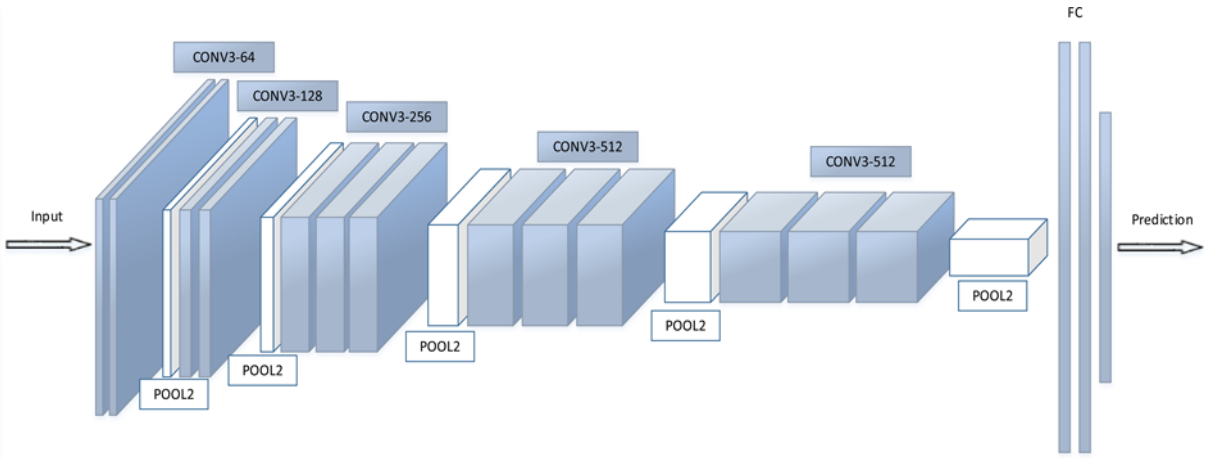

Figure 2.11: An overview of the VGG-16 [5] network architecture.
Diagram courtesy of El Khiyari et al. [23]

To summarise, recent work in surface normal estimation from RGB images can roughly categorised into two groups. On the one hand, Wang et al. [18] and Dharmasiri et al. [12] combine features extracted by task-specific sub-networks (room layout, edge labels, depth, curvature and so on) to enrich and diversify the data representations. Although these sub-networks add complexity and computational costs to the system, their results are only a few percent better than the simpler networks. On the other hand, Eigen et al. [11] and Bansal et al. [21] solely use the features that are learnt by the network layers. While in network architecture of Eigen et al.[11], the focus is on the features that are learnt at the last layer of each of the three scales, in work of Bansal et al. the features are extracted from six different layers. On the whole, considering the relative simplicity and better results of these architectures, leaving the task of feature extraction to the CNNs and avoiding the use of task-specific representations looks as a better choice. Also, based on the findings of Bansal et al.[21], using more representations at various levels of abstraction, may have a positive effect on the final results.

Chapter 3

Data Set

3.1 Overview

In success of any software system that is based on deep neural networks, two main factors play a crucial role: network architecture and training data set. Although, a good network design is necessary for getting acceptable results; without an appropriate data set, training the network and getting any result is not possible. In fact, not only the quality of data set matters, but also as Sun et al. [24] show, the performance of deep neural networks in vision tasks increases logarithmically based on volume of training data.

For the task of surface normal estimation from RGB images of indoor scenes, a data set containing the RGB images and the corresponding surface normal maps is needed. As mentioned in related work section (page 13), it is possible to indirectly calculate the needed surface normal maps by using the depth map of the images. Therefore, to acquire such data set, a collection of images and their corresponding depth maps should be obtained; either by directly taking various pictures of indoor scenes or by using publicly available RGB-Depth data sets.

Capturing new images is not only time consuming, but also makes it difficult to compare the results of the experiments with previously published results. Therefore, in this project, a publicly available RGB-D data set (NYU Depth V.2 [25]) is used. All related work discussed in section 2.2, use this data set for training their models. Also, to explore the effect of using a higher quality data set on improving network performance, a more recently released data set (SUN RGB-D [1]) is used in this project.

There are several methods for computing the surface normal maps based on the depth map of the scene. For consistency with past research in this field and producing comparable results, this project used surface normal maps computed by using two more popular methods of Ladicky et al. [26] and Silberman et al. [25]. These methods and the aforementioned data sets are discussed in next sections.

3.2 Publicly Available RGB-D Data Sets

There are many existing RGB-D data sets available; but, only a few of them are suitable for the purpose of this project. For example, some data sets (such as [27] and [28]) use a turntable for capturing the image of objects, instead of using real-world scenes. Even in data sets that use real-world indoor scenes, some of them (such as Berkeley 3-D Object Dataset [29]) contain unrealistic scene layouts (e.g. snapshot of a computer mouse on the floor) or have a small size. In the following sections, two of the data sets that are more suited for use in this project are briefly introduced.

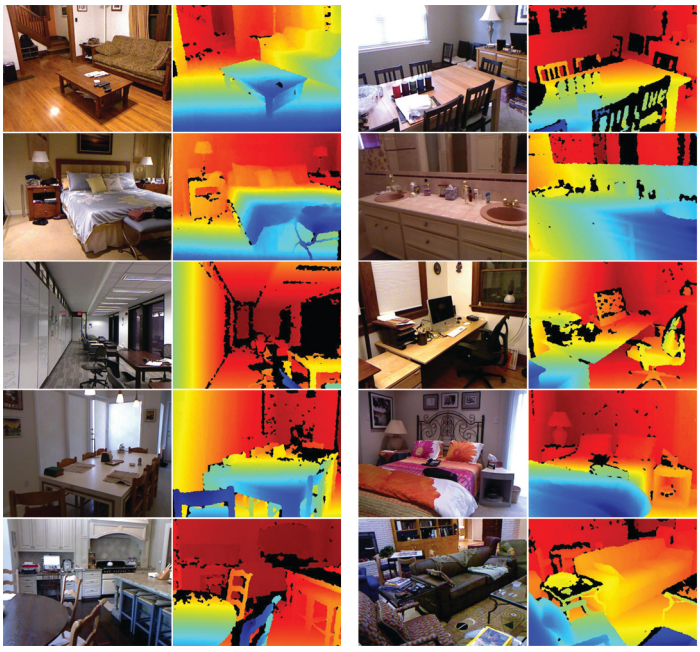

Figure 3.1: NYU Depth V.2 data set, samples of the RGB images and the corresponding depth maps (missing data is coloured in black)[25]

3.2.1 NYU Depth V.2

This data set has been the most popular data set for the task of surface normal estimation of indoor scenes. It is comprised of 1,449 pairs of aligned RGB and depth images, gathered from a wide range of commercial and residential buildings in three different US cities. These images are extracted from short video sequences captured by Microsoft Kinect version 1, from 464 indoor scenes across 26 scene classes.

The data set has three components: the raw data (rgb, depth and accelerometer) as provided by the Kinect, preprocessed RGB and depth images, and a toolbox for data manipulation. In preprocessed part of the data set, in addition to the projected depth maps, a set of preprocessed depth maps whose missing values have been filled in (using the colourisation scheme of Levin et al. [30]) is also included. This data set also provides meta-data about each image; which is irrelevant to this project (such as dense per-pixel labeling for each image). Figure 3.1 illustrates a few samples of this data set.

Silberman et al. also suggest a train/test split for the samples of the data set. In this project, this split was used for selecting the samples for training and evaluation processes. The preprocessed samples and the toolbox are provided in format of MathWorks MATLAB files.

3.2.2 SUN RGB-D

Princeton SUN RGB-D data set [1], is a super-set of the NYU Depth V.2 [25] data set. To construct this data set, Song et al. [1] capture 3,784 images using Microsoft Kinect v2 and 1,159 images using Intel RealSense sensors. They also include the 1,449 images from the NYU Depth V.2 data set and manually select 554 realistic scene images from the Berkeley B3DO Dataset [29]. Finally, by including the manually selected 3,389 distinguished frames from the SUN3D [31] videos captured by Asus Xtion sensor, the total number of images in this data set reaches 10,335.

