Depth Estimation from Monocular Image using Convolutional Neural Network

Saurav Sharma

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Depth Estimation from Monocular Image using Convolutional Neural Network

Thesis submitted in partial fulfillment

of the requirements of the degree of

Master of Technology

in

Computer Science and Engineering
(*Specialization*: *Computer Science*)

by

Saurav Sharma

(Roll Number: 215CS1074)

based on research carried out

under the supervision of

Prof. Pankaj Kumar Sa

May, 2017

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Prof. Pankaj Kumar Sa
Assistant Professor

May 20, 2017

Supervisor's Certificate

This is to certify that the work presented in the thesis entitled *Depth Estimation from Monocular Image using Convolutional Neural Network* submitted by *Saurav Sharma*, Roll Number 215CS1074, is a record of original research carried out by him under my supervision and guidance in partial fulfillment of the requirements of the degree of *Master of Technology* in *Computer Science and Engineering*. Neither this thesis nor any part of it has been submitted earlier for any degree or diploma to any institute or university in India or abroad.

_____
Pankaj Kumar Sa

Dedication

To my mother who always wanted me to do post-graduation.
To my father who never give up on me and for constant source of encouragement.
To Almighty for providing me strength and calmness to overcome my fears.

Declaration of Originality

I, *Saurav Sharma*, Roll Number *215CS1074* hereby declare that this thesis entitled *Depth Estimation from Monocular Image using Convolutional Neural Network* presents my original work carried out as a postgraduate student of NIT Rourkela and, to the best of my knowledge, contains no material previously published or written by another person, nor any material presented by me for the award of any degree or diploma of NIT Rourkela or any other institution. Any contribution made to this research by others, with whom I have worked at NIT Rourkela or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the sections "Reference" or "Bibliography". I have also submitted my original research records to the scrutiny committee for evaluation of my dissertation.

I am fully aware that in case of any non-compliance detected in future, the Senate of NIT Rourkela may withdraw the degree awarded to me on the basis of the present dissertation.

May 20, 2017
NIT Rourkela

Saurav Sharma

Acknowledgment

This thesis is a culmination of a year of failures, success, endless discussions and constant motivation of several people around me. Simply naming them would not be enough to justify their contribution in shaping the thesis.

My sincere gratitude to my advisor *Prof. Pankaj Kumar Sa* for allowing me to pursue my interests and motivating me to go beyond limits. His constant guidance and support helped me evolve as a researcher.

Sincere thanks to *Prof. B. D. Sahoo* and *Prof. S. Bakshi* for instilling the right mindset for solving problems.

My humble acknowledgment to the Head of Department, *Prof. D. P. Mohapatra* for allowing me to use lab resources for carrying out research.

Immense regard for my PhD seniors *Ram P. Padhy* and *Suman K. Choudhury* who stood with me throughout the research and guided me whenever needed. Many thanks to my friends and other research fellows for being with me.

Finally, my eternal gratefulness to my beloved parents for their love, support and constant push to reach my goals. Forever thankful for the sacrifices they made for my betterment.

May 27, 2017 — *Saurav Sharma*

NIT Rourkela — Roll Number: 215CS1074

Abstract

Estimating depth from an image is an important task for understanding physical geometry of a scene. It becomes challenging when only single images are considered for estimation due to lack of depth cues in a single input image. In this thesis, a fully convolutional architecture is proposed which learns the complex mapping of the pixels in the input RGB image and its corresponding depth image. The architecture employs DenseNet, a state-of-the-art CNN with an added network of deconvolution layers to estimate the depth image. It is free from any post-processing layers like CRF which increases the time taken for estimation on test data. Transfer learning paradigm is used in which pre-trained CNN are fine-tuned for depth estimation task. The proposed architecture inherently performs feature reuse and feature propagation without needing multiple scales of convolution layers. The network is evaluated on NYU Depth V2 indoor dataset consisting of multiple video frames of different indoor environments taken from Microsoft Kinect camera. The unified architecture is trained end-to-end and optimization performance are compared for berHu and RMSE loss functions. Exhaustive simulation on benchmark dataset not only reveals the superiority of the proposed model over the current state-of-the-art but also requires significantly less number of training samples for convergence.

***Keywords*: *Depth Estimation*; *Deep Learning*; *Monocular Image*; *Convolutional Neural Network*.**

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Rise of Deep Learning

Deep learning over the years has become a household name among artificial intelligence communities. Its stellar use in computer vision problems has resulted in huge improvements over its counterpart pre deep learning methods. Before that, computer vision problems were mostly solved by shallow machine learning algorithms such as Support Vector Machines (SVM), Multi-layer perceptron with one hidden layer etc. These shallow learning algorithms relied on fewer mappings of the input features to output label compared to deep learning algorithms which incorporate a large number of mappings. Different computer vision problems demand a different set of features like color, textures, edge orientation, etc. For example, features generated with SIFT algorithm are used in iris recognition and HOG features are used in pedestrian detection. So, same features cannot be used for other types of datasets. Engineering features is a time-consuming task and it requires considerable domain expertise and it may not be feasible nowadays for a wide range of computer vision problems. This was the reason feature learning was getting more importance than feature engineering. In feature learning, the algorithm learns to extract features from a given dataset and it is done automatically.

Deep learning is a technique where multiple layers are used in an architecture to learn the mapping of input features to their corresponding target label. The features from a layer are stacked together, passed through a non-linear function to extract information of higher complexity from the data. For example, extracting edge, blobs requires less number of layers, but to extract a face or an object from input data will require features at multiple levels. Much of the success in deep learning methods can be attributed to two major components - large size dataset and GPU computing. Over the years, computational resources required to implement deep learning methods have become cheaper and more research has been done in deep learning since then. Initial deep learning based architectures suffered from problems such as vanishing (exploding) gradients, overfitting, unavailability of fast optimizers, etc. Vanishing (exploding) gradient problem is solved by using ReLU as the activation function. Dropout and batch-normalization layers have prevented overfitting of the network and the

training has become faster. Optimizers such as RMSProp, SGD and its variants like Adam, AdaDelta etc. have resulted in the faster convergence of the network.

Recently, in deep learning, transfer learning method is applied when large-scale datasets are not available. In transfer learning method, a network pre-trained for a problem can be used for other similar problem. It usually works because the pre-trained network has the knowledge to extract relevant features from the data. The network can later be trained on problem-specific dataset requiring less time for convergence. Nowadays, deep architectures are usually trained on ImageNet, CIFAR datasets and their weights are used for training on a different dataset.

1.2 Convolutional Neural Network

In the early experiments performed by Hubel and Weisel [2] in 1968 on cat's visual cortex, it was found that many cells are structured together in a complex manner where each of these cells responds to a local region of the visual field. These cells are thus formed by sliding over the local regions of the visual field by behaving as a local filter to find structural similarity in an image. This and other sets of experiments based on this work were the fundamental ideas behind the emergence of convolution neural network. Convolution Neural Network (CNN) is a special type of architecture where the spatial structure of pixel values in the input image are exploited for recognizing patterns in an image compared to a regular neural network where all the pixels in the image taken at input layer are weighted equally. CNN works on the input image by producing output feature map where each of the values in the output feature map corresponds the output obtained from the overlapping regions in the input layer.

Convolution neural network has many advantages compared to a regular neural network. Since convolution neural network works on local regions of the input image, training of the network becomes faster. This would lead to an explosion in the number of parameters due to which overfitting of the network can occur. CNN are constrained to be used for images only where the features learned over the local regions forms a volume of output neurons ordered across the depth, width, and height.

These features are learned by convolving a parameterized filter over the local regions. To further reduce the number of parameters involved in a CNN, a filter typically has same weights across the input image volume. This is based on the assumption that a filter can be used in other spatial locations of the image for identifying a feature which was found in other spatial location.

Using CNN in majority of the computer vision problems is greatly improving the performance of the models used in tackling these problems. Earlier models based on hand-crafted features are slower compared to CNN as they require extra time in manually finding and accumulating the features to fit into the model. Also, crafting these features manually requires a huge domain expertise. CNN avoids these by learning features on its

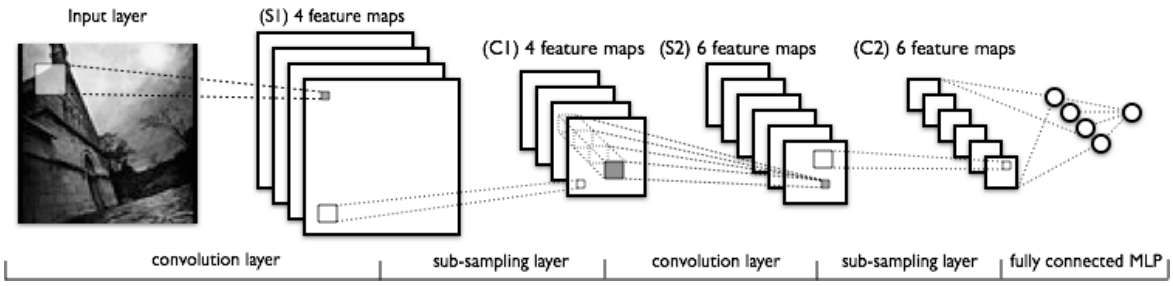


Figure 1.1: LeNet Architecture taken from deeplearning.net

own from the input images using the raw pixels only. The spatial structure and correlation between adjacent pixels in an input image helps the model to find features in layers that build upon features found in previous layers. This leads to an efficient representation of the features across varied degree of complexity level i.e. from lower to higher. For example, features like edges in the case of human face as input image are combined to form middle level features like ears, noses which are combined to form higher level features like different possible structure of faces. Figure 1.1 shows a CNN based LeNet architecture where the different components of CNN are stacked together for performing computer vision task.

1.3 Components of Convolutional Neural Network

Convolutional neural network can be build by stacking number of layers which are mainly Convolution layer, Pooling Layer, Activation Layer, Regularization Layer and Fully Connected Layer. Convolution and Pooling layers are the primitive layers used majority of the time in all of the CNN based deep learning architectures. Each of these layers transform an input feature volume to an output volume feature volume by means of non-linear and differentiable functions. Each of these components along with activation functions and regularization functions are discussed in this chapter.

