Depth map generation using a pair of stereo images

Summer Research Fellowship Report

Varun Kumar
Registration No. ENGT151

July 15, 2015

ABSTRACT

Depth estimation of the scene from images taken from multiple viewpoints is a very old yet unsolved problem. Depth estimation has various applications such as autonomous navigation in the filed of robotics, in the field of scene understanding, activity recognition etc. Various methods ranging from the use of hand engineered features such as SIFT to data driven approaches have been applied to solve this problem but the problem still remains challenging till date with the major challenges being the specular highlights and texture less surfaces. In this project an attempt has been made to solve this problem using a data driven approach, wherein a novel architecture based on Convolutional Neural Network is proposed and applied on 2 images of the same scene taken from cameras placed a small distance apart. Two Convolutional Neural Nets are trained parallely which is assumed to extract good features from images and then these features are concatenated to finally predict the depth map of the scene. The reason behind choosing CNNs is that they had recently been used in problems related to images and have given state of the art results. Therefore it can be concluded that they are the best feature extractors today. The whole architecture was made using a framework known as torch7 which is made on scripting language lua. This framework provides inbuilt modules on convolution and other functions which are important in this project. The model needs a high performance computer with a minimum of 15GB RAM to start execution

1

INTRODUCTION

Stereo Vision deals with having 2 cameras kept along different horizontal positions looking at the same scene and capturing the images. The goal is to compute the disparity(d) for each pixel in the left image. Disparity refers to the difference in horizontal location of an object in the left and right image; an object at position (x, y) in the left image will appear at position $(x - d, y)$ in the right image. Knowing the disparity d of an object, we can compute its depth z (i.e. the distance from the object to the camera) by using the following relation:

$$z = \frac{fB}{d} \tag{1}$$

Where f is the focal length of the camera and B is the interocular distance i.e. distance between two camera centers.
Hence the problem of depth computation boils down to finding the specific pixel of the left image in the right image. This is known as correspondence problem. It is a fairly difficult problem to solve due to the presence of surfaces which are textureless or when there are reflections on the surface. According to the taxonomy of [16] a typical 2 fame stereo matching consists of 4 steps: (1) Matching cost computation (2) Cost aggregation (3) Optimization and (4) disparity refinement. The approached used in my project tries to combine the first 3 steps and compute them using just the image samples. The refinement could be then taken care of by using algorithms such as CRF or MRF.
Estimating depth is an important component of understanding geometric relations within a scene. In turn, such relations help provide richer representations of objects and their environment, often leading to improvements in existing recognition tasks, as well as enabling many further applications such as 3D modeling, physics and support models, robotics, and potentially reasoning about occlusions. The most important criteria for a good stereo algorithm is that it should be computationally efficient and the depth map that it produces is not sparse. In this project I have tried to accomplish that. Recently the Convolutional Neural Networks have shown promising results on various vision related tasks. This is due to the availability of high performance GPUs. Earlier the networks that took months to be trained are now able to get trained in a matter of days. [19] tried to visu-

alize each layer of the Convolutional Neural Network and showed that the first layer learns the filters that learn to detect edge strokes. Next layer learns the filters that detects basic shapes composed of the previously learned edges. As we move further down the layers, they showed that we learn parts of the images such as if the network is trained to learn faces, the later layer will contain filters that will detect the presence of eyes, mouth etc. Therefore nowadays they are used as feature extractors. In this project CNNs are used as feature extractors. The extracted features are then joined along one dimension and then are passed through a fully connected neural network for depth map prediction. The specification of the network will be discussed later.

To implement the architecture I have used the torch7 library. It is widely used in the research community for developing deep neural networks. It as got pre-built modules for 2D convolution, various loss functions such as Mean Squared Error, Negative Log Likelihood etc. and various activation functions such as Sigmoid, Tanh, Rectified Linear Unit etc. Construction of deep neural networks i.e. neural networks with many layers is quite intuitive with this library. To create a network we have to keep on adding the modules one after another. The backpropogation algorithm is also implemented in it. To train such models, torch7 has also provided various optimization algorithms such as LBFGS, Stochastic gradient descent etc. Details will be discussed later.