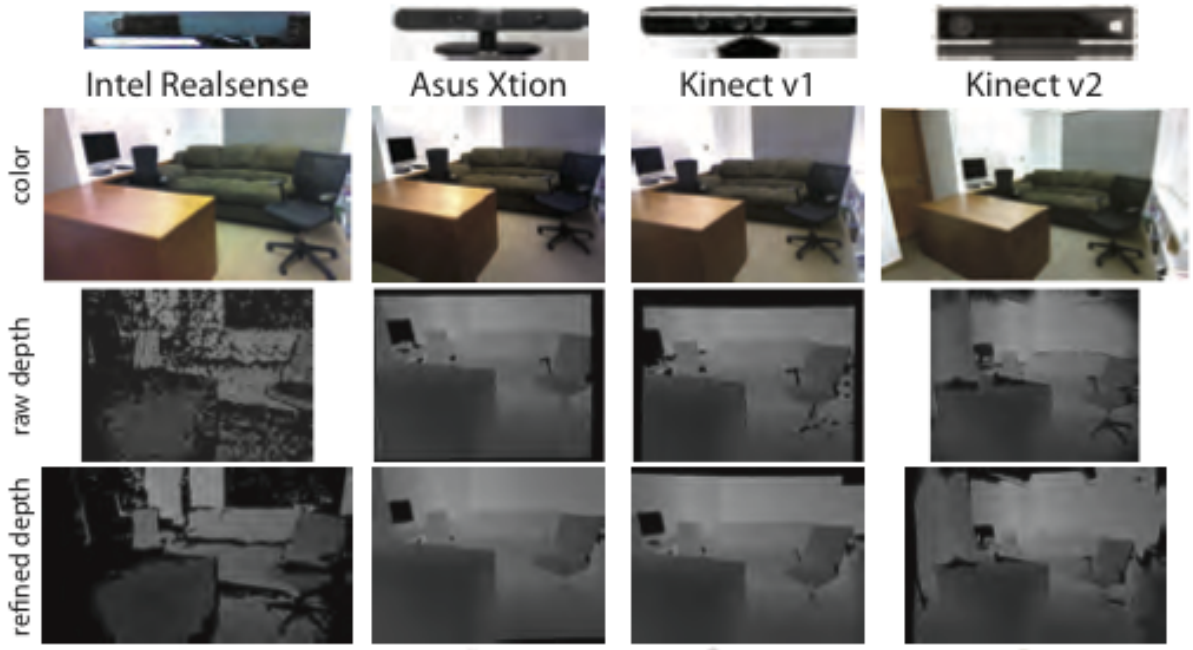

Figure 3.2: Comparison of the four RGB-D sensors and the refinements in depth maps achieved by using the algorithm of Song et al. [1, p. 2]

These RGB-D images are extracted from short videos captured from indoor scenes (such as universities, houses, and furniture stores) in North America and Asia. Like NYU Depth V.2, the raw depth maps captured by the depth sensors in this data set is not perfect; due to measurement noise, reflection of surfaces like mirrors, and occlusion boundaries.

Also, the resolution of RGB images and the quality of raw depth maps captured by different sensors is not equal. Intel RealSense outputs noisier depth maps with more missing values, while Asus Xtion and Kinect v.1's depth maps have observable quantisation effects. Kinect v.2 captures depth maps with more details, but it is more sensitive to reflection and dark colours. Thus, to improve these depth maps, Song et al. propose an algorithm that uses depth data captured during multiple frames of the video, to obtain a single refined depth map for each image (Figure 3.2).

Alongside this data set, a toolbox (MathWorks MATLAB files) for loading and visualisation of data is also provided. The RGB images and depth maps are presented as PNG files.

3.3 Surface Normal Maps

As mentioned earlier, several different methods are used for computing the surface normal maps from depth data. In all related work discussed in chapter 2, the published results are evaluated based on the surface normal maps computed by Ladicky et al. [26] on the testing data set. For training the network, the raw data from the training data set is used. Because Ladicky et al. have not published the surface normal maps for the raw data of the NYU Depth V.2 data set [25], the method of Silberman et al. [25] is used for computing these normal maps. Dharmasiri et al. [12] also additionally evaluated their result based on the surface normal maps computed by the method of Spek et al. [32].

In this project, the method of Silberman et al. [25] was used for computing the surface normal maps in SUN RGB-D data set [1]. To get comparable results, surface normal maps provided by Ladicky et al. [26] were used in evaluating the output of the networks.

In method used by Silberman et al. [25], first by using the intrinsic parameters of the camera, the depth points are projected from image plane to the 3D world coordinates. Then, for the neighbouring sets of points in the 3D point cloud that their distance is not too large, a least-square plane is fitted. The normal vectors of these planes are the computed surface normals corresponding to the depth map points.

On the other hand, Ladicky et al. [26] first apply a denoising technique (based on the second order Total Generalized Variation (TGV) [33]) on depth data. Then, normals are computed on the 3D point cloud for each point in a local 3D spatial neighbourhood. Finally, to estimate the point-wise normals, a least squares regression kernel in a RANSAC scheme is utilised in order to preserve surface edges [26, p. 10].

The difference between these methods is due primarily to the fact that Ladicky et al. [26] uses a more aggressive smoothing which results in flatter areas, while Silberman et al. [25] produces noisier results but with more details present [11, p. 5]. Figure 3.3 illustrates the difference in output of these methods.

Figure 3.3: Comparison of normal maps computed by different methods
Top: based on Ladicky et al. [26] , Bottom: based on Silberman et al. [25]

Figure 3.4: Top: normal maps computed based on the method of Silberman et al. [25]
Bottom: masked normal maps after denoising (invalid values are coloured in gray).

To reduce the noise presented in the output of Silberman et al.'s method [25], inspired by the work of Wang et al., a denoising technique (total variation regularised least-squares deconvolution [34]) was applied on the computed normal maps in this project. Figure 3.4 illustrates the results obtained by applying this method.

3.4 Implementation

Three different data sets were created for use in this project:

- A main data set based on the RGB images and depth maps of the NYU Depth V.2 [25], and surface normal maps provided by Ladicky et al. [26].
- An alternative data set based on the NYU Depth V.2, and surface normal maps computed based on the method of Silberman et al. [25]
- A data set based on the RGB images and depth maps of the SUN RGB-D [1], and surface normal maps computed based on the method of Silberman et al. [25]

MathWorks MATLAB was used for developing the scripts and functions required for creating the data sets of this project. To build the main data set, first the RGB images and raw depth maps were extracted from NYU Depth V.2 data set. The raw depth data was used to create a mask of invalid data points in each sample (i.e. pixels with missing or invalid depth values). Then, the surface normal maps provided by Ladicky et al. were loaded and decoded. These normal maps are provided as PNG images. Because, it was not possible to find any information about the encoding used in the process of converting the original normal maps to these images, the histograms of each channel in several samples were manually analysed to develop a decoding function. Finally, the RGB images and corresponding masked normal maps were stored in a MAT file as output.

To create the alternative data set, similar to main data set, the RGB images and the masks of invalid data points were extracted. The surface normal maps were computed and a denoising technique was applied as discussed in section 3.3. The masked surface normal maps and RGB images were stored in a MAT file as output.

As mentioned before, the SUN RGB-D data set consists of RGB images and refined depth images in PNG format. To explore the effect of using higher quality samples in this project, only the subset of SUN RGB-D data set is used that has corresponding images in NYU Depth V.2 data set.

A meta-data data set and a toolbox are also provided for loading and manipulation of data. To create the last data set, the full resolution RGB and raw depth images were loaded by using the information provided in meta-data data set. Similar to the data provided by Ladicky et al., a custom encoding was used for storing the original data in PNG format. Surface normal maps were computed based on the raw depth data and denoising applied on the results. Finally, the RGB images and corresponding masked normal maps were stored in a MAT file.

Figure 3.5 illustrates the difference in quality of samples obtained from different data sets. In general, samples based on the data obtained from SUN RGB-D have less invalid values (coloured in gray) and slightly less noise.

The instructions for obtaining the data sets and source codes are provided in appendix A. During the development, to ensure the validity and quality of the outputs, code snippets were interactively tested and the output variables were inspected after each modification in their values.