1.3.1 Convolution Layer

Convolution layer is the basic module used for constructing any CNN based architecture. It primarily operates on images as input feature volume with dimensions such as width, height, and depth. A single convolution layer may consist of one or more parameterized filters (kernels) which scan all the local regions in the input volume and look for features present in these regions.In the forward propagation, a filter is convolved with all the local regions present in an input feature volume one at a time. It basically performs convolution operation of the filter parameters with the feature values in a local region. This produces a single output for a single local region. So, convolving a single filter through all the local regions in an input feature volume produces an output feature map of 2 dimensions. A filter operates on the input feature volume across its depth. The depth of the filter used in a convolution

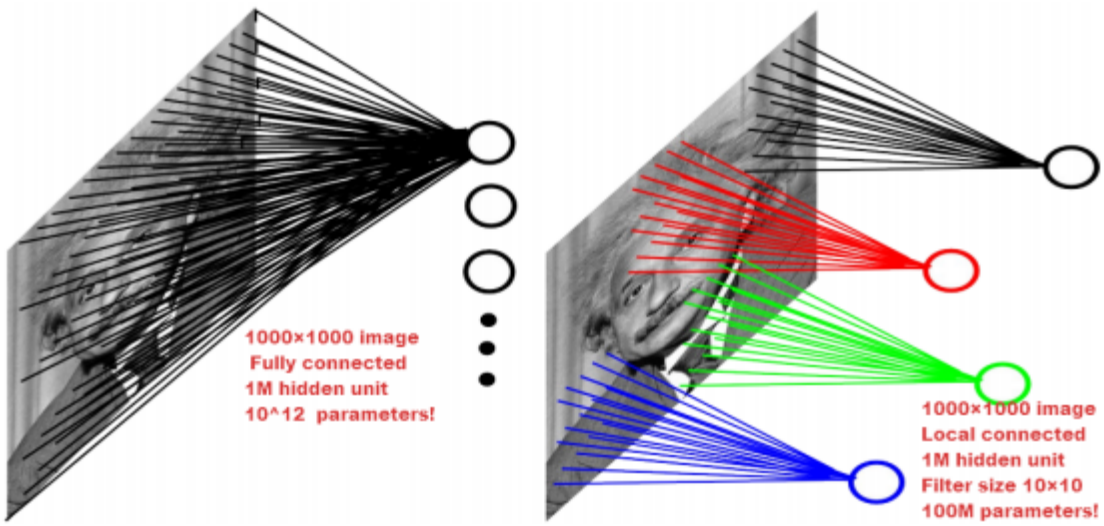


Figure 1.2: Comparison between convolutional layers with locally connected and fully connected regions. **Source:** www.deeplearning.net

layer depends on the depth of the input feature volume on which it is convolved. The output feature maps generated for each convolution operation by a set of filters are stacked together to form a bigger output feature volume. Other layers in a CNN architecture operate on this stacked feature volume. Convolution layers or CNN architecture, in general, exploit two important ideas which are localized connections and parameter sharing.

In a convolution layer, the filters operate on the local region of the input feature volume. Compared to a regular neural network where there is a mapping from all the neurons in layer L_i to an output neuron in layer L_{i+1}, convolution layer maps a local region to an output neuron in the next layer. These parametric links in a regular neural network require a large number of parameters which is impractical for images with large spatial resolution. Convolution layer reduces the number of parameters used for mapping by operating only in the local regions. The spatial range of the local region on which the filter operates is governed by a hyper-parameter called **receptive field** or **filter size** (F). The filter scans through the local regions of the input volume and it is extended across the input volume. Parameter sharing is another important concept in the context of convolution layer operation. It is based on the assumption that a filter used in one local region can be reused in other local regions of the input feature volume directly instead of learning it for every occurrence of the feature. This helps in greatly reducing the number of parameters involved for a convolution layer by a huge margin. Figure 1.2 shows the advantage of filters convolving over a local region in a convolution layer.

The dimensions of the output feature volume obtained after a single pass of convolution layer on an input feature volume are governed by three hyper-parameters viz. depth (K), stride (S) and zero-padding (P).

- **Depth (K)** hyper-parameter is the number of filters used in a convolution layer and it

does not affect the spatial size of the output feature volume.

- **Stride (S)** hyper-parameter is the number of unit steps by which the filter is slided on an input feature volume both horizontally and vertically. This hyper-parameter reduces the final spatial size of the output feature volume.
- **Zero-padding (P)** hyper-parameter is the number of zeros that is padded to an input feature volume. Zero-padding is generally done before the convolution operation to maintain the same output spatial size as that of input feature volume. Depending on the amount of zero-padding used, the final spatial size of the output feature volume changes accordingly.

The equations for calculating the dimensions of output feature volume of size W_1xH_1xD_1 after applying a convolution layer on an input feature volume of size W_0xH_0xD_0 with the above hyper-parameters is given in the equation 1.1.

$$W_1 = (W_0 - F + 2P)/S + 1 \tag{1.1a}$$

$$H_1 = (H_0 - F + 2P)/S + 1 \tag{1.1b}$$

$$D_1 = K \tag{1.1c}$$

1.3.2 Pooling Layer

Pooling layer in a CNN architecture is placed after convolution layers to reduce the spatial size of the input feature volume. Pooling layer summarizes the information present in a bigger input feature volume to a smaller output feature volume. By reducing the spatial size of the input feature volume, the parameters involved are also reduced which helps control overfitting in CNN and improve performance. Pooling helps to elucidate the information generated by a preceding convolution layer producing a compressed feature map. There are variants of pooling function used in a pooling layer viz. max-pooling, average-pooling etc.

Max-Pooling is one of the commonly used pooling technique in CNN. It operates by taking the maximum of the values present in a local region as the final value in the output feature volume. In pooling operation, the depth of the output feature volume is same as that of input feature volume. In average pooling, the average of all the values present in the local region is taken as the final value in output feature volume. In recent CNN based architectures, max-pooling is generally favored over other types of pooling operation as it has been found to give better results practically. Pooling operation only reduces the spatial

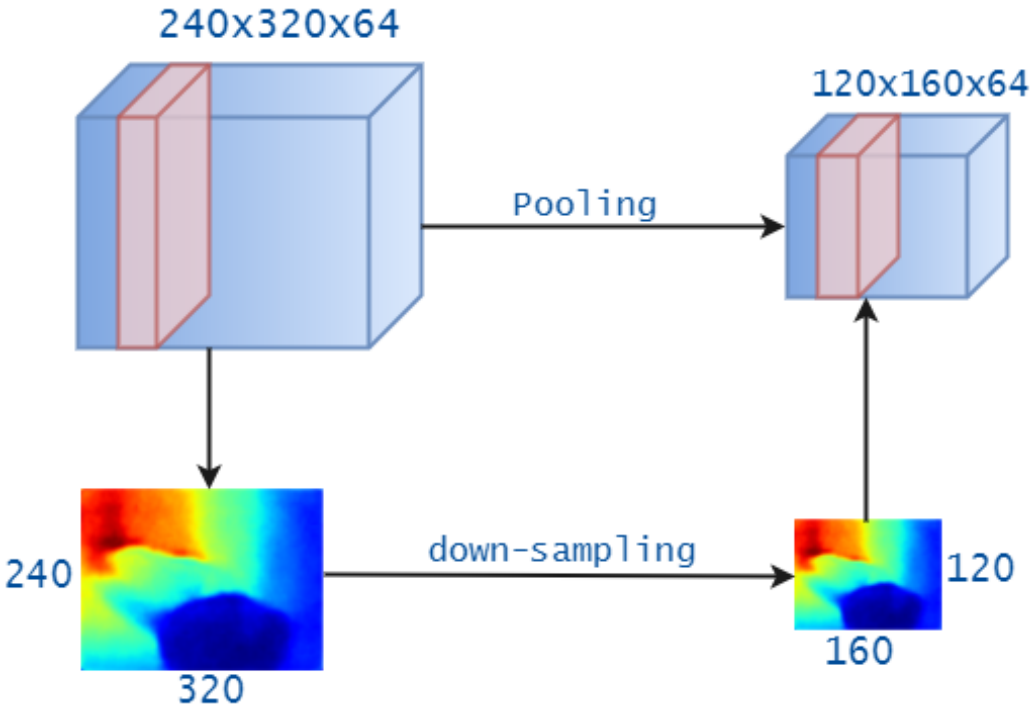


Figure 1.3: Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [240x320x64] is pooled with filter size 2, stride 2 into output volume of size [120x160x64]. Notice that the volume depth is preserved. **Right**: The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

size of the input feature volume and it does not involve any parameters. Figure 1.3 shows an example of pooling operation. The mathematical form of output feature volume of size W_1xH_1xD_1 after applying a pooling layer on an input feature volume of size W_0xH_0xD_0 is given in the equation 1.2.

$$W_1 = (W_0 - F)/S + 1 \tag{1.2a}$$

$$H_1 = (H_0 - F)/S + 1 \tag{1.2b}$$

$$D_1 = D_0 \tag{1.2c}$$

Max-pooling layer, in general, carries forward the features present in input feature volume to the next layer in CNN. It incorporates translation invariance by producing similar output as that of a convolution layer. For example, a number present in different spatial locations in two images are max-pooled to same output feature volume which is then correctly classified by a classifier. This reduces the computation time during training as parameters are reduced and generalizes the CNN architecture thereby achieving faster convergence rate.

1.3.3 Fully Connected Layer

A fully connected layer is equivalent to a regular neural network where a neuron in one layer is connected to all the neurons in the next layer. This introduces a large number of parameters for each fully connected layer due to which the training becomes slower. The fully-connected layer contributes the majority to the number of parameters of an architecture. In recent CNN architectures, fully connected layers are generally replaced by equivalent convolution layers which speed up the training in practical situations. Convolution layer when used in place of a fully-connected layer, reduces the number of parameters and these parameters are shared across the input feature volume for a filter. There is a problem with the fully-connected layer where it constrains the CNN architecture to accept only fixed size inputs. In convolution layer, only filters are learned for feature identification, unlike regular neural network where the entire mapping from one layer to the next layer is learned. In practice, the output feature volume obtained from CNN layers is squashed to a single dimension vector of features before forwarding to a fully connected layer(s) and output is produced as per computer vision problems which can be either classification or regression. Since the fully-connected layer is controlled by a number of neurons present in it, it is parametric in nature having a single dimension.

The layers explained above forms the basis of a convolutional neural network architecture. Different type of layers are usually stacked together in any order, but in general, for various computer vision tasks, the sequence of layers should be *convolution→pooling→fully-connected* .

1.3.4 Activation Layer

An activation layer is an essential component of CNN architecture. It is required to incorporate nonlinearity in the CNN architecture by using an element-wise non-linear function on the input feature volume. It helps in modeling complex nonlinear functions in the CNN architecture. A node in CNN is activated only when the weighted sum of input information is greater than a threshold value. In general, activation function decides on whether a particular input data should be forwarded to the next layer or not. There are different types of activation functions used in CNN such as sigmoid, tanh, ReLU, LeakyReLU, ELU, etc. These activation functions have its own advantages and disadvantages and are explained below.

sigmoid

The sigmoid function is heavily used activation function in a neural network and earlier deep learning architectures. It implements nonlinearity by using the function given in the equation 1.3.4. The sigmoid activation curve is shown in Figure 1.3.4(a).