2

RELATED WORK

As one of the oldest and most fundamental problems in computer vision [10], computational stereo has witnessed great progress making it nowadays an attractive alternative to costly laser scanners for outdoor environment perception, e.g., in autonomous driving [3, 6]. The recent success of stereo methods can be attributed to the development of benchmarks such as Middlebury [16]. More recently, the KITTI benchmark [5] has pushed limits even further by providing a larger and more realistic stereo dataset with meaningful object classes and ground truth annotations. Local stereo methods [4, 9, 11] often fail in challenging scenarios as they suffer from matching ambiguities in weakly textured, saturated or reflective regions. Thus, recent efforts mainly focus on global methods [1, 8, 14] which are able to overcome some of these problems by imposing smoothness constraints between adjacent pixels or superpixels. Saxena et al. [15] proposed to directly integrate depth-from-appearance constraints into the data term. In contrast, Ladicky et al. [13] model stereo estimation and semantic segmentation jointly by learning the dependency between height over ground and the semantic class. Wei et al. [17] follow a data-driven approach which directly transfers disparity information from regions with similar appearance in the training data using SIFT flow Unfortunately, the nature of interaction in these models is very local and thus cannot constrain large ambiguous regions well enough. Konda et al. [12] suggested that depth cues can be learnt from data itself. They used a model based on 'Synchrony autoencoder' to learn depth /features/weights in an unsupervised way. They used these weights to extract depth from a stereo image pair. The depth map generated was sparse and failed in textureless regions. Zbontar et al. [18] have trained a Convolutional Neural Network with image patches to predict how well two patches match (Correspondence problem) to create a coarse depthmap. This coarse depthmap is refined afterwards by cross-based cost aggregation and semiglobal matching. Left-Right consistency check is done for occluded regions. they performed poorly where specular highlight is present. Guney et al. [7] have improved the performance on specular surfaces by detecting cars in the scene. They then optimized Conditional Random Field energy function for depth estimation. For cars, they matched the resulting depthmap with the depthmap artificially generated by constructing

the 3D CAD model of cars and adjusted the depth map by generating score based on level of agreement. Their algorithm takes 265 sec for calculating disparity of one stereo pair. Optimization based depth estimation tends to be slow. Moreover modelling of 'n' number objects that may appear is tough in realtime. Eigen et al. [2] took a single image for depth map prediction. They trained a CNN with full images along with the corresponding Depth map as Ground Truth. They used 2 CNNs. First one predicts a coarse depthmap Second one uses both the images and the coarse depthmap generated previously to predict the refined depthmap. It fails at specular surfaces.

3

CONVOLUTIONAL NEURAL NETWORKS

This chapter explains the underlying components and various terminologies used in this project. This chapters builds the vocabulary which helps the reader understand the later sections of methodology and results.

3.1 NEURAL NETWORKS

This section describes the basic architecture of Neural Network. A traditional neural network is trained in a supervised fashion. We have a set of labelled training samples $\{x^{(1)}, x^{(2)}, \ldots\}$, where $x^{(i)} \in \Re^n$. And then there are class labels $y^{(i)}$. Figure 4 is the architecture of basic Neural Network that makes an effort to learn hypothesis $h_{W,b}(x) \approx y$ which is the output after after the feed forward phase. Here W is given by,

$$W = \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\} \tag{2}$$

Where, $W^{(1)}$ and $b^{(1)}$ constitute the weight matrix and the bias vector respectively between the input layer and the hidden layer. W^l_{ij} denotes the weight between the j^{th} neuron of layer l and i^{th} neuron of layer $l+1$ and b^l_i denotes the bias associated with the i^{th} neuron of layer $l+1$. Each neuron activation denoted by a^j_i signifies the activation of i^{th} neuron in the j^{th} layer of the network. The activation value for a neuron shown below in figure 1 is calculated in equation 3.

$$h_{W,b}(x) = f\left(\sum_{i=1}^{n} W_i x_i + b\right) \tag{3}$$

where, W_i is the weight associated with x_i and b is called bias and it the weight associated with the '+1' input. The $f(.)$ defined above is called activation function. Some of the widely used activation functions are:

• **Sigmoid**: Sigmoid function $f : \Re \mapsto \Re$ is given as

$$f(x) = \frac{1}{1+e^{-x}} \tag{4}$$

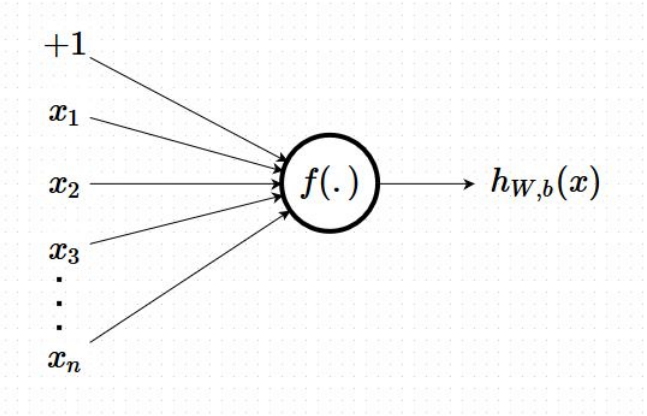

Figure 1: Artificial Neuron

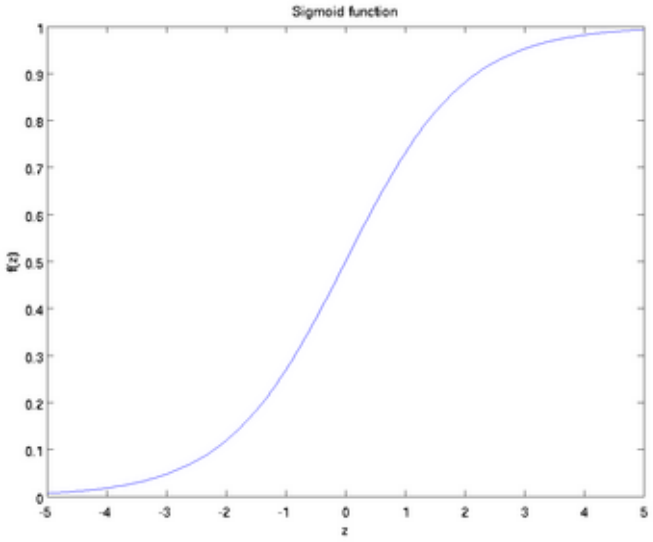

Figure 2: Sigmoid Function

Figure 2 shows the plot of sigmoid function.

• **Hyperbolic Tangent**: Also known an tanh, this function $f : \Re \mapsto \Re$ is given as

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5}$$

Figure 3 shows the plot of tanh function.

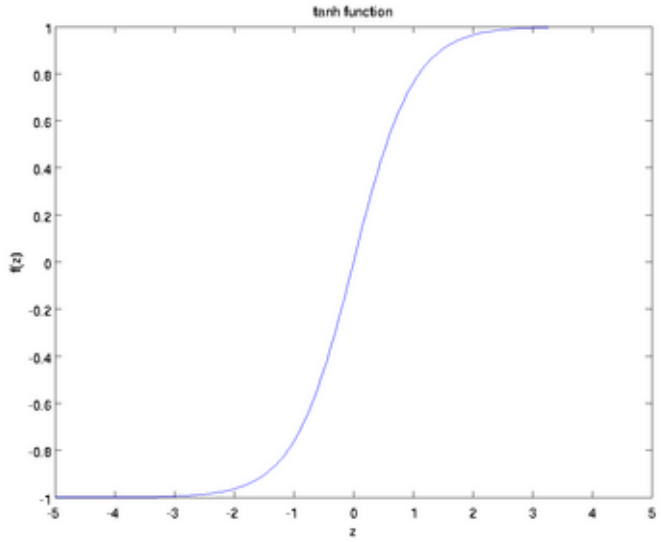

Figure 3: Hyperbolic Tangent Function

- **Rectified Linear Unit**: Also known as ReLU, this function $f : \Re \mapsto \Re$ is given as

$$f(x) = ReLU(x) = max0, x \tag{6}$$

Figure 5 shows the plot of ReLU function.

3.1.1 *Training*

Artificial Neural Network is trained to minimize the loss/error. Steps for training are given below.