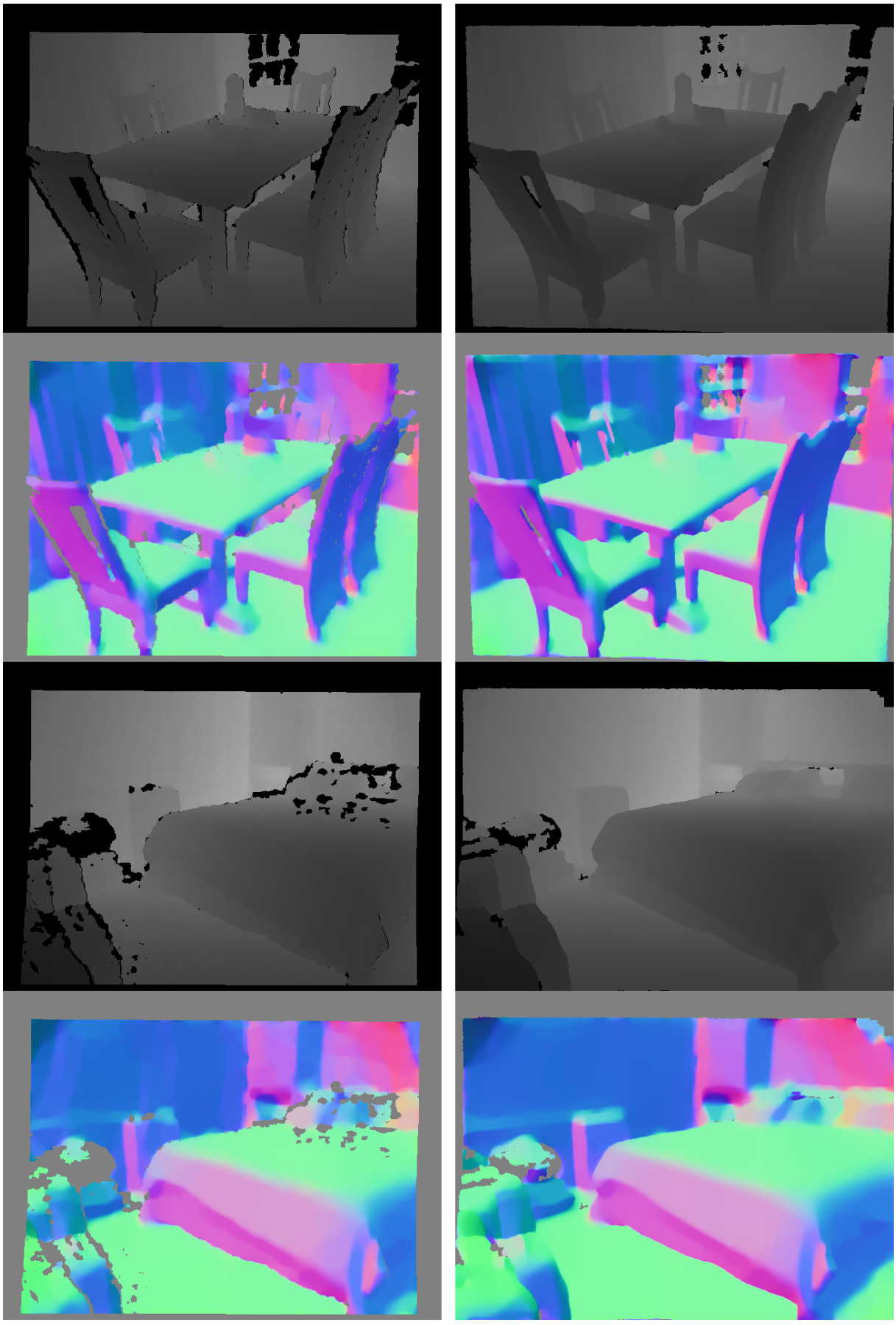

Figure 3.5: Left: depth and surface normal maps based on NYU Depth V.2
Right: depth and surface normal maps based on SUN RGB-D

Chapter 4

Model Architecture

Arguably, the most important aspect of a system based on deep neural networks, is the network architecture. Year over year improvement in performance of DNNs in field of computer vision in recent years, was possible by using deeper and more sophisticated models. Figure 4.1 illustrates the number of layers in DNNs that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in recent years.

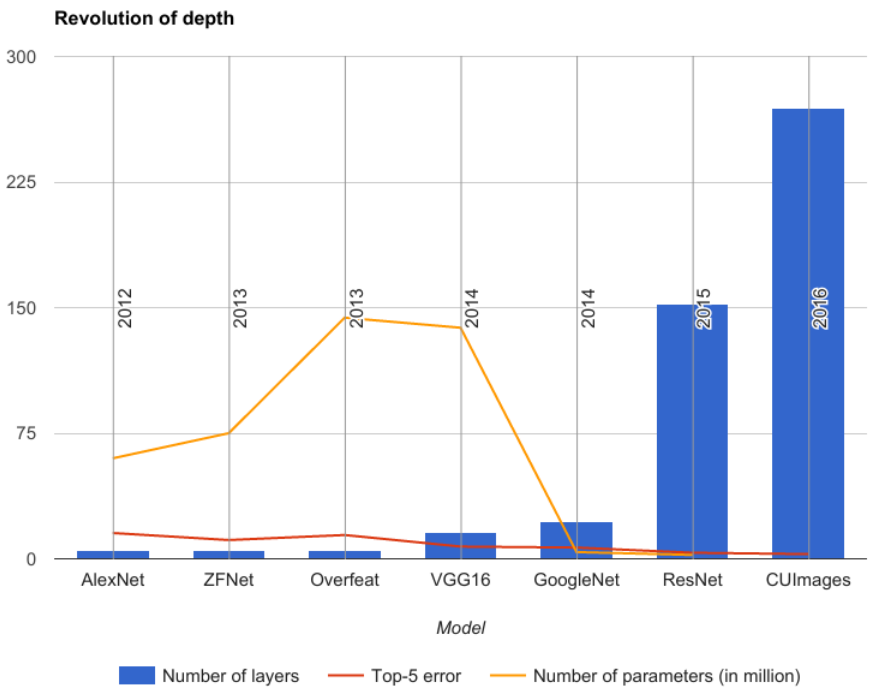

Figure 4.1: Evolution of depth, error-rate, and number of parameters in best-performing DNNs in ILSVRC Challenge over the years
Diagram courtesy of https://medium.com/@pierre.ecarlat

In this project three main types of network architectures were implemented: Baseline, VGG16, and ResNet50. Afterwards, various modifications to the architecture of these networks were investigated. The outcome of each experiment was used to guide the process of designing new modified versions of the network. The architecture of these networks are discussed in next sections.

4.1 Baseline

Based on the top-down sub-network of Wang et al. [18], a seven layer convolutional neural network was designed as the baseline model. Table 4.1 illustrates the architecture of this network.

Layer Type	Number of Filters/Units	Size	Stride	Output Shape
Input				240x320x3
Convolution	64	5x5	1x1	240x320x64
Max Pooling		2x2	2x2	120x160x64
Convolution	192	3x3	1x1	120x160x192
Max Pooling		2x2	2x2	60x80x192
Convolution	384	3x3	1x1	60x80x384
Convolution	256	3x3	1x1	60x80x256
Convolution	3	3x3	1x1	60x80x3
Flatten				14400
Fully-Connected	4096			4096
Fully-Connected	14400			14400
Reshape				60x80x3
Resize				240x320x3
Normalise				240x320x3

Table 4.1: Baseline network architecture

The input image passes through five convolution and two pooling layers. The output is flattened to a vector with size of 14400. Then, two fully connected layers are applied. The output of fully connected layers is reshaped to a tensor with size of 60x80x3. A bi-linear resizing layer, upsamples the output of the network to size of 240x320x3. Finally, the estimated values for the normal map are normalised by last layer of the network.

Based on this architecture, several modified versions of the network were designed and the effect of these modifications on the results was investigated. These modifications are discussed as follows:

4.1.1 Number of Convolution Filters

Number of filters in convolutional layers is a parameter that affects the number of features extracted from input data. One can hypothesise that by estimating the normal maps based on more features, the performance of the network might increases.

To investigate the effect of this parameter on output, the number of filters in last convolutional layer was increased from three to the maximum possible value. The other parameters in the network was kept as before. This decision was made because not only increasing the number of filters in other convolutional layers was not possible due to memory constraints, but more importantly, the main bottleneck in number of extracted features accessible to fully connected layers for normal estimation was the last convolutional layer.

Unfortunately, because of memory constraints (12GB GPU Memory), having more than twelve filters at last convolutional layer was not possible. As mentioned before, fully connected layers have weighted connections to all feature maps of the last convolutional layer. Therefore, increase in number of these filters largely affects the total number of weights in the network and consequently memory usage.

4.1.2 Size of Convolution Filters

Another parameter that affects the extracted feature maps of the network, is the size of the convolution kernels. Larger kernels compute the value of feature maps at each location, based on

a larger spatial neighbourhood. Estimating the normals by considering a bigger spatial region might produce more consistent outputs in each local neighbourhood and less noise in result. Although, there is a higher chance that local out-liners (e.g. at edge of an object) affect the neighbouring regions.

In this model, the kernel size of the last convolutional layer in baseline architecture, was increased from 3x3 to 15x15. All other parameters were kept same as baseline model except the size of the padding at last convolutional layer. By using a larger padding, the shape of the output was kept the same size as baseline model (i.e. 60x80x3).

4.1.3 Average Pooling

As mention in section 2.1.2, one of the main reasons of using a pooling layer is to keep the memory usage in bound. The max pooling layers achieve this task by choosing the maximum values of the feature maps under their filter window and discarding the rest of data. Therefore, always the feature with strongest response dominates the result.

Alternatively, using an average pooling layer and considering the weaker features, might result in including more details in representation of input data. Based on this speculation, all the max pooling layers in baseline architecture were replaced with average pooling layers in a new model. Because this change does not affect the output shape of the layer, all other parameters of model remained intact.

4.1.4 Structure of Fully Connected Layers

Another component of the network is the fully connected layers. These layers play the important role of estimating surface normals based on the feature maps provided by convolutional layers. Therefore, to have better outputs, it is important to have fully connected layers with carefully designed structure.