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1.3}$$

The sigmoid function takes in a real-valued number and squashes the number in the range between 0 and 1. The sigmoid curve has a constant slope for extreme positive or negative numbers as it is evident from Figure 1.3.4(a). Sigmoid function suffers from vanishing (or exploding) gradient problem which can slow down the training process and hence it is not preferable in practical situations. Vanishing or exploding gradient occurs when a sigmoid function has a constant slope at extreme ends of the sigmoid curve. This makes the gradients calculated on these ends during backpropagation to have either very high or very low values. If the gradient is too high, then it will blow the gradient updates causing the network to overshoot the minimum and become unstable. If it is too low, there will be minimal or no change at all in parameters during gradient updates. Then, the network saturates and does not learn anything at all or in plain terms, the neuron dies. The activations produced by sigmoid functions are not centered around the mean. This leads to slow convergence of the network as the gradients required for updating the parameters of the network are biased completely in either positive or negative direction. This induces a zig-zag behavior when parameters are updated. Another disadvantage of using sigmoid function is the computation of exponential function for each element in the feature volume which is computationally expensive.

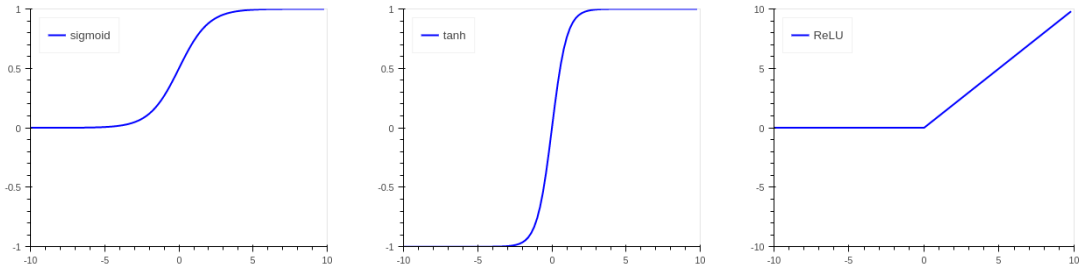
tanh

The tanh function is another non-linear activation function and limits the input feature values in the range -1 to 1. Compared to the sigmoid activation function, the activations of tanh function are zero-centered. It also suffers from vanishing (or exploding) gradient problem due to which the convergence of the network is slow and performs poorly in practice. The mathematical form of tanh activation function is shown in equation 1.4 and the curve are shown in Figure 1.3.4(b)

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{1.4}$$

relu

Rectified Linear Unit also popularly called ReLU is the recent widely used activation function in deep neural network architectures. The mathematical form and the curve are shown in equation 1.5 and Figure 1.3.4(c) respectively.



(a) sigmoid (b) tanh (c) ReLU

Figure 1.4: Activation Functions

$$f(x) = max(0, x) \tag{1.5}$$

ReLU activation function has many advantages over sigmoid and tanh activation functions. It does not suffer from vanishing (or exploding) gradient problem as observed in sigmoid and tanh. Hence, in ReLU backpropagation occurs smoothly which results in efficient training. Since the mathematical form of ReLU has only comparison operation, it is much computationally inexpensive than exponential operation present in both sigmoid and tanh activation functions. The feature volume obtained after applying ReLU activation function has sparse representations which are desirable compared to dense representations as it reduces the number of parameters involved.

But, there are disadvantages with ReLU activation function. First, it is not bounded at the top resulting in much higher values for the feature volume which can overwhelm the architecture. Second, the capacity of the network is reduced as some of the neurons make transition to a dead state for all the inputs. This problem is also known as dying ReLU problem. To remedy this problem, some variations are applied to existing ReLU activation function which results in small negative slopes for values less than 0. These modified ReLU activation functions are Leaky ReLU, Parametric ReLU, etc. discussed in Xu **et al.** [3].

1.3.5 Regularization Layer

CNN architectures are often deep and complex which involves a huge number of parameters (in millions). If the architecture is trained for low size dataset, then overfitting occurs resulting in the poor generalization of the architecture to unseen data. In overfitting, the architecture memorizes the training data and a large number of parameters increases the complexity of the architecture. Regularization layer is an important part of CNN architecture to prevent overfitting of the network. The solution to avoid overfitting is to suppress some parameters of the network which maps the complex function. It is done usually by adding an extra regularization function to the existing objective function of the network. The addition of regularization function propels the network to choose simple representations

instead of complex representations and helps optimization methods to perform better when the objective function is not convex. Regularization functions which are widely used in the CNN architecture are explained in this subsection.

L2-regularization

L2-regularization function along with L1-regularization are the fundamental regularization functions used in deep neural network architectures. It simply adds a regularization term to the objective function of the model as given in the equation 1.6.

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{1.6}$$

Here, C_0 is the existing objective function of the network and the regularization term consists of the squared summation of parameters of the network. A hyper-parameter called regularization parameter is used with the regularization term to control the amount of regularization that is applied to the network. L2-regularization helps to avoid the overfitting of the network to training data by favoring smaller values of the parameters.

Dropout

Dropout [4] is another regularization method added to the parametric layers to prevent overfitting. Dropout is generally incorporated between fully-connected layers which contributes the maximum to the number of parameters of the network. In Dropout during training, a random subset of neurons and its connections to the next layer are dropped and training is done on the remaining neurons of the layers as shown in Figure 1.5. It allows the layers to incorporate robustness in feature identification instead of memorizing it thereby reducing overfitting. In each training iteration when Dropout is used, less number of parameters are tuned compared to a large number of parameters originally present in the network. This way the network does not overfit and learns a structure present in the input feature volume. Dropout can be alternatively thought of as an ensemble of multiple networks with random neurons and their connections to the next layer being dropped. During the testing phase, all the neurons in the network are used for prediction.

1.3.6 Batch-Normalization

Recently since the inception of ResNet [5], the depth of the CNN architecture has increased tremendously. ResNet incorporated batch-normalization [6] as a regularization layer which gives superior results than Dropout and can be used to reduce overfitting. For a batch of input data, batch-normalization layer changes its mean and variance to be zero and one respectively before forwarding to the next layer. If batch-normalization is not used, then with each layer the features computed are not zero-centered for which the parametric layers take considerable

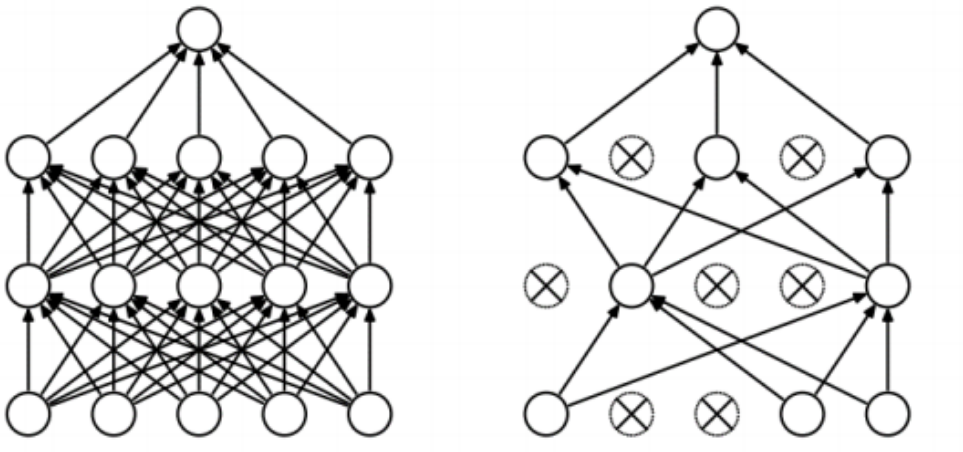


Figure 1.5: Dropout example. **Left:** Neural network without dropout. **Right:** Neural network with randomly dropped links between neurons of successive layers.

time to learn these features. This occurs because each layer has to adapt itself to learn features of different distribution and it mostly affects the performance in deeper architectures. With batch-normalization, layers in a CNN architecture can learn the distribution faster because the inputs for each of the layers will have the same distribution. Batch-normalization also speeds up the training and it is practically faster than networks which have Dropout as regularization function.

Generally, for data in each layer which are not zero-centered, the learning rate for gradient descent step during training must be small. It is desirable because each layer has to respond to data with a different distribution, so if higher learning rate is used, then the parameters of the network oscillate around the minimum of the objective function. Batch-normalization allows the model to use higher learning rates as data going into all the layers are always zero-centered. The distribution learned by batch-normalization layer can be better than the distribution with zero mean and unit variance. Batch-normalization gives better and faster convergence in training requiring fewer iterations than Dropout and is generally preferred in recent CNN architectures.

1.4 Thesis Organization

The following paragraphs outline the thesis layout with an emphasis on the contributions.

Chapter 1: Introduction This chapter introduces depth estimation from a single image as a computer vision problem. Deep learning methods are compared with traditional machine learning and feature engineering approaches are compared. Most of the works on computer vision employ Convolutional Neural Network (CNN) and a background information on CNN components are discussed in this chapter.

Chapter 2: Literature Survey This chapter discusses works that have been done in the field of depth estimation involving a single image. It discusses different deep learning methods like CNN and probabilistic graphical models used for estimating depth. Some of the existing works perform depth estimation and semantic segmentation jointly as there is a high correlation between these two tasks.

Chapter 3: Overview of CNN Architectures This chapter compares some of the CNN architectures that have been successful over the years in popular ImageNet [7] challenges. Different settings of filters and pooling operations for these CNN architectures with individual advantages and disadvantages are mentioned in this chapter.

Chapter 4: Depth Estimation using Fully Convolutional Architecture This chapter introduces the proposed network that has been used to perform depth estimation and the results obtained in different parameter settings. The proposed network consists of pre-trained DenseNet [8] with an added network of deconvolution blocks for gradually increasing the spatial resolution of the final depth estimate. The arrangement of layers present in a deconvolution block along with advantages with DenseNet is also discussed. The loss function required for optimization of the network parameters are mentioned along with their mathematical interpretations.

NYU Depth V2 [9] is discussed in details and preprocessing required for the images to be used as input for the proposed network. Different types of data augmentations performed on the dataset are mentioned. An evaluation of the results obtained after training is discussed in detail along with training loss curve. Finally, a table of standard dataset metrics is presented for comparison with existing methods for depth estimation.

Chapter 5: Conclusion This final chapter provides concluding remarks on the depth estimation performed by the proposed network and also provides pointers for further improvements. Possible extension of the depth estimation task to SLAM based applications involving obstacle detection, 3D scene reconstruction are also mentioned in this chapter.

Chapter 2

Literature Survey

This chapter discusses research that has been made in the field of vision-based depth estimation using both monocular and stereo approaches. The stereo based methods deploy a pair of cameras along one common plane and apply the triangulation geometry to obtain the depth using various cues such as, stereopsis, disparity, eye-convergence, and so forth. However, depth estimation from monocular images possesses many challenges owing to its ill-posed behavior in the absence of local and global information.

Preliminary work on monocular depth estimation usually applies various hand-engineered features [10], where coarse geometric characteristics of a scene are learned by drawing various assumptions about the 3D plane. The hand-engineered features considered here can be any combination of colour, texture, location, shape or 3D geometry. In this approach, the pixels in an image are combined together to form superpixels and each of these superpixels is mapped to a label which can be any one of three geometric classes viz. ground plane, the surface above the ground or the sky. The image is segmented into multiple homogeneous regions which are geometric in nature and a model then tries to maximize the likelihood of homogeneity of a region. For a region to be homogeneous, all the superpixels belonging to this region must have same label.