- **Forward Propagation**: Input vector is fed into the neural net and it undergoes two transformations.
  - For the j^th hidden neuron, activation is given by:

$$a_j^2 = f\left(\sum_{i=1}^{n} W_i^{(1)} x_i + b^{(1)}\right) \tag{7}$$

  - For the $j^{(th)}$ output neuron, activation is given by:

$$h_{W,b} = a_i^3 = f\left(\sum_{i=1}^{k} W_i^{(2)} a_i^2 + b^{(2)}\right) \tag{8}$$

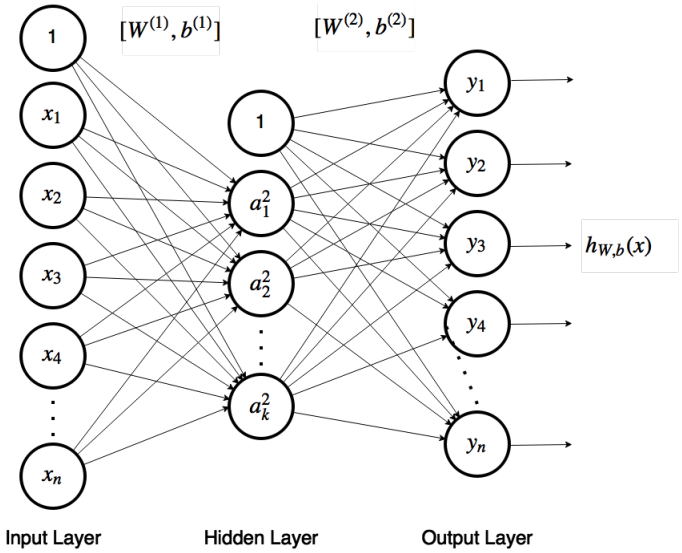

Figure 4: Neural Network

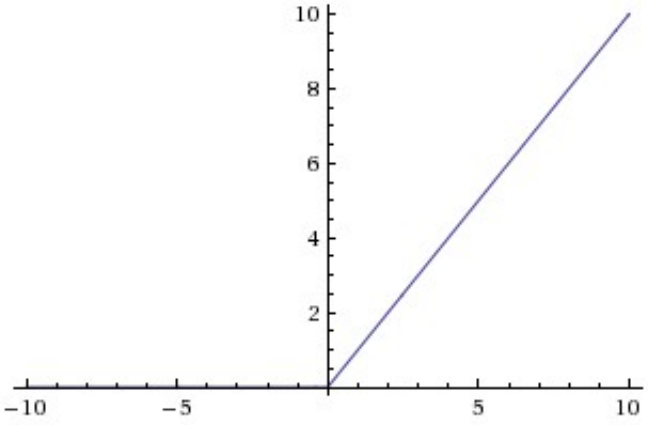

Figure 5: Rectified linear unit

- **Objective Function** After getting the output of the final layer of the network, the observed output $h_{W,b}$ is compared with the target output y and the error function is created. There are usually two types of error functions used.
  - **Squared Error based**: In this we minimize the squared difference of the observed output and the target output. The cost function is given by:

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 \tag{9}$$

Given 'm' training samples the overall cost function can be written as,

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(W_{ij}^{(l)}\right)^2 \quad (10)$$

λ is called regularization parameter, which penalizes theta taking higher values. This helps to prevent overfitting. λ is also called weight decay term.

- **Cross-Entropy based**: In this we minimize the negative log-likelihood of the hypothesis i.e the observed output. Given 'm' training samples the overall cost function can be written as,

$$J(W,b) = -\frac{1}{m}\left[\sum_{i=1}^{m}\left(y^{(i)}\log(h_{W,b}(x^{(i)})) + (1-y^{(i)})\log(h_{W,b}(x^{(i)})\right)\right] \quad (11)$$

- **Backpropogation:** Backpropogation algorithm is the most effecient way to compute error gradients. These error gradients are back-propagated from upper layers to lower layers using this algorithm. The gradients calculated can then be used by any optimization algorithm for optimizing the parameters.
- **Optimization**: Next step is the minimization of the cost function $J(W,b)$ as a function of W and b. First the parameters W and b are initialized randomly. Then the cost function is minimized using optimization algorithms. There are various optimization algorithms available for this purpose such as -
  - Stoschastic Gradient Descent
  - Conjugate descent
  - BFGS
  - LBFGS

3.2 CONVOLUTIONAL NEURAL NETWORK

In the previous section the network is efficient if the size of the input is small, like small image patches. If we want to scale up to use more realistic datasets that have high resolution images, we need to use CNNs.