The baseline model has two fully connected layers and thus there are two ways to modify its structure. Two separate models were designed; each with the goal of maximising the number of neurons in one of the fully connected layers. The models were executed several times to find the maximum number of neurons that fit in the memory.

In one model, the number of neurons in last fully connected layer was increased from 14,400 to 57,600 neurons. By having a bigger outer fully connected layer, the resolution of output normal maps increases. Higher resolution output might increase the accuracy of results in a very good performing model, but it was expected to result in worse performance in our baseline model; because the model needs to produce more estimated values based on limited input from previous layer. Also, the resize layer in the network was accordingly modified to keep the overall output shape of the network the same as baseline model.

In the alternative model, the number of neurons in first fully connected layer was increased from 4,096 to 14,400. Hypothetically, a bigger inner fully connected layer can pass more information from convolutional layers to other fully connected layers. By doing so, it might be possible that this extra information improve the output of the network. Because of memory constraints, the last fully connected layer could not be bigger than 6,480 neurons. While this problem affects the comparison of this model to the baseline model, the results are still comparable to the other model.

4.1.5 Up-Sampling

The up-sampling layer in baseline model, increases the resolution of the output by applying the bi-linear up-sampling method. Bi-linear up-sampling is a simple method that produces acceptable results and is used in other related work discussed in section 2.2. There are also other up-sampling methods such as nearest neighbour, and cubic spline interpolation available for use in this layer.

To explore the effect of using a different up-sampling method on final results, a model with simpler up-sampling layer was implemented. In this layer up-sampling was performed by simply repeating the values. The motivation for using a simpler up-sampling method is that the other layers of the network do not need to learn how to deal with more complex transformations performed by bi-linear up-sampling layer.

4.1.6 Replacing a Fully Connected Layer with Convolutional Layer

In some applications in the field of computer vision, the fully convolutional neural networks have achieved considerable results. These networks only rely on use of convolutional/pooling layers and do not have any fully connected layers. Based on this idea, a modified network architecture was implemented.

In this model, the first fully connected layer was removed and the size of the filters in last convolutional layer was changed from 3x3 to 1x1. By doing so, the last convolutional layer acts on each individual value on the feature maps.

4.2 VGG16

VGG16 [5], is a very popular network architecture in many computer vision tasks. Therefore, this network was used as a basis for another group of network architectures that were explored for the task of surface normal estimation. First, the original network architecture was adapted for use in this project. Table 4.2 illustrates this architecture.

The parameters of the network were initialised based on a VGG16 model trained on ImageNet data set. Each block of this network extracts features with a different level of abstraction. To investigate the effect of using different levels of data representation abstraction in output of the network, three different models were implemented. In addition to these models, a network architecture that concatenates the different levels of abstraction was also designed. These models are discussed as follows.

4.2.1 Models Based on the Low, Mid, and High Parts of the VGG16

The first model was designed based on the output from the lower blocks of the VGG16 architecture. In this model only the block 1 and block 2 of the original VGG16 network was implemented. After the last layer of the block 2, a convolutional layer with 12 filters (size of 3x3) was applied and the output was flattened and passed to the fully connected layers. Considering the relatively large size of the feature maps in block 2, it was not feasible to include more than 12 filters in the last convolutional layer due to memory constraints.

Block No.	Layer Type	Filters/Units	Size	Stride	Output Shape
	Input				240x320x3
Block 1	Convolution	64	3x3	1x1	240x320x64
	Convolution	64	3x3	1x1	240x320x64
	Max Pooling		2x2	2x2	120x160x64
Block 2	Convolution	128	3x3	1x1	120x160x128
	Convolution	128	3x3	1x1	120x160x128
	Max Pooling		2x2	2x2	60x80x128
Block 3	Convolution	256	3x3	1x1	60x80x256
	Convolution	256	3x3	1x1	60x80x256
	Convolution	256	3x3	1x1	60x80x256
	Max Pooling		2x2	2x2	30x40x256
Block 4	Convolution	512	3x3	1x1	30x40x512
	Convolution	512	3x3	1x1	30x40x512
	Convolution	512	3x3	1x1	30x40x512
	Max Pooling		2x2	2x2	15x20x512
Block 5	Convolution	512	3x3	1x1	15x20x512
	Convolution	512	3x3	1x1	15x20x512
	Convolution	512	3x3	1x1	15x20x512
	Max Pooling		2x2	2x2	7x10x512
	Flatten				35840
	Fully-Connected	4096			4096
	Fully-Connected	14400			14400
	Reshape				60x80x3
	Resize				240x320x3
	Normalise				240x320x3

Table 4.2: Adapted VGG16 network architecture

The next model also included the block 3 layers. Likewise, a convolutional layer connected the output of the last pooling layer to the flattening layer. This time, 48 filters in the last concolutional layer was the maximum number of filters that fit in memory.

Finally, the last model implemented the blocks 1 to 4 of the VGG16 architecture. The additional convolutional layer had 192 filters with size of 3x3. The outputs of these models were compared to the results of the adapted VGG16 model.

4.2.2 Concatenating the Low, Mid, and High Parts of the VGG16

Inspired by the work of Bansal et al. [21], a new network architecture based on VGG16 was designed. This model included all the convolutional/pooling layers of the original VGG16 network; but, instead of connecting the last pooling layer of the network to the flattening layer, the feature maps from the last convolutional layers of all blocks were concatenated and passed to an additional convolutional layer with 12 filters. Figure 4.2 illustrates the architecture of this network.

Hypothetically, by using feature maps with different levels of abstraction, the fully connected layers have access to a much richer data representation. Therefore, a better performance might be achievable. A problem with using the output of different layers is the difference between the size of feature maps. To solve this problem, additional pooling layers were applied on the outputs of block 1 and 2, and the outputs of block 4 and 5 were up-sampled to size of 60x80x3.

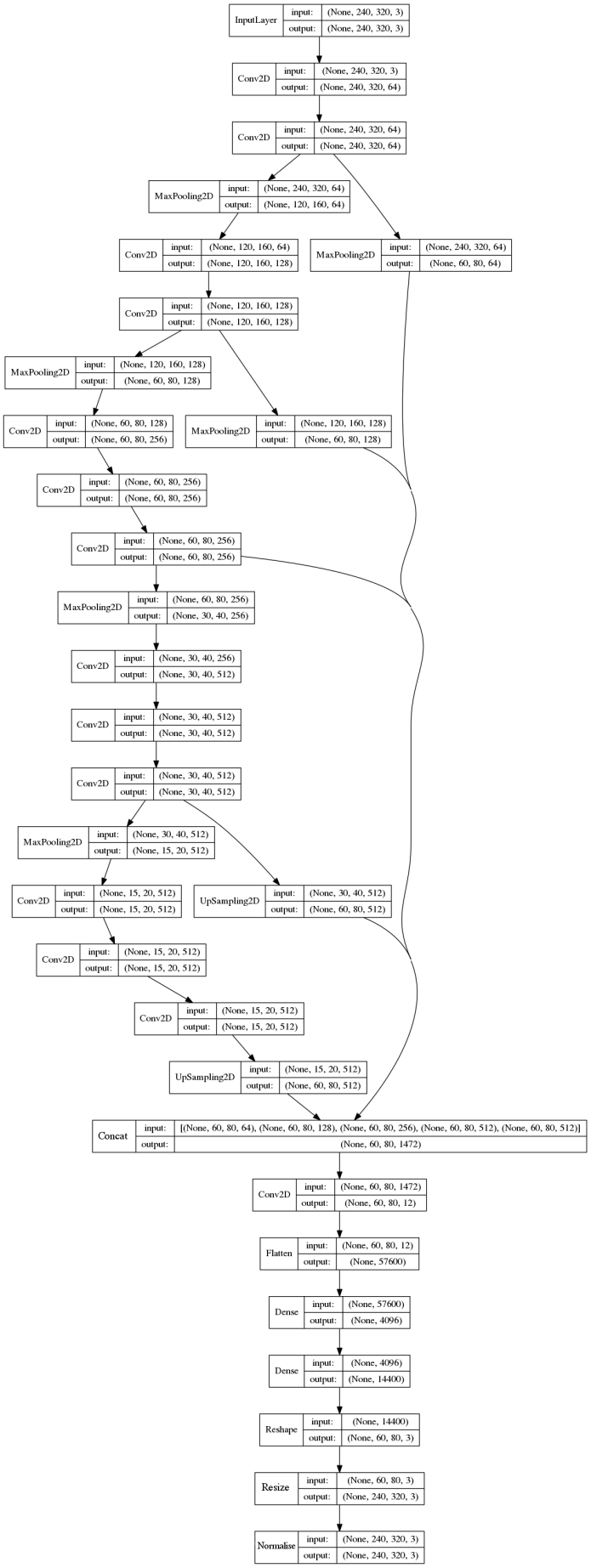

Figure 4.2: Network architecture based on the concatenation of feature maps from different layers

4.3 ResNet50

ResNet [35] is another popular deep neural network in computer vision tasks over the last few years. This network is based on the concept of Residual blocks. The general structure of a residual block is illustrated in figure 4.3. The right path in the block is referred to as shortcut. By using this structure, training very deep neural networks is possible. Using several variations of residual blocks in a network is a common practice.