Saxena *et al.* [11] used the Markov random field (MRF) to extract both local and global features from monocular RGB images to build a system called "Make3D" [12] for depth prediction. They have prepared their own dataset using a 3D laser scanner which outputs depth images. The input RGB image is segmented into small size patches and depth value is estimated for that small patch. Two class of feature vectors are used - absolute and relative. In the case of an absolute feature vector, patches are scaled at multiple resolutions to capture global image properties which help to estimate absolute depth. Relative feature vector uses the relationship between two adjoining patches to compute the relative depth for a patch. The strong correlation between adjacent patches impacts the depth of individual patches. Both absolute and relative feature vectors are modeled in the MRF which optimizes the distribution of depth values with respect to feature vector considered.

Instead of directly estimating the depth, Liu *et al.* [13] utilized the semantic labels of an image to construct the 3D geometry of the scene. They proposed two different approaches

using MRF to perform semantic segmentation based on pixels and super-pixels. These two approaches use apriori estimated depth values and semantic labels. The apriori estimated values represent the relationship between pixels or superpixels.

Ladicky *et al.* [14] predicted the likelihood depth value for a pixel by performing both semantic segmentation and depth estimation using a classification based approach. Depth estimation using a classifier is computationally efficient compared to depth estimation using a pixel-wise regressor. The same approach can be applied to obtain semantic information for a given scene.

Karsch *et al.* [15] in their approach, uses a matching technique on a predefined dataset containing RGBD images to extract a set of similar, look-alike images to that of the input image. These matching candidate images and their depth maps are warped with SIFT Flow [16] so that the resultant warped images matches the structure of the input RGB image. The warped image and depth pairs are then applied global depth optimization method to produce an approximate depth map. The model also gives good results in the case of a video where information obtained over the time is passed to the global depth optimization method to produce a per-frame depth map. It is based on the key idea that depth distributions of semantically similar scenes are also similar.

Konrad *et al.* [17] uses the input RGB image to find a set of nearest images and depth pairs from a dictionary of image and depth pairs. The depth maps from these nearest images are then fused together to form a final depth map which is later smoothened by using cross-bilateral filtering which removes invalid variations that may arise in the fused depth map. For the smoothened depth map, the RGB image is generated which along with the fused depth map forms the 3D image.

Liu *et al.* [18] used the Conditional Random Field (CRF) to realize depth estimation as an optimization problem with discrete and continuous potential variables with the assumption that the pixels having similar RGB properties possess similar depth values. The continuous potential variables map the relationship between a superpixel and its corresponding depth value. The discrete potential variables are required for mapping the complex correspondence between the neighboring superpixels. These two variables are modeled in CRF and depth is inferred from a single RGB image by applying particle belief propagation.

Recent advances in deep learning have led to the usage of various CNN architectures for depth estimation from single images. The deep CNN architectures usually follow the principle of transfer learning; a pre-trained CNN model, on a large dataset such as ImageNet [7], is used as a good initialization of weights upon which the domain specific dataset is trained to fine-tune the model in regard to the desired application.

Eigen *et al.* [19] used a two-stage architecture of different scales for depth estimation; in the first stage, a CNN model is applied to the input RGB image to extract coarse depth map. In the second stage, the coarse depth map is fed to the second network of CNN along with the input RGB image to produce a much refined depth map. The first coarse network

is employed to get a global overview of the objects present in the scene. This network consists of multiple convolution layers and the last few layers are fully-connected layers. Thus, the coarse network is not fully-convolutional and the presence of full-connected layers makes the entire image to be present in their view range. The second fine network which is fully-convolutional uses the output obtained in the coarse network to produce finer details of the depth map. This is done by using the localized object details present in an input RGB image to orient with the global structure obtained in the first stage of the network.

This work is further extended [20] to develop a three-stage CNN architecture where the first stage is pre-trained on either AlexNet [21] or VGGNet [22] to produce coarse details from the input RGB image which is refined further in later stages of the architecture. Apart from estimating depth, the same architecture is also being used to estimate surface normals and semantic labels. The output type is decided by the number of channels that are desired in the output image which is one in the case of the depth map. The first stage consists of state-of-the-art architectures which are popular in giving better performance in ImageNet [7] competitions. These architectures are fully adept in extracting features present in the input image and requires neither superpixels nor segmented patches of the input image.

Roy *et al.* [23] proposed a neural regression forest that employs both CNN and an ensemble of regression trees to produce continuous depth maps. Any pixel location from input RGB image is passed to the regression tree, where a CNN is employed in each of the nodes of the tree. The output from a CNN at a node can flow to either left or right subtree which is decided by Bernoulli probability. The output depth for an input pixel is estimated by combining the weighted path probabilities of the depth estimations made in each of the paths from the root to the leaf. This remodels the pixel-wise regression problem to a binary problem as each of the pixel values can move forward in either left or right subtree. Due to the association of CNN with each node in a tree, the output depth estimation for a path from the root to leaf resembles the computations being performed in a deep CNN.

Liu *et al.* [24] combined CNNs with probabilistic graphical models and a CRF loss layer to improve the quality of predicted depth images. The input RGB image is segregated into multiple superpixels and each one of these superpixels is fed to a network of only convolution layers in the case of unary potentials. For pairwise potentials, a set of similarities are considered for a pair of localized superpixels and it is fed to a network of fully-connected layers. The output obtained for both unary and pairwise part are passed to a CRF loss layer which optimizes negative log-likelihood to infer depth map.

Wang *et al.* [25] employed a joint CNN with hierarchical CRF layers to perform both semantic segmentation and depth prediction on input RGB images. The input image is first fed to a pre-trained CNN for jointly estimating depth map and semantic labels. They emphasized that as the depth map and semantic information of a scene have a strong association, allowing interactions between them improves the predicted depth values compared to CNN exclusively used for depth prediction. The depth map is fine-tuned further

by forwarding region-wise and pixel-wise potentials of an input image segment to a two-layer hierarchical CRF layer. Depth map and semantic information of an image are generated by performing inference through the hierarchical CRF layers.

Chakrabarti *et al.* [26] used a network of convolutional layers to produce distributions of depth derivatives having a different order, scale and orientation. This outputs several distributions for each position on the input image. These distributions are passed to a global optimization process which outputs uniform depth image that harmonizes with a group of local distributions.

Laina *et al.* [27] used berHu [28, 29] loss function to train a deeper CNN architecture combined with different sets of up-convolution and up-projection blocks to perform depth estimation. They used a variant of ResNet [5] which contains 50 parametric layers and the final network is devoid of any fully-connected layers. The up-projection blocks consists of multiple filters of different sizes and the output of these filters are interleaved together to produce the final feature map. Using filters in this way avoids zero multiplications that may arise if interleaving is not performed. This allows high-level features to be forwarded efficiently in the network. Their network when trained with berHu [28, 29] gives better qualitative output compared to L2 loss.

The hand-crafted methods are either parametric or scene-constrained, and hence cannot be deployed across varying environments. The CNN based models, on the other hand, obviously produce comparatively better result, however, still needs either multi-tier architecture or other means of post-processing to further fine-tune the result. The present work alleviates the above issue by presenting a fully convolutional deep architecture for improved depth estimation; a CNN model is chosen that can maximize the information flow along the deeper network withou any means of degradation issues. Additionally, an up-sampling strategy is proposed to improve the spatial resolution of the depth map so as to precisely recognize various objects present at discrete depth levels.

Chapter 3

Architectural Overview of Convolution Neural Networks

Convolution Neural Network revolutionized the pattern recognition approach applied on images by working on the raw pixels without requiring any hand-engineered features. It is designed to work on image volumes of different spatial sizes. It consists of various layers like convolution, pooling, activation, etc. Currently, many other types of layers like BatchNormalization [6], Dropout [4] are used for regularizing the parameters of convolution neural network. Different architectures based on these layers have successfully performed many computer vision tasks like object recognition, object localization, digit classification, etc. by beating the hand-engineered features used in pre-deep learning era by a huge margin. This chapter discusses some of the popular CNN architectures which won competitions on the widely popular ImageNet [7] dataset and explains key features present in it.

3.1 LeNet

One of the earliest introduction of deep learning was in 1998 when LeNet architecture was introduced for recognizing handwritten characters present in a document. This was published at a time when SVM and kernel learning were quite popular. The hand-engineered features (e.g. SIFT [30], HOG [31]) were dominant at that time for pattern recognition tasks. This architecture is trained on MNIST dataset which consists of 60000 training samples and 10000 test samples each of them being in grayscale with a spatial resolution of 28x28. The LeNet architecture is shown in Figure 3.1.

LeNet architecture consists of convolution, max-pooling and fully-connected layers as shown in Figure 3.1. The activation function used is either sigmoid or tanh for non-linearity in the model. Convolution layer at the shallower part of the architecture is used to extract features from the input volume which is followed by alternating sub-sampling layer to reduce the spatial resolution of the input volume. The final feature map from the sub-sampling layer is fed to regular fully-connected layer for classification of the digit present in the input image.

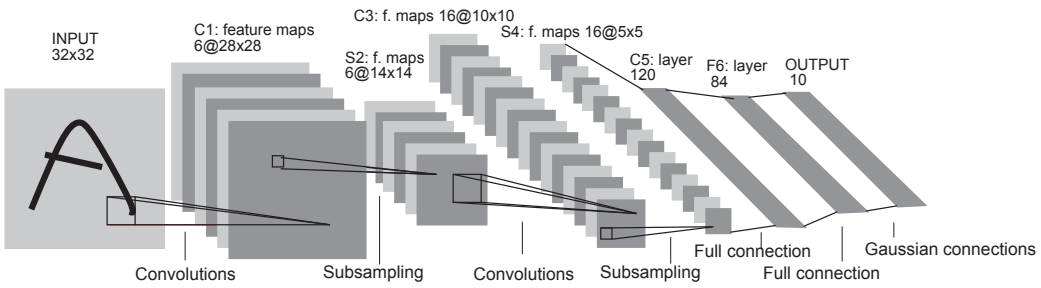


Figure 3.1: A forward pass in LeNet Architecture. **Source:** LeCun *et al.* [1].