In traditional neural nets, we fully connect all the hidden units to all the input units. On the relatively small images (e.g. 28x28

images for the MNIST dataset), it was computationally feasible to learn features on the entire image. However, with larger images (e.g., 96x96 images) learning features that span the entire image (fully connected networks) is very computationally expensive as we would have about 10^4 input units, and assuming we want to learn 100 features, we would have on the order of 10^6 parameters to learn. The feedforward and backpropagation computations would also be about 10^2 times slower, compared to 28x28 images.

One simple solution to this problem is to restrict the connections between the hidden units and the input units, allowing each hidden unit to connect to only a small subset of the input units. Specifically, each hidden unit will connect to only a small contiguous region of pixels in the input. This idea of having locally connected networks also draws inspiration from how the early visual system is wired up in biology. Specifically, neurons in the visual cortex have localized receptive fields (i.e., they respond only to stimuli in a certain location).

3.2.1 *Convolution*

Natural images have the property of being "stationary", meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.

More precisely, having learned features over small (say 8x8) patches sampled randomly from the larger image, we can then apply this learned 8x8 feature detector anywhere in the image. Specifically, we can take the learned 8x8 features and "convolve" them with the larger image, thus obtaining a different feature activation value at each location in the image.

An image in its basic form is a matrix. When we do a 2D convolution, we convolve this image with another matrix whose dimension is less than the image. Convolving means we center the kernel on the image pixel and then do an element wise multiplication of the neighbourhood pixels and then summation of the result. The result substitutes the pixel intensity on which the kernel was centered on. The kernel is then slided and centered on the next pixel. e.g Consider a kernel F of size 3×3. The convolution is given by:

$$F * I(x, y) = \sum_{j=-3}^{3} \sum_{i=-3}^{3} F(i, j) I(x - i, y - j) \tag{12}$$

As shown in the figure 6 kernel is convolved with the image resulting in another image.

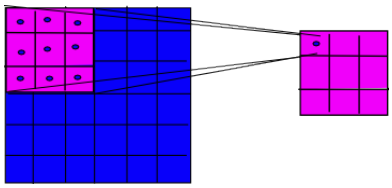

Figure 6: Convolution

From the figure above it is clear that each output depends on the number of elements of the kernel. These elements are termed as weights. These weights are shared by full image. This is in contrast to the traditional neural nets where number of weights is equal to number of pixels in the image. Hence, it is computationally more efficient when the the input images are of high resolution.

3.3 CONVOLUTIONAL NEURAL NETWORK

In a CNN we give the complete image as input. Then we convolve the image with the kernel. Note that if the image is 3 dimensional i.e. it contains the RGB planes then the kernel will also be 3 dimensional. As we fix the number of hidden neurons in a layer of traditional neural network, here also we fix a number of hidden units. Each hidden unit will act like a different kernel and will produce a feature map. Each feature map is the result of convolution of the image will a specific kernel. See figure 7.

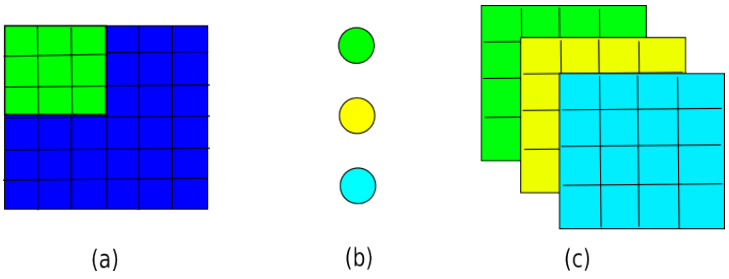

Figure 7: (a) This shows the kernel centered on pixel (2×2) which will slide to the right and then continue to the next row. (b) This shows 3 hidden units each of which correspond to a different kernel. (c) This shows the feature maps produced by each of the kernel.