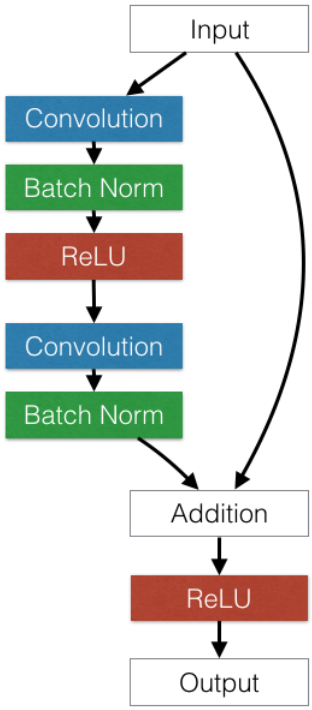

Figure 4.3: A residual block in ResNet network architecture
caplicenseDiagram courtesy of
`https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/`

In this project a ResNet-based network architecture with 50 layers was adapted to the task of surface normal estimation. Two variations of residual blocks were used:

- *Identity Residual Block*: three convolutional layers and a shortcut path
- *Convolutional Residual Block*: three convolutional layers and a convolutional layer on the shortcut path

As illustrated in table 4.3, four convolutional and twelve identity residual blocks were used in the architecture of this network. For the last pooling layer, an average pooling was chosen. Finally, the same structure as the baseline model was used for the upper layers of the network.

Given enough training samples, a complex network such as ResNet50, could potentially achieve good results. The performance of this network was explored in this project.

Type of Component
Input Layer
Convolutional Layer
Max Pooling Layer
Convolutional Residual Block
Identity Residual Block
Identity Residual Block
Convolutional Residual Block
Identity Residual Block
Identity Residual Block
Identity Residual Block
Convolutional Residual Block
Identity Residual Block
Identity Residual Block
Identity Residual Block
Identity Residual Block
Identity Residual Block
Convolutional Residual Block
Identity Residual Block
Identity Residual Block
Average Pooling Layer
Flattening Layer
Fully-Connected Layer
Fully-Connected Layer
Reshaping Layer
Resizing Layer
Normalisation Layer

Table 4.3: Adapted ResNet50 network architecture

Chapter 5

Experiments and Results

5.1 Overview

A conventional pipeline of any deep learning system consists of three main phases: training, prediction, and evaluation. For ease of use and providing more flexibility, a configurable pipeline for this project was implemented. This pipeline was developed in python programming language and is based on the TensorFlow machine learning library. Each experiment in this project, was described in a configuration file. The configuration file specifies the data set, network model, and training parameters used for an experiment.

New network models and data set drivers can easily be added to the project in form of separate python source code files. By specifying the name of this new models or data sets in a configuration file, the experiment is ready for start; no further changes to the pipeline is required.

A Linux shell script was used for submitting the experiments to the Advanced Research Computing (ARC3) HPC service provided in University of Leeds. The submitted jobs were executed on Nvidia Tesla P100 GPUs. The time spent for running the experiments varied between 90 minutes to 10 hours per experiment.

For consistency between results, each model was trained for 100 epochs on the training data set in batches with size of 32 or 16 samples. Most experiments were repeated at least two times to double check the results and the evaluation was performed on a separate test data set. The trained model, predicted normal maps, and evaluation results were stored for each experiment.

For data augmentation, a per-batch random scaling by factor of 1 or 1.5, random cropping, and random horizontal flipping was performed on the data set samples and the results were resized to the input size of the network. Also, random hue and saturation modifications were applied on images of each batch.

For consistency with previous research, the loss function used in these experiments is the average of dot products between estimated and desired normal maps over pixels with valid depth data.

To ensure the quality of software, the source code was developed in interactive Project Jupyter notebooks. Each snippet of code was manually tested and the validity of input and output variable were investigated. The verified changes were committed to a Git repository accessible on GitHub website. On HPC servers, each new model was executed in an interactive session and after confirming the model, the experiments based on that model were sent to scheduler for batch execution.

The evaluation metrics and the results of experiments are discussed in following sections.

5.2 Metrics

The metrics used for evaluation of the outputs in this project, are the same metrics used in related work discussed in section 2.2. As mentioned before, by doing so, it is very easy to compare the result of this project with state of the art models.

Seven metrics are defined for the evaluation of the output of a network. All of them, are based on an error value. To compute this error value, the dot product between normalised predicted normal maps and the desired normal maps for all the samples of the test data set is calculated. Then, the arc cosine function is applied on these values, and the result is converted to degree. In other words, the angle between the corresponding normal vectors in estimated normal maps and desired normal maps is computed in degree. Then the following metrics are calculated based on these angles over the valid pixels in the data set.

- *Mean*: the average of angles between normal vectors
- *Median*: the median of angles between normal vectors
- *RMSE*: $\sqrt{\frac{1}{n}\sum_{i=1}^{n} E_i^2}$, for E= angle between normal vectors and n= number of valid pixels in data set
- 11.25° *Error*: the percentage of valid vectors with angles less than 11.25°
- 22.5° *Error*: the percentage of valid vectors with angles less than 22.5°
- 30° *Error*: the percentage of valid vectors with angles less than 30°
- 45° *Error*: the percentage of valid vectors with angles less than 45°

5.3 Quantitative results

The result of experiments are illustrated in table 5.1. Based on the results obtained from several times training the baseline model with same configuration, a difference more than one percent in 11.25° error is statistically significant.

5.4 Qualitative results

Figures 5.1 and 5.3 illustrate some of the better results obtained by training the baseline model on the main data set. The results of training the VGG16 model on SUN RGB-D based data set are depicted in figures 5.2 and 5.4. Generally the models perform their best in scenes with a few big and simple objects like beds or tables.

For comparison, the qualitative results of relevant work to this project is also included (figure 5.5). It is clear that there is a big difference between the results obtained in this project and the state of the art.

Experiment	Mean	Median	RMSE	11.25°	22.5°	30°	45°
Baseline	43.5	39.7	51.0	11.2	27.3	37.6	56.5
12 Conv. Filters	43.6	39.4	51.3	11.6	27.7	38.1	56.6
15x15 Conv. Kernel	43.1	39.5	50.5	10.7	27.0	37.7	56.8
Avg. Pooling	43.8	40.2	51.1	9.9	26.0	36.7	56.0
Larger FC1	43.8	40.0	51.2	10.1	26.2	36.9	56.1
Larger FC2	43.0	39.1	50.5	11.3	27.7	38.2	57.2
No FC1	47.5	43.5	55.2	9.0	24.4	34.1	51.7
Up Sampling	40.4	37.1	47.3	11.9	29.0	40.3	60.4
VGG16	40.7	36.4	48.5	13.8	31.1	41.8	60.6
VGG Low	44.1	40.1	52.0	10.9	27.0	37.3	55.9
VGG Mid	42.2	37.2	50.5	13.6	30.8	41.2	59.0
VGG High	43.0	38.1	51.0	12.0	29.2	39.8	58.0
VGG Concat.	48.5	44.4	55.0	4.7	17.5	28.7	50.9
ResNet50	45.0	41.6	50.5	4.7	17.5	29.6	55.3
Baseline NYU	42.2	38.2	50.0	12.3	28.5	39.2	58.7
Baseline SUN	45.2	42.2	52.7	10.5	24.9	34.7	53.5
VGG16 NYU	41.0	37.7	47.9	10.6	27.3	38.8	60.4
VGG16 SUN	42.2	38.4	50.0	13.0	29.0	39.4	58.2
Wang et al. [18]	25.0	13.8	35.9	44.2	63.2	70.3	
Eigen et al.[11]	23.7	15.5	-	39.2	62.0	71.1	
Dharmasiri et al.[12]	20.6	13.0	-	44.9	67.7	76.3	
Bansal et al.[21]	19.8	12.0	28.2	47.9	70.0	77.8	