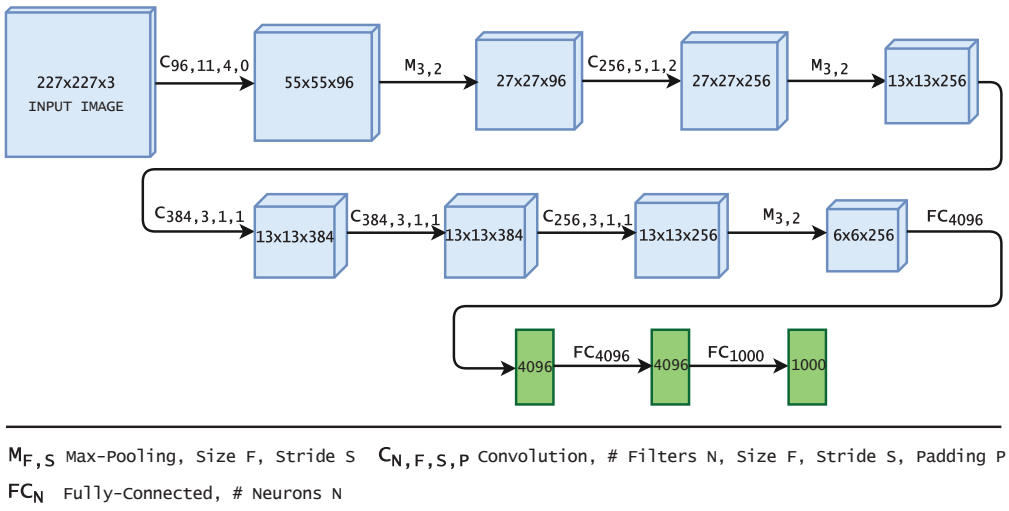


Figure 3.2: AlexNet Architecture, 2012 trained on ImageNet dataset.

3.2 AlexNet

In 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC) which consists of 1.2 million high-resolution images, AlexNet [21] architecture was the first deep convolutional neural network to be used for such computer vision task compared to shallower LeNet. AlexNet consists of 11 CNN layers of which 5 were convolution layers, 3 fully-connected layers and remaining were max-pooling layers. The architecture is trained on ImageNet dataset and the spatial resolution of the input image to the CNN model needs to be 227x227. AlexNet is a much deeper model compared to LeNet and it operated on a large volume of data compared to LeNet which operated on only 60000 training images. The architecture is shown in Figure 3.2.

It uses ReLU for non-linearity instead of either sigmoid or tanh as the latter activation functions leads to saturating gradients. The activation curve for these functions becomes constant for extremely low and high values of the input as seen in Figure 1.3.4 (a) and (b). When gradient gets saturated, the training gets slower as the gradient gets diminished while returning from the output layer to initial layers. This problem is popularly known as vanishing (exploding) gradient problem. ReLU avoids this problem by introducing

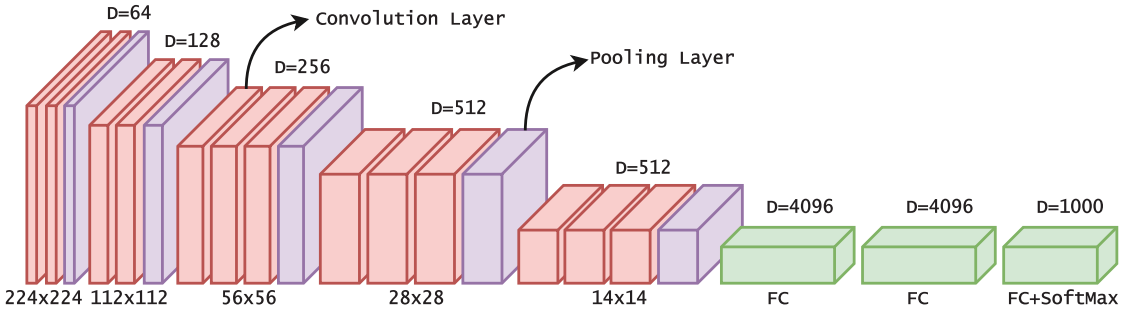


Figure 3.3: VGGNet Architecture, 2014 trained on ImageNet dataset.

non-saturating gradients leading to the faster training of the architecture. Originally, the architecture is executed over 2 GPUs in parallel where one GPU runs the layer-parts on the top part of the input volume and other GPU runs the layer-parts on the bottom. The convolution operations are implemented in GPU which gives 50 times more speed than the respective CPU implementation. A major addition to the architecture is the use of dropout for regularization. Dropout is a technique where neuron and its parametric connections to the next layer are randomly dropped. This results in different variants of the architecture during each iteration and thus incorporates robustness to variations in the input data.

3.3 VGGNet

VGGNet [22] architecture introduced in 2014 came first and second in ImageNet localization and classification task respectively. The main motive behind its inception is to explore the effects of much deeper layers in an architecture. VGGNet has two architectural variants consisting of 16 and 19 number of parametric layers which were deeper compared to prior CNN architectures. To reduce the number of parameters in the architecture, the filter size which was set to high (around 7x7) in previous architectures is reduced to smaller sizes (3x3) in all the convolution layers. In this architecture, the convolution layers are stacked on top of each other in increasing order of depth. The size of the volume of the feature map is reduced by means of max-pooling which are present in between a group of convolution layers. Zero-padding layers are used in this architecture to maintain the resolution of the output volume same as that of input volume. Dropout, as seen in AlexNet, is also used in this architecture for regularization. VGGNet-16 architecture is shown in Figure 3.3.

VGGNet emphasised that deeper architectures lead to better results. Compared to AlexNet, VGGNet employed simpler configurations, for example, 3x3 filters instead of 7x7 filters. VGGNet also used ReLU as a choice of activation function for faster training. The only drawback with VGGNet is the huge number of parameters present in the architecture which is majorly due to fully-connected layers present in it.

3.4 ResNet

The fundamental problems with VGGNet are slower training and large architecture weights (around 530MB).This can be attributed mainly due to the depth of the architecture and a large number of neurons present in fully-connected layers. This resulted in longer training time. ResNet [5] was proposed in 2015 where it beat all the state-of-the-art results across all the domains in the ImageNet competition.

The maximum number of parametric layers present in a convolutional neural network till ResNet was only 19 in one of the VGGNet variants. As network depth is very crucial for good results in deep learning based computer vision tasks, it is difficult to train neural networks that are much deeper than VGGNet. Apart from the difficulty in training, it also results in poor performance. This mainly happens due to degradation problem. For example, if we consider VGGNet-16 and copy over the layers to form an architecture with a number of layers more than 16, the new architecture produces poor results than the VGGNet-16 architecture. This degradation problem states that when the network depth starts increasing, the accuracy gets saturated and then degrades rapidly. The degradation problem, however, is not caused by overfitting as it is expected, but mainly due to the failure of the layers deep in the network to propagate the feature map produced by the previous layer.

ResNet addressed the degradation problem by using smaller micro-architecture modules as building blocks to create deep residual learning framework. Compared to previous CNN architectures which were sequential in terms of the layer connection, ResNet used small residual modules to build a bigger network. The sole reason for going with residual modules is to make the parametric layers fit a residual mapping rather than fit an underlying mapping between layers as a whole. It is observed that for a deeper network it is easier to train the residual mapping between parametric layers than the original mapping with no residuals. This idea of residual mapping is done with the help of a shortcut connection to skip over some of the parametric layers. The input feature map is added directly to the feature map obtained after forwarding through a number of convolution layers (either 2 or 3) in a block to produce output feature map as shown in Figure 3.4. For example, if $H_l\left(.\right)$ is a composite function consisting of weight layers and other non-weight layers in a ResNet block where l is the layer index, then for an input volume x_{l-1} at the $(l+1)^{th}$ layer, the output volume is given by equation below.

$$x_l = H_l\left(x_{l-1}\right) + x_{l-1} \tag{3.1}$$

To form deeper networks, layers are added as identity mappings such that a deeper model should not have training error greater than its corresponding shallower version. When deeper layers are used in a model, the gradient computed at the final layer gets diminished by the time it reaches initial layers of the networks during backpropagation. Having a

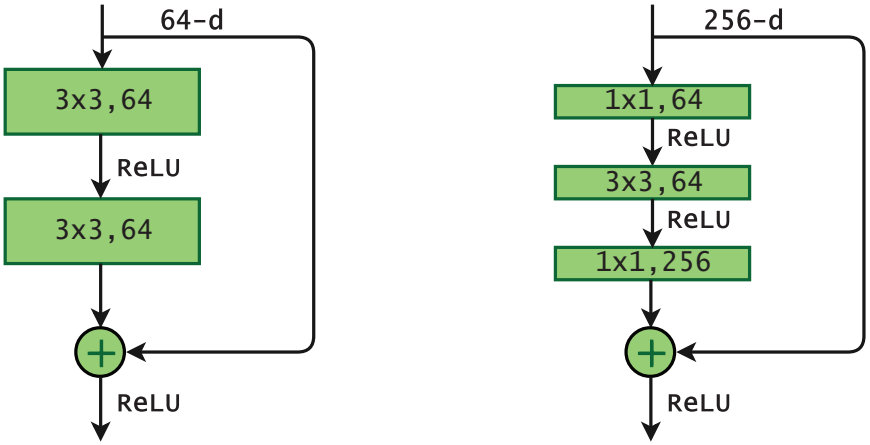


Figure 3.4: Building Blocks of ResNet, 2015 trained on ImageNet dataset. Left: with 2 convolution layers. Right: with 2 convolution layers including a bottleneck layer.

shortcut connection over some of the parametric layers helps in the efficient flow of the gradients during backpropagation as it can flow directly to a layer backwards by skipping these parametric layers. When shortcut connections are not used, the flow of gradients in backpropagation can cause trouble due to non-linear activation functions involved. A new regularization layer called Batch-Normalization layer is used in the residual modules which are generally preferred over Dropout for better performance. Batch-Normalization layer normalizes the input batch before forwarding it to the next layer so that the batches to each of these layers corresponds to similar distribution. If Batch-Normalization is not used, then with the change of parameter values, the distribution of input values to a layer also changes and the network takes longer to train. Even though ResNet can be much deeper compared to VGGNet, the size of the architecture is substantially smaller due to the presence of global average pooling layer which produces output directly instead of fully-connected layers which contain a large number of parameters. The version of ResNet which won the ImageNet 2015 competition consisted of 152 layers. A unit residual block in ResNet consists of either 2 convolution layers or 3 convolution layers with a bottleneck convolution layer as shown in Figure 3.4.