Table 5.1: Quantitative results of evaluation

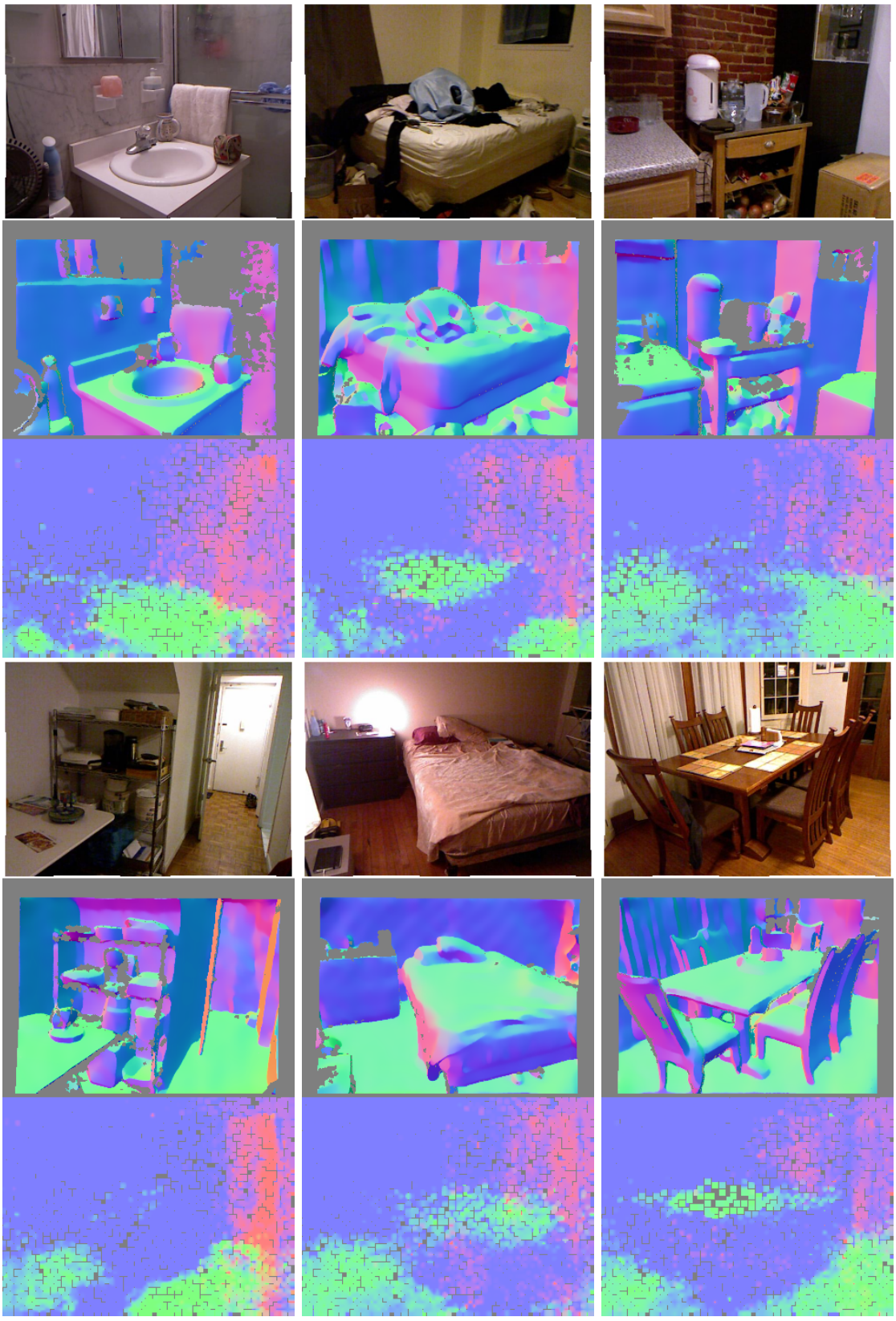

Figure 5.1: Baseline model trained on main data set:
RGB images, ground truth normal maps, and estimated normal maps

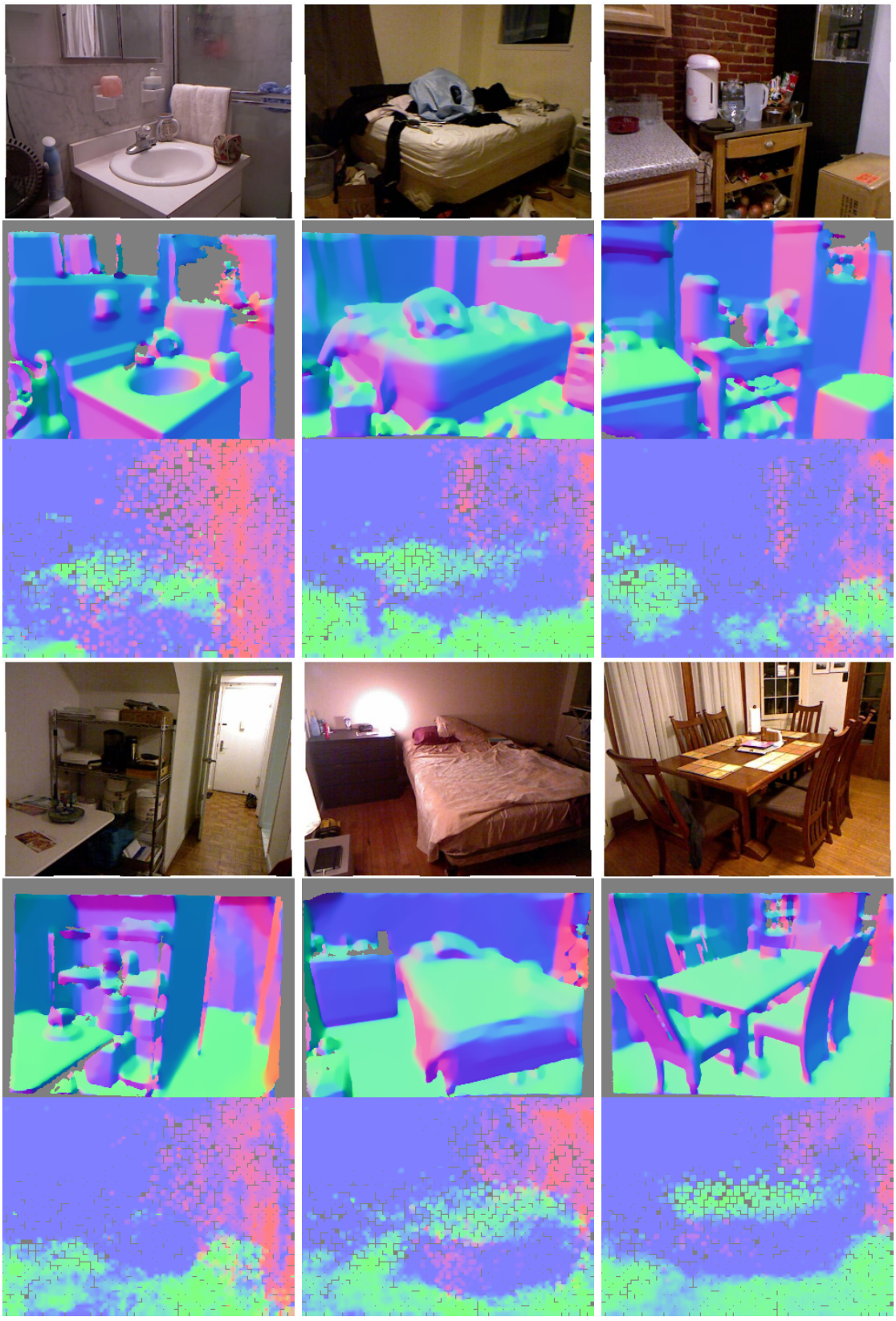

Figure 5.2: VGG16 model trained on SUN data set:
RGB images, ground truth normal maps, and estimated normal maps