Recent modifications to ResNet architecture have shown promising results for ImageNet competitions. In one of the changes, the layers present in a building block of ResNet are reordered such that the activation and regularization layer precede the convolution layer. This setting of layers before convolution layer is known as pre-activation which improves the performance achieved by original ResNet modules as shown in Figure 3.5. In another change to regular ResNet, high dimensional convolution layers are used for building a ResNet block. This results in the increase of width and decrease of depth of a residual network. This way, a wide residual network can achieve comparable performance to that of a much deeper residual network and the issues with training deeper residual networks will not occur. The motivation

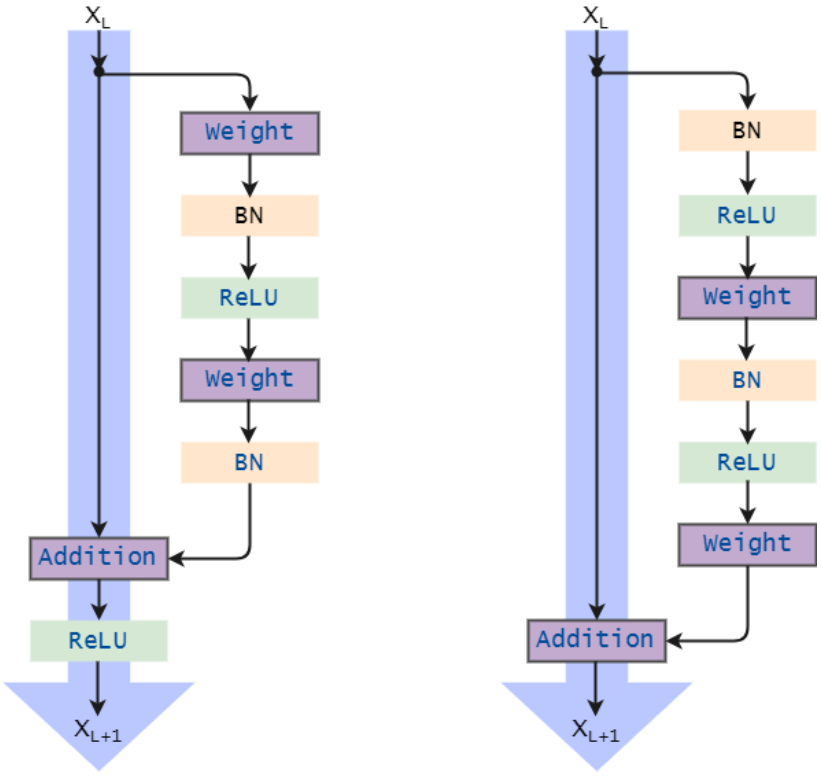


Figure 3.5: ResNet modules Left: without pre-activation. Right: with pre-activation.

behind the inception of a wide residual network is that to achieve a significant performance improvement in a residual network, the number of layers needs to be doubled. This leads to the problem of diminishing feature reuse where the features computed at the initial layers of the residual network fades away when it reaches the deeper layers of the network. This happens primarily due to a number of parametric multiplication with the input feature map at each of the parametric layers resulting in the loss of actual feature map. Wide residual network tackles these problems with a less deep network than the conventional ResNet. For example, wide residual network with 50-layer achieves better performance than 152-layer residual network with 3 times fewer layers and is significantly faster.

3.5 DenseNet

DenseNet was introduced in late 2016 and it introduces shortcut connections between every pair of layers in a feed-forward manner rather than between successive layers as seen in ResNet. Recent works on convolutional neural network showed that if initial layers are connected to the deeper layers near the output with shortcut connections, then it is efficient and takes less time to train. Traditional convolutional neural networks with L layers have L connections which are sequential in nature, whereas DenseNet with L layers has $L\,(L+1)\,/2$ direct connections between every pair of layers. DenseNet concatenates all the previous feature maps from 0 to $l-1$ layers to be used as an input to l^{th} layer as given in the

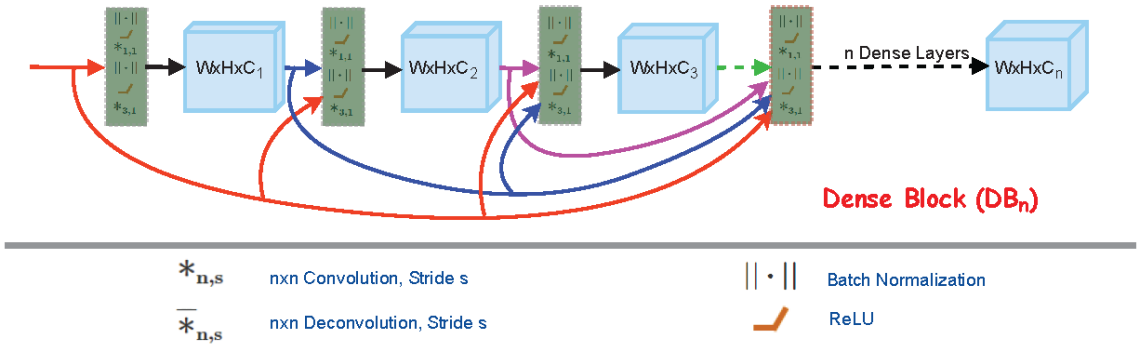


Figure 3.6: An dense block with n dense layers. Each layer takes all preceding feature maps as input.

equation 3.2. This is in different from ResNet where input feature map is added to the output of the input feature map through parametric layers in a residual block.

$$x_l = H_l\left([x_0, x_1, ..., x_{l-1}]\right) \tag{3.2}$$

All the features before l^{th} layer are effectively preserved while explicitly differentiating between the features newly added. This concatenation of feature maps learned by different layers increases the variation in the input for subsequent layers and improves efficiency. This is the major difference compared to ResNet which makes DenseNet simpler and more efficient. Figure 3.6 shows an example of dense connectivity between dense layers present in a dense block.

The concatenation operation is not suitable for variable feature map sizes. Pooling operation helps to change the size of feature maps. To incorporate pooling in DenseNet, the network is broken down into multiple densely connected dense blocks. Layers between dense blocks are called transition layers which perform convolution and pooling. To prevent the network from becoming too wide and to improve parameter efficiency, a hyper-parameter k is used as the growth rate of the network. It controls how much new information each layer contributes to the global state. This global state can be accessed from everywhere in the network. DenseNet as a sequence of dense block and transition layers.

If the composite function H_l is assumed to produce k feature maps as output, then the number of feature maps $N_{feature-maps}$ at l^{th} layer is given by the equation 3.3.

$$N_{feature-maps} = k \times (l-1) + k_0 \tag{3.3}$$

There are several advantages of using DenseNet. The vanishing gradient problem is alleviated as each layer has access to every other layer in the network, the gradients computed in backpropagation stage can propagate to initial layers without getting diminished. DenseNet also encourages feature re-use and feature propagation which was a problem in the case of ResNet with very large (> 1000) number of layers. This is due to the dense

connectivity of layers in DenseNet where the features computed at earlier layers can be used in deeper layers without any loss. The feature map of all the previous layers are concatenated and thus preserved for use as input to the next layer. DenseNet has better parameter efficiency and as the flow of information and gradient in the network is better than compared to previous architectures, they are easier to train. There are different variations of DenseNet available either with a bottleneck or compression layer which further improves the performance of DenseNet.

Chapter 4

Depth Estimation using Fully Convolutional Architecture

4.1 Proposed Methodology

This section presents a fully convolutional architecture for monocular depth estimation using single RGB image. The proposed framework is a sequence of two modules. A deep convolutional network is first employed to extract the feature map. It can be realized that the output map of a deep CNN usually possesses very low spatial resolution as compared to that of the input resolution. It works well for any classification task, where the low resolution map is usually fed into either a fully-connected layer or an equivalent convolutional layer to predict the desired object class. In contrast to this, a regression task, such as depth estimation, requires at-least a minimum higher resolution output, where the objects at different depth should be precisely distinguished. Accordingly, we suggest a simple yet efficient mechanism to up-sample the low-resolution images with minimal parameter overhead.

A number of deeper architectures have been reported over the last few years. We embed the Densely Connected Convolutional Networks [8], abbreviated as DenseNet, to learn the feature representation. Subsequently, few additional "deconvolution layers" are incorporated to learn the up-samping within the unified framework. The architectural diagram of our model is depicted in Figure 4.1. All the steps of the proposed architecture are enumerated in the subsequent paragraphs.

DenseNet. The DenseNet model comprises a number of intermediate layers such that each layer accepts inputs from all its preceding layers and forwards its output to all the subsequent layers of the network. Mathematically, the p^{th} layer accepts the feature maps of all its preceding layers and computes a non-linear mapping function $H\left(p\right)$, given by: $x_s = H_s\left([x_0, x_1, \cdots, x_{p-1}]\right)$, where $[x_0, x_1, \cdots, x_{p-1}]$ represents the concatenation of the feature maps generated by all its preceding layers $0,\ 1, \cdots, p-1$. The DenseNet model is arranged as a sequence of multiple dense-blocks as shown in Figure 4.1. A dense block computes a number of composite mapping functions. Each of these functions performs

three consecutive operations: batch normalization followed by a ReLU followed by a 3×3 convolution. Each 3×3 convolutional layer produces a fixed k number of output feature maps, termed as the growth rate of the network. However, it accepts many more inputs owing to the dense connectivity of the model. Therefore, a 1×1 bottleneck convolution layer is employed prior to to the 3×3 convolution that reduces the input to $4 \times k$ feature maps only. All the output feature maps within a dense block possess the same spatial resolution. To facilitate more compactness, the concatenated feature map of one dense block is fed into its next dense block via a transition layer that reduces the spatial resolution of the concatenated input map by half of its original size; a transition layer performs a batch normalization, a 1×1 convolution and a 2×2 average pooling. An architecture with such dense connectivity is chosen to facilitate direct connectivity among all the layers and thereby maximizes the information flow along the network. The original paper of DenseNet [8] prepare four different models out of which the DenseNet-161 (K=48) performs superior as compared to others for the ImageNet classification task.

The present work also adopts the DenseNet-161 model for the regression task as shown in Figure 4.1; it takes an input size of $320 \times 240 \times 3$ (width $\times$ height $\times$ # input channels), forwards it though the CNN having four dense blocks, and yields an output size of $10 \times 7 \times 2208$ (width $\times$ height $\times$ # feature maps).

Deconvolution Blocks. The low resolution feature map, obtained by the CNN, needs to be up-sampled to precisely identify the objects at discrete depth levels. An easiest way is to incorporate a fully-connected layer to the end of the last convolutional layer of the DenseNet. The present simulation, if embed a fully connected layer, requires more than 3.4 billion parameters to connect a $10 \times 8 \times 2208$ feature map to a 175×127 fc layer. Moreover, the fully connected layer cannot well exploit the local correlation within the neighborhood pixels. In contrast to this, we concatenate a network of deconvolution layers towards the end of the DenseNet to learn the up-sampling within a unified network. Each of the deconvolution layer performs three sequential operations in a pre-activated arrangement: $\langle$`Batch Normalization-ReLU-Convolution`$\rangle$. ResNets [32] has already shown through experimentation that such a pre-activated arrangement of convolution performs well across a DCNN network. Even, the DenseNet model also performs the batch normalization and ReLU prior to each 3×3 and 1×1 convolution. In short, the proposed architecture first applies a sequence of BN-ReLU-Conv operations, via DenseNet, to yield a low resolution feature map, and then again append a sequence of BN-ReLU-Conv operations to upsample the feature map size within the same network. The proposed architecture, as shown in Figure 4.1, first applies a bottleneck layer (1×1 convolution) that reduces the number of feature maps from 2208 to 512, while keeping the same spatial resolution. Then, a total of four deconvolution blocks (BN-ReLU-Conv$_{5\times5,\texttt{stride = 2}}$) is appended to produce the desired depth map. It can be noted that more number of deconvolution blocks may be

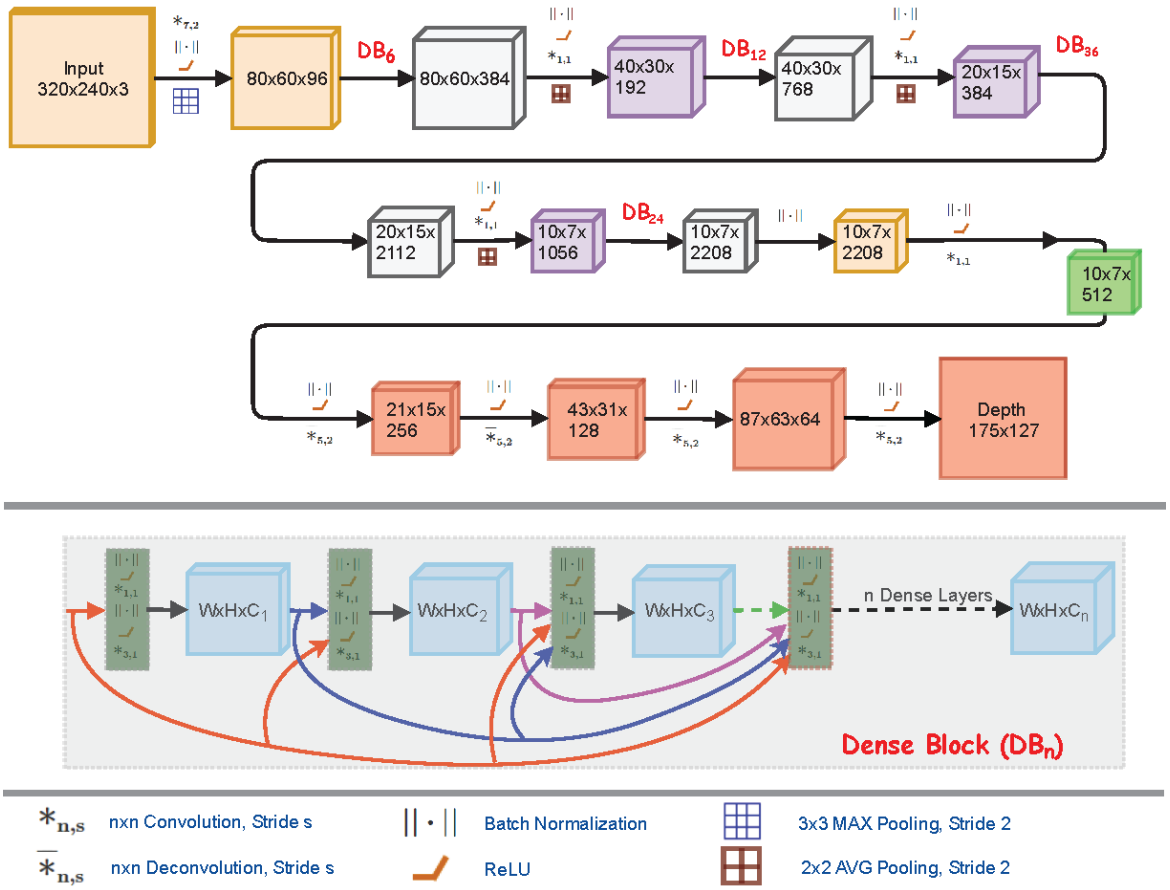


Figure 4.1: Proposed fully convolutional architecture. Stage (1): DenseNet-161 model accepts input image of size $320 \times 240 \times 3$ and produces a feature map of size $10 \times 7 \times 2208$. Stage (2): a bootleneck layer reduces the number of feature maps: $10 \times 7 \times 2208$ to $10 \times 7 \times 512$ (Green box). Stage (3): a sequence of four deconvolution layer up-samples the spatial resolution of feature map : $10 \times 7 \times 512$ to 175×127 (Deconvolution outputs in red boxes). Bottom row depicts the the schematic diagram of a Dense Block; DB_n represents a Dense Block with n layers.

required depending on the size of the depth map. The present simulation requires only 5.4 million parameters, to be learned, between a $10 \times 8 \times 2208$ DenseNet output map and a 175×127 depth map.

Loss function. The choice of the loss function plays a crucial role in any CNN based optimization task. We simulate the proposed model in equation with minimizing two loss functions separately that have been extensively used in depth prediction; (i) RMSE loss, (ii) Reverse Huber (berHu loss).

The RMSE loss minimizes the squared-root euclidean norm between the predicted depth map ($\hat{y}$) and the ground-truth map (y), given by,

$$RMSE\left(\hat{y} - y\right) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{y} - y\right)^2} \tag{4.1}$$

where n represents the number of input pixels in one mini-batch over an image.

The berHu loss poses a good trade-off between the L_1 norm and L_2 norm so as to provide higher weights to pixels having higher residual. The berHu loss [28, 29] between the predicted depth map and the ground-truth map can be given as,

$$berHu\left(\hat{y}-y\right)=\begin{cases}L_1\left(\hat{y}-y\right) & \text{if } \left(\hat{y}-y\right)\in[-t,\,t]\\ \frac{1}{2t}\times L_2\left(\hat{y}-y\right)+t^2 & \text{otherwise}\end{cases} \tag{4.2}$$

$$L_1\left(\hat{y}-y\right)=\left|\hat{y}-y\right| \tag{4.3}$$

$$L_2\left(\hat{y}-y\right)=\left\|\hat{y}-y\right\|_2^2 \tag{4.4}$$

It can be observed that Equation 4.2 is continuous, and moreover, first order differentiable at threshold t that decides the switching between L_1 to L_2. In each step of gradient descent, t is set as the 20% of the maximal per-batch error, given by, $t=0.2\max_i\left(\left|\hat{y}_i-y_i\right|\right), i=1,2,\cdots,n$.

4.2 Experimental Results

This section provides a detailed analysis of the experiments carried out for the proposed architecture. Our model is implemented using PyTorch[1] and trained on a system having dual Quadro K2200 GPU with 4GB memory. The first stage of our architecture takes the pre-trained DenseNet-161 model, on ImageNet dataset [7], as initialization. The second stage, having one bottleneck convolutional layer and four deconvolution blocks are initialized with weights taken from a normal distribution with 0 mean and 0.02 variance.

NYU Depth Dataset. The proposed model is evaluated on the standard NYU Depth v2 [9] RGB-D image dataset. It comprises images of multiple indoor scenes taken from a Microsoft Kinect depth camera. Moreover, it contains both labeled and raw images for evaluation. The trainin ang test set, of the raw dataset, comprises 249 and 215 indoor scenes respectively; each scene contains a number of RGB images as well as its corresponding depth maps.

We consider a subset of the raw dataset that further needs to be preprocessed in accordance with the depth prediction task. A kinect camera can process a maximum depth of 10 meters, and hence all the ground-truth depth maps are normalized in the range of 0 to 10. Besides, the spatial resolution of the RGB images (640×480) is reduced to 320×240 for our model evaluation. We perform data augmentation to increase the number of training samples; in particular, we take a total of 5900 training samples to train our model. Prior to training, all the input images are normalized with the mean and standard deviation of the ImageNet dataset.

[1] https://github.com/pytorch/pytorch

Data Augmentation. We resort to standard data augmentation techniques to increase the number of training samples. The input RGB images and corresponding target depth images are subjected to following data augmentation techniques —

- **Scaling:** images are scaled randomly with a factor that lies between 1 to 1.5.
- **Rotation:** images are rotated randomly with an angle that lies between $-5°$ to $+5°$.
- **Flipping:** images are flipped randomly with a probability of 0.5.

Although the augmentation is done randomly, it may be noted that the augmentation value for a specific RGB image and its corresponding depth image is kept same. Data augmentation helps the model to learn efficiently by adding different variations of the same training sample.

Architecture Evaluation: Our model is trained using Stochastic Gradient Descent [33] as the optimization method and berHu, RMSE as the loss functions (Section 4.1). The training for both the loss functions is done separately and the quantitative results are shown in Table 4.1. The training is performed with a batch-size of 5 and initial learning rate is set to 0.001. The learning rate is decreased by a factor of 5, when the loss becomes more or remains same for consecutive 5 epochs. It has been observed that berHu loss produces better qualitative predictions as compared to RMSE loss. This can be attributed to the fact that berHu is observed to give better convergence in comparison to RMSE. Also, in terms of training loss, berHu outperforms RMSE. This can be better visualized in Figure 4.2, which represents the epoch-wise loss values for both the loss functions. The resolution for the predicted depth image of our proposed model is taken as 175×127. Deconvolution network containing 4 deconvolution layers enhances the depth image from low resolution to high resolution. The prediction is done on the standard labeled test dataset containing 654 images. The predicted depth images are then compared with ground truth images for quantitative evaluation using standard error metrics defined in previous depth estimation works [14, 19, 20, 24, 34]. Table 4.1 reports our model performance as compared to other state-of-the-art models. It can be observed that our model outperforms existing state-of-the-art techniques with fewer training samples. The predicted depth images obtained using berHu loss function is delineated in Figure 4.3. It can be noticed that our model exhibits remarkable visual quality for the predicted depth images.

The proposed architechture, including the DenseNet and deconvolution blocks, has a total of 167 parametric layers. Also, our model contains very less parameters as compared to state-of-the-art models. The number of parameters in our model (31 million) is far less as comapred to Eigen *et al.* [20] (218 million).

NYU Depth V2	lower is better				higher is better		
	rel	rms	rms(log)	log_{10}	δ_1	δ_2	δ_3
Karsch *et al.* [15]	0.374	1.12	-	0.134	-	-	-
Ladicky *et al.* [14]	-	-	-	-	0.542	0.829	0.941
Liu *et al.* [18]	0.335	1.06	-	0.127	-	-	-
Li *et al.* [34]	0.232	0.821	-	0.094	0.621	0.886	0.968
Liu *et al.* [24]	0.230	0.824	-	0.095	0.614	0.883	0.971
Wang *et al.* [25]	0.220	0.745	0.262	0.094	0.605	0.890	0.970
Eigen *et al.* [19]	0.215	0.907	0.285	-	0.611	0.887	0.971
Roy and Todorovic [23]	0.187	0.744	-	0.078	-	-	-
Eigen and Fergus [20]	0.158	0.641	0.214	-	0.769	**0.950**	**0.988**
Proposed method (RMSE)	0.159	**0.549**	0.213	0.064	0.791	0.946	0.984
Proposed method (berHu)	**0.153**	**0.549**	**0.208**	**0.062**	**0.799**	**0.950**	0.985

Table 4.1: Comparative analysis of various methods on NYU Depth V2 dataset.