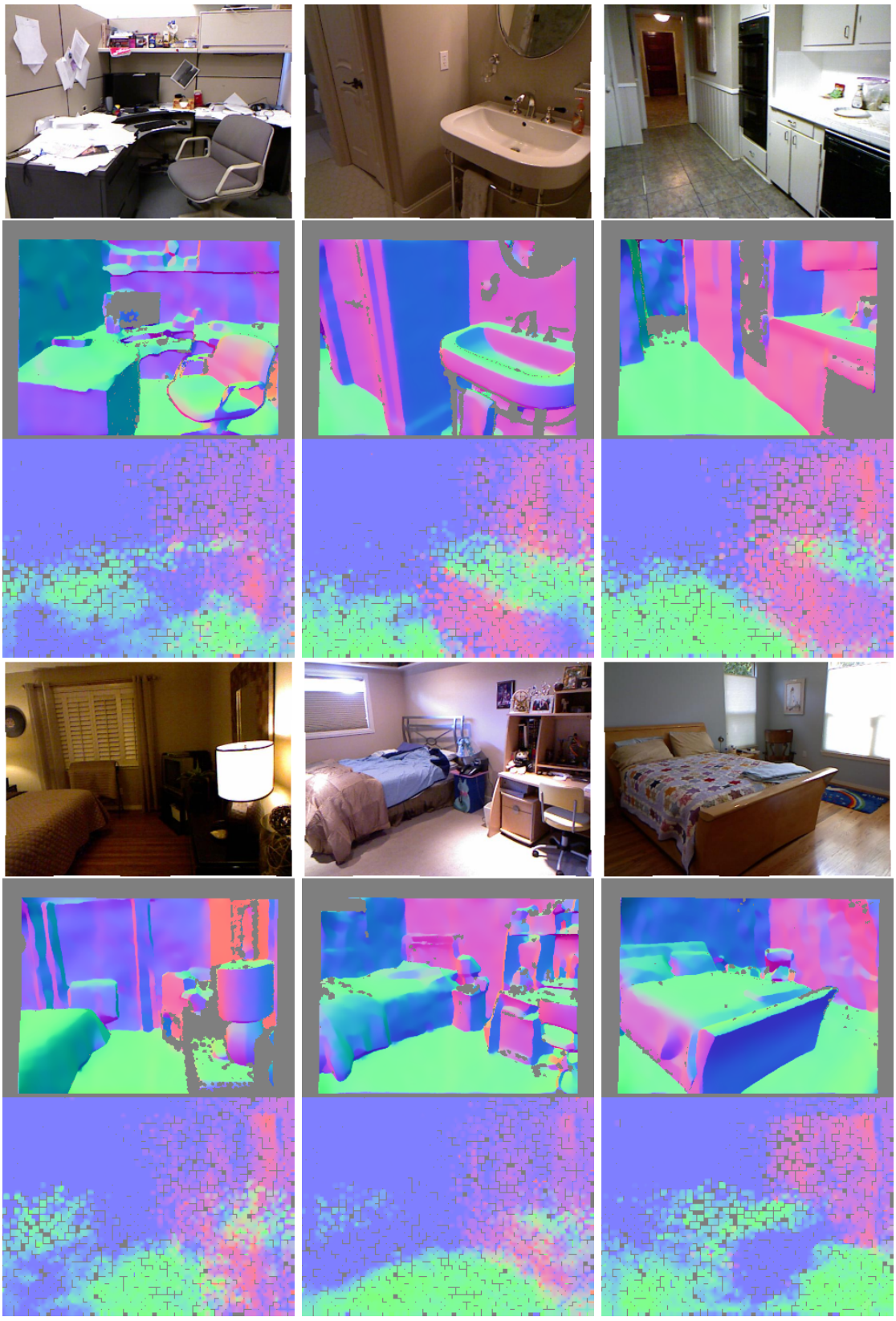

Figure 5.3: Baseline model trained on main data set:
RGB images, ground truth normal maps, and estimated normal maps

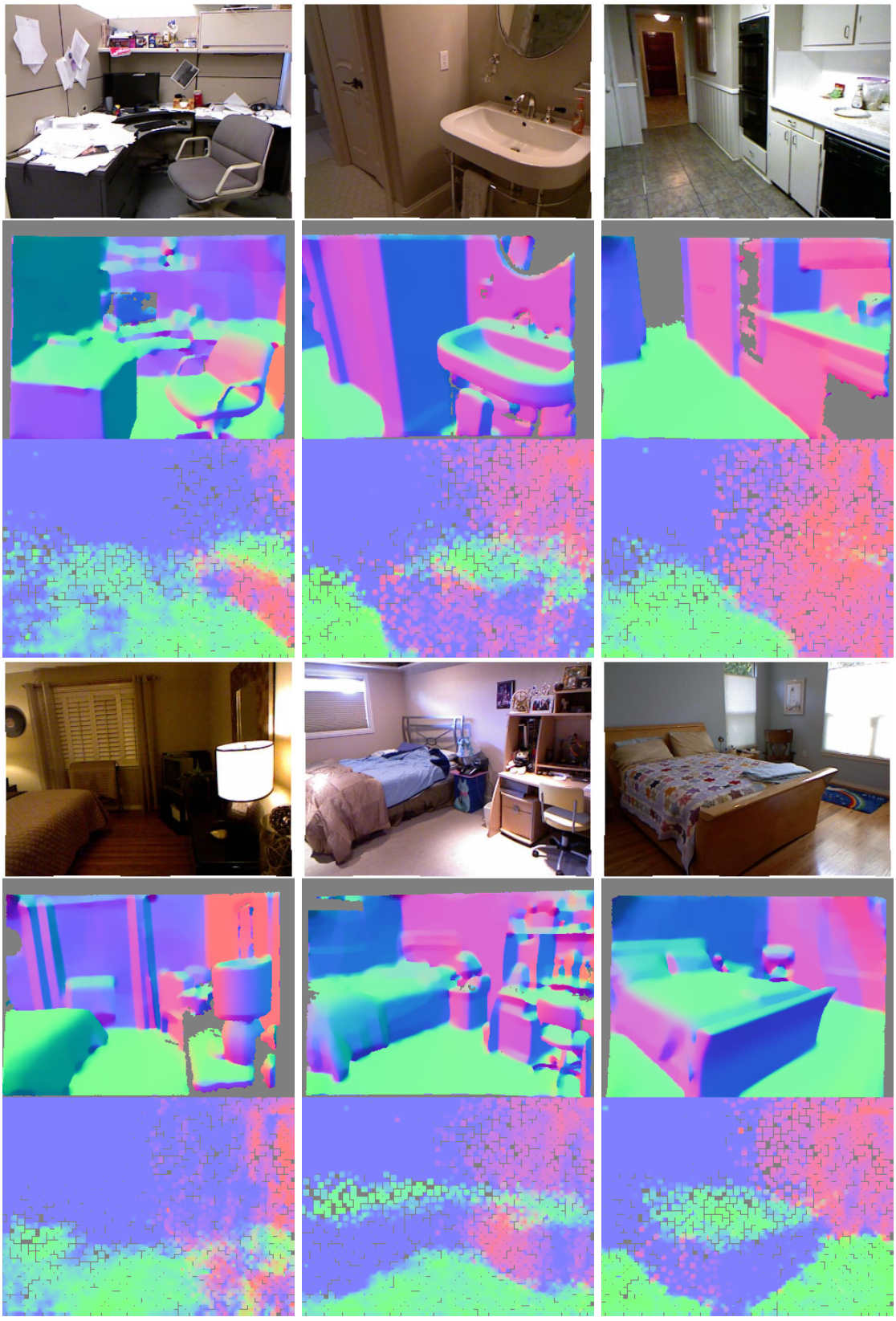

Figure 5.4: VGG16 model trained on SUN data set:
RGB images, ground truth normal maps, and estimated normal maps

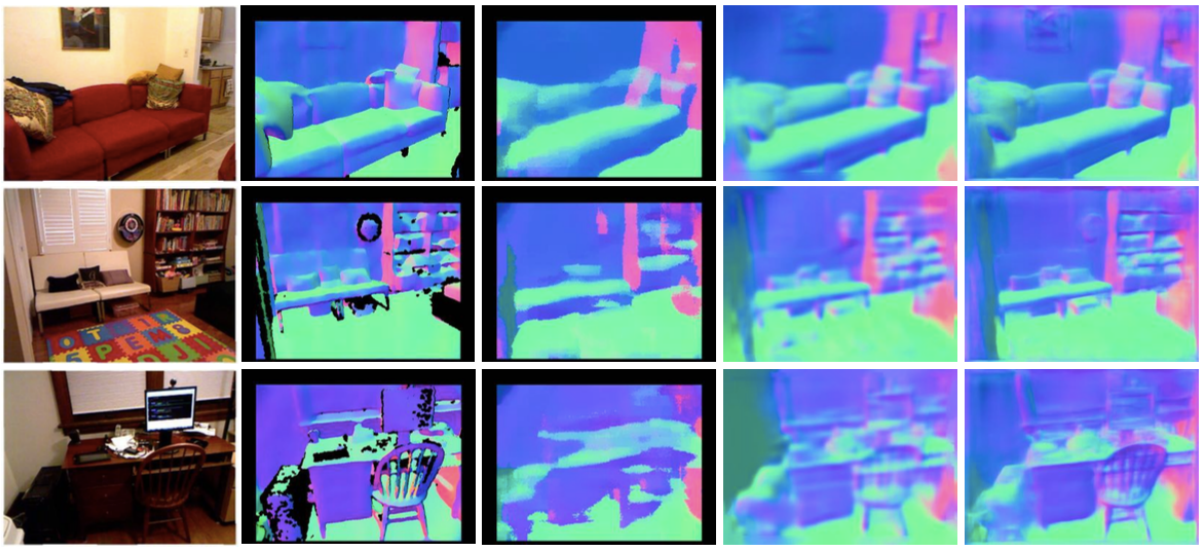

Figure 5.5: From left to right: RGB images, ground truth normal maps, qualitative results of Wang et al., Eigen et al., and Bansal et al.