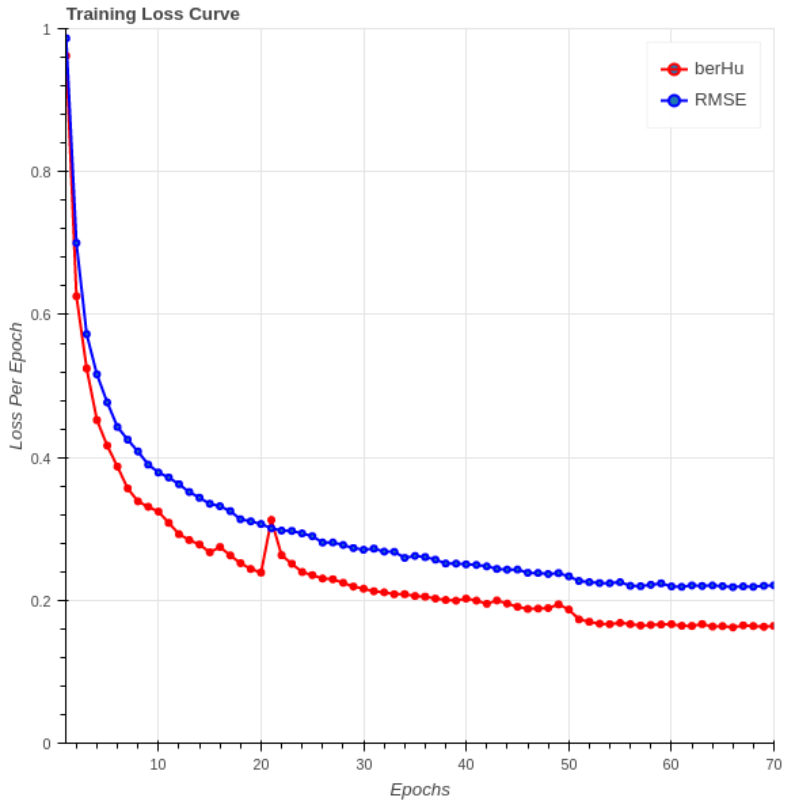


Figure 4.2: Training Loss Curve for RMSE and berHu loss functions.

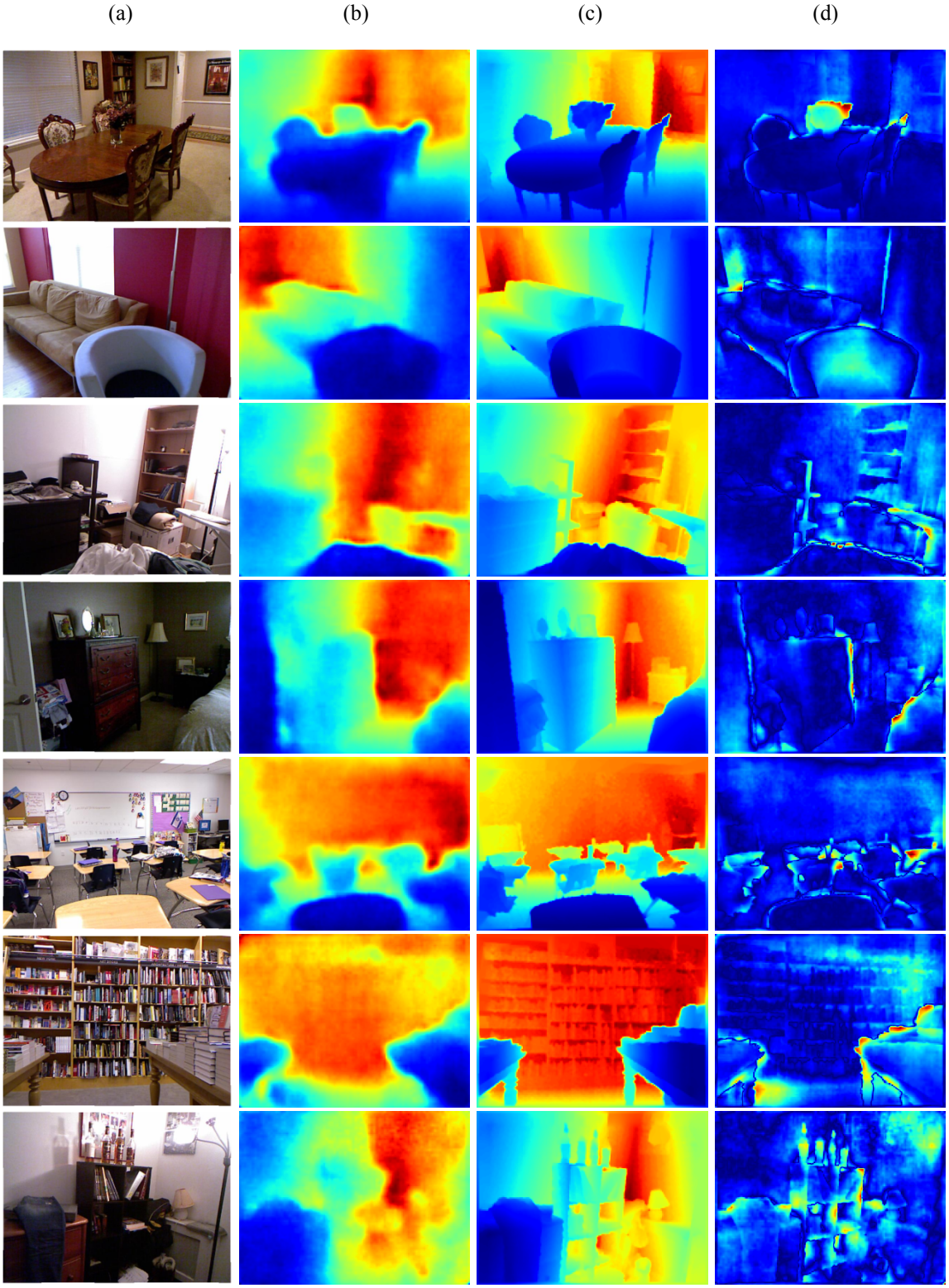


Figure 4.3: Prediction results of sample images by our proposed architecture on NYU Depth V2 dataset. The figure shows (a) input image (b) predicted depth image (c) ground depth image (d) absolute error map. For better comparison, all the colormaps are scaled equally.

Chapter 5

Conclusion

This chapter summarizes the work that has been done for the thesis. The primary focus of the work is to devise a deep fully-convolutional architecture based on transfer learning approach to estimate depth from a monocular image. An architecture based on DenseNet [8] is proposed which is a sequence of two modules. The first module consists of DenseNet architecture pre-trained on ImageNet [7] dataset where the final full-connected layer is removed. The second module consists of a bottleneck layer and four deconvolution blocks joined to the output of the first module. DenseNet is preferred over earlier CNN architectures such as ResNet [5], VGGNet [22], etc. DenseNet requires fewer parameters for optimization and supports feature-reuse, feature-propagation. The dense connectivity of layers in a dense block speeds up the learning process. Earlier CNN architectures required more than 100K for convergence in depth estimation task for NYU Depth V2 dataset. The proposed architecture requires only 5900 of training samples for producing low RMSE value than the state-of-the-art architecture. These training samples belong to different scenes provided in NYU Depth V2 dataset. Random augmentations are applied to incorporate variations in the training data. Deconvolution blocks are used in the proposed architecture to generate the final depth map of the input image. It consists of batch-normalization, ReLU and deconvolution layers in order. The feature map generated by the first module has a low spatial resolution which needs to be enhanced for comparison with ground truth values and inferring depth in real-time scenarios. The order of layers in a deconvolution block follows a pre-activation style which gives better performance compared to no pre-activation style as observed in He *et al.* [32]. Experimental results have shown faster convergence with the pre-activation style of layer ordering. Also, instead of simply upscaling the low-resolution feature map, 5x5 filters are used to learn the mapping from low to high-resolution feature map. The proposed architecture is optimized on both berHu and RMSE as an objective function. Using berHu over RMSE leads to faster convergence as the L1 distance between two pixels is weighted heavily instead of L2 distance as seen in RMSE. The proposed architecture does not use any kind of pre-processing like CRF on the resultant depth image.

Scope for Further Research

In this thesis, depth is estimated from monocular images. The depth image contains the relative position of the objects from a viewpoint with 10m as the maximum distance. The number of training samples may be increased with augmentations to further strengthen the capacity of the proposed architecture. Outdoor depth datasets such as Make3D [12] and KITTI [35] may be used for training the architecture. Further research may be carried out by using depth image in a Simultaneous Localization and Mapping (SLAM) framework for generating the 3D structure of an environment on the fly. Obstacle detection is one another task where depth image may be used to infer the location of the objects in the scene.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[3] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[4] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[8] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[9] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," *Computer Vision–ECCV 2012*, pp. 746–760, 2012.

[10] D. Hoiem, A. A. Efros, and M. Hebert, "Geometric context from a single image," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 1. IEEE, 2005, pp. 654–661.

[11] A. Saxena, S. H. Chung, and A. Y. Ng, "Learning depth from single monocular images," in *NIPS*, vol. 18, 2005, pp. 1–8.

[12] A. Saxena, M. Sun, and A. Y. Ng, "Make3d: Learning 3d scene structure from a single still image," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 824–840, 2009.

[13] B. Liu, S. Gould, and D. Koller, "Single image depth estimation from predicted semantic labels," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1253–1260.

[14] L. Ladicky, J. Shi, and M. Pollefeys, "Pulling things out of perspective," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 89–96.

[15] K. Karsch, C. Liu, and S. Kang, "Depth extraction from video using non-parametric sampling," *Computer Vision–ECCV 2012*, pp. 775–788, 2012.

[16] C. Liu, J. Yuen, and A. Torralba, "Sift flow: Dense correspondence across scenes and its applications," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 978–994, 2011.

[17] J. Konrad, M. Wang, and P. Ishwar, "2d-to-3d image conversion by learning depth from examples," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*. IEEE, 2012, pp. 16–22.

[18] M. Liu, M. Salzmann, and X. He, "Discrete-continuous depth estimation from a single image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 716–723.

[19] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in neural information processing systems*, 2014, pp. 2366–2374.

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

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[23] A. Roy and S. Todorovic, "Monocular depth estimation using neural regression forest," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5506–5514.

[24] F. Liu, C. Shen, and G. Lin, "Deep convolutional neural fields for depth estimation from a single image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5162–5170.

[25] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. L. Yuille, "Towards unified depth and semantic prediction from a single image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2800–2809.

[26] A. Chakrabarti, J. Shao, and G. Shakhnarovich, "Depth from a single image by harmonizing overcomplete local network predictions," in *Advances in Neural Information Processing Systems*, 2016, pp. 2658–2666.

[27] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *3D Vision (3DV), 2016 Fourth International Conference on*. IEEE, 2016, pp. 239–248.

[28] A. B. Owen, "A robust hybrid of lasso and ridge regression," *Contemporary Mathematics*, vol. 443, pp. 59–72, 2007.

[29] L. Zwald and S. Lambert-Lacroix, "The berhu penalty and the grouped effect," *arXiv preprint arXiv:1207.6868*, 2012.

[30] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[31] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[33] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[34] B. Li, C. Shen, Y. Dai, A. van den Hengel, and M. He, "Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1119–1127.

[35] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[0]This reference format follows ASME style. You are advised to follow one reference format of any dominant journal of your field.