Chapter 6

Conclusion

Based on the quantitative evaluation of the models, no statistically significant change to the outcome was made by changing the size or number of the filters in last convolutional layer of baseline model. Use of average pooling instead of max pooling in the network, considerably decreased the performance. A larger outer fully connected layer did not make any difference in the results, while larger inner fully connected layer decreased the performance of the network. By removing the first fully connected layer, the performance was degraded considerably and the application of a different up sampling method was not able to make any difference.

Overall, the VGG16 model had the best performance between the implemented architectures. Using the lower and higher parts of the network decreased the performance, while using the mid layers made no difference in output.

Surprisingly, both of the more complex models, the model that used the concatenation of different layers in VGG16, and the ResNet50, had a very bad performance. One can argue that the main reason for this might be the lack of enough training data.

By using the data set based on the SUN RGB-D, the performance of baseline model decreased but VGG16 showed a better performance. This indicates that more complex models could benefit from training on a higher quality data set.

The results show a large gap between the surface normal maps estimated by models of this project and the state of the art. Although it worth noting that the state of the art models are trained on a much larger data set (200K samples from raw NYU data set).

6.0.1 Achievements

Based on the objectives defined for this project, an analysis of the previous research was conducted. The published source codes were studied and the network architectures were compared. Finally, the weaknesses and points of strength in each model were discussed in section 2.2.

Data required for use in this project was obtained from two publicly available data sets. MATLAB code was developed to compute the ground truth surface normal maps based on two different methods and the results were used to produce the final data sets.

A deep learning pipeline was developed and several different network architectures were designed and implemented. Various experiments were conducted to explore the improvements to the baseline model.

Finally, well-established evaluation metrics were implemented and the quantitative and qualitative result of evaluation were presented and compared to the state of the art methods.

6.0.2 Future Work

Improving the result of this project is possible based on two previously mentioned aspects: designing better network architectures and the use of more and higher quality training data. The state of the art models are trained on a much larger data set (200K samples from raw data set in comparison to 795 samples from training data set that were used in this project). In

this project only a subset of SUN RGB-D data set was used. Using all the samples of the SUN RGB-D data set or the raw data from NYU Depth data set, to train the models designed in this project and comparing the results is another subject for investigation.

Designing network architectures that utilise several different levels of data representation looks as a good direction for developing future models. Also, the structure of the top layers of the network is an area which is not explored in depth and may yield improved results.

6.0.3 Personal Reflection

After getting the confirmation about the topic of my MSc project, I decided to do a comprehensive background research before starting the work on project. Although reading tens of scientific articles and understanding the various technical or theoretical concepts was difficult, my knowledge about the field of computer vision was considerably improved. Sometimes understanding concepts that were described in a few sentences in an article, needed a vast amount of background knowledge. The background research helped me to have a clear understanding of the problem and inspired most of the solutions to the various problems arose during the project.

Implementing a working pipeline and a baseline model was very challenging. Although the length of the final source codes was relatively small, almost for implementing every functionality, spending many days and referring to various documentations was needed. Learning all the required deep learning concepts and software libraries such as TensorFlow in a short period of time was not an easy task.

One of the main challenges in this project was the computational resources needed for training and evaluation of the models. Working on my laptop or PC was not possible due to limited GPU processing power. I remotely used the HPC servers provided by university for running and debugging the models. Having a limited access with a command line interface to the server was one of the other challenges. Because it was not possible to use the root account, automatically installing software packages was not feasible. A lot of time was spent for finding workarounds for setting up the development environment on the servers.

The raw data set was very big (400GB) and after running MATLAB batch jobs for several days for computing the surface normals; due to resource constraints it was not possible to actually use this data in this project. Also, using two different programming languages (MATLAB and python) for different components of the project and maintaining the interoperability between these components was another challenge in this project.

Developing and testing the code in interactive development environments (MATLAB scripts and Jupyter notebooks for python) was very effective in debugging and testing the code and experimenting different code snippets. It was always possible to check the value of variables at any point instead of using a more traditional debugging approach. By using git version control system, the code management and synchronisation between servers and development environment was made much easier.

Finally, using LaTeX for writing the report and managing the documents on a version control system considerably increased my productivity in the last phase of this project.

References

[1] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. "Sun rgb-d: A rgb-d scene understanding benchmark suite". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 567–576.

[2] Samuel F. Dodge and Lina J. Karam. "A Study and Comparison of Human and Deep Learning Recognition Performance Under Visual Distortions". In: *CoRR* abs/1705.02498 (2017).

[3] C. Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60 (2012), pp. 84–90.

[5] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015.

[6] Song Han et al. "Learning Both Weights and Connections for Efficient Neural Networks". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 1135–1143. URL: http://dl.acm.org/citation.cfm?id=2969239.2969366.

[7] Yanming Guo et al. "Deep learning for visual understanding: A review". In: *Neurocomputing* 187 (2016), pp. 27–48.

[8] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *CoRR* abs/1603.07285 (2016).

[9] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014).

[10] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016).

[11] David Eigen and Rob Fergus. "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2650–2658.

[12] Thanuja Dharmasiri, Andrew Spek, and Tom Drummond. "Joint Prediction of Depths, Normals and Surface Curvature from RGB Images using CNNs". In: *arXiv preprint arXiv:1706.07593* (2017).

[13] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. "3-D Depth Reconstruction from a Single Still Image". In: *International Journal of Computer Vision* 76 (2007), pp. 53–69.

[14] Changick Kim Jaeseung Ko Manbae Kim. *2D to 3D stereoscopic conversion: depth-map estimation in a 2D single-view image*. 2007.

[15] B. Liu, S. Gould, and D. Koller. "Single image depth estimation from predicted semantic labels". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. June 2010, pp. 1253–1260.

[16] Fayao Liu, Chunhua Shen, and Guosheng Lin. "Deep Convolutional Neural Fields for Depth Estimation From a Single Image". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

[17] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network". In: *Advances in Neural Information Processing Systems 27*. 2014, pp. 2366–2374.

[18] Xiaolong Wang, David Fouhey, and Abhinav Gupta. "Designing deep networks for surface normal estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 539–547.

[19] Varsha Hedau, Derek Hoiem, and David Forsyth. "Recovering the spatial layout of cluttered rooms". In: *Computer vision, 2009 IEEE 12th international conference on*. IEEE. 2009, pp. 1849–1856.

[20] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[21] Aayush Bansal, Bryan Russell, and Abhinav Gupta. "Marr Revisited: 2D-3D Model Alignment via Surface Normal Prediction". In: *CVPR*. 2016.

[22] Bharath Hariharan et al. "Hypercolumns for object segmentation and fine-grained localization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 447–456.

[23] Hachim El Khiyari and Harry Wechsler. "Face recognition across time lapse using convolutional neural networks". In: *Journal of Information Security* 7.03 (2016), p. 141.

[24] Chen Sun et al. "Revisiting unreasonable effectiveness of data in deep learning era". In: *arXiv preprint arXiv:1707.02968* (2017).

[25] Nathan Silberman et al. "Indoor segmentation and support inference from rgbd images". In: *Computer Vision–ECCV 2012* (2012), pp. 746–760.

[26] Lubor Ladicky, Bernhard Zeisl, and Marc Pollefeys. "Discriminatively Trained Dense Surface Normal Estimation". In: *ECCV*. 2014.

[27] Kevin Lai et al. "A large-scale hierarchical multi-view rgb-d object dataset". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1817–1824.

[28] Arjun Singh et al. "Bigbird: A large-scale 3d database of object instances". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 509–516.

[29] S Karayev et al. "A category-level 3-D object dataset: putting the Kinect to work". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2011, pp. 1167–1174.

[30] Anat Levin, Dani Lischinski, and Yair Weiss. "Colorization using optimization". In: *ACM transactions on graphics (tog)*. Vol. 23. 3. ACM. 2004, pp. 689–694.

[31] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. "Sun3d: A database of big spaces reconstructed using sfm and object labels". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1625–1632.

[32] Andrew Spek, Wai Ho Li, and Tom Drummond. "A Fast Method For Computing Principal Curvatures From Range Images". In: *Proceedings of the Australasian Conference on Robotics and Automation (ACRA)*. 2015.

[33] Kristian Bredies, Karl Kunisch, and Thomas Pock. "Total generalized variation". In: *SIAM Journal on Imaging Sciences* 3.3 (2010), pp. 492–526.

[34] Stanley H. Chan et al. "An augmented Lagrangian method for total variation video restoration". In: *IEEE Transactions on Image Processing* 20.11 (2011), pp. 3097–3111. URL: `http://videoprocessing.ucsd.edu/~stanleychan/deconvtv`.

[35] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

Appendices

Appendix A

Source Code and Data Sets

The source code and instructions for obtaining the data sets and training the models is provided on the web page of this project on GitHub web site:

`https://github.com/kaykanloo/msc-project`

Appendix B

Ethical Issues Addressed

The data sets used in this project are publicly available for research purposes. The credit has been given in the source codes or project documents when an external library or code snippet was used or adapted.