The next step after convolution is *pooling*.This is used for down sampling the data. Pooling operation is applied on the image/feature map resulted after the convolution. The most basic type of pooling is *max-pooling*. The resultant feature map contains the maximum value selected from a neighbourhood of pixels.If the image is 6×6, then after pooling on neighbourhoods of 3×3 we get the output feature map which is of size 2×2. See figure 8

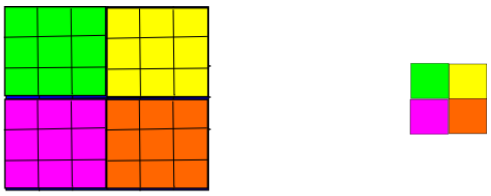

Figure 8: (a) It shows the 6×6 image subdivided into regions of 3×3. (b) It shows one value from each region after the pooling operation

The training is done similar to traditional neural net with slight difference in backpropogation.

One of the major problems in neural networks is that if the network is very big then it tends to over fit the data. We want our network to be generalized i.e. it should perform good on unseen data samples as well. We need to regularize the network which can be done using a technique known as *dropout*. In this we randomly select some of the inputs and set them to zero.

4

TORCH7

This chapter explains the library used for making the project. Torch is a scientific computing framework with wide support for machine learning algorithms. It has various numeric optimization routines, linear algebra routines, neural network and energy-based models and routines of working with n-dimensional matrices. At the heart of Torch are the popular neural network and optimization libraries which are simple to use, while having maximum flexibility in implementing complex neural network topologies. It is maintained by research scientists and engineers from Facebook, Google and Deep Mind.

The modules that I have used in the project are as follows

- **nn.Sequential**:
This is the base module that helps to create a sequential network. In this module we add all the modules we want, such as Sigmoid, Dropout etc.

- **nn.ParallelTable**:
As the network that I have created consists of two parallel network, I have used this module to have two sequential networks in parallel.

- **nn.SpatialConvolutionMM**:
This module is responsible for doing 2D convolution on the input image. It is a generalized module which can perform convolution on input of any depth.

- **nn.SpatialMaxPooling**:
This module helps us to do a max pooling over neighbourhood values. This module finds the maximum in the neighbourhood and gives as the result.

- **nn.ReLU**:
This module works as the Rectified Linear Unit activation function.

- **nn.Dropout**:
This module accepts the percentage of inputs to be put to zero and acts as Dropout hence regularizing the network.

• **nn.Linear**:
This module performs a simple linear regression i.e. multiplies the weights to each of the inputs.

• **nn.joinTable**:
This module joins/concatenates the tensor/output produced by the two parallel networks.

• **nn.Forward**:
This function accepts the input data and feed forwards it via complete network to return the outputs.

• **nn.Backward**:
This function performs the backpropogation on the network to get the parameter gradients.

• **nn.MSECriterion**:
This is the Mean square loss function that I have used in the network to find out the error between network predictions and the targets.

• **optim.sgd**:
Optim is the optimization package that contains stochastic gradient descent (SGD) algorithm that is used for optimizing the parameters of the network.

5

METHODOLOGY

This chapter explains the architecture used in the project. The model consists of two sequential networks aligned in parallel which are joined at the end. Each of the sequential network consists of two convolutions with max pooling and ReLU as activation functions after each convolution.

parallel = nn.ParallelTable()
This creates a parallel stack in which we can add our modules.

model = nn.Sequential()
model:add(nn.SpatialConvolutionMM())
model:add(nn.ReLU())
model:add(nn.SpatialMaxPooling())
This creates one of the model which has to run on this parallel stack.

parallel:add()
This adds the the above created model into the parallel stack.

model:add(nn.JoinTable()) This joins the output of the above created parallel networks.

model:add(nn.ReLU())
model:add(nn.Dropout(0.5))
model:add(nn.Linear())
model:add(nn.Sigmoid())
This is called the fully connected layer which gives the predicted values as output.

criterion = nn.MSECriterion()
This is a loss function which tracks how much is the error. The objective is to reduce the error by using some optimization algorithm.

See figure 9 for the architecture. (1) is the first layer which is a parallel layer. In this layer there are 2 convolutional neural nets of 12 layers each. The outputs from this parallel pipeline is concatenated in layer (2). Layers (2) to (7) are the fully connected layers. We give two images as input to the network. These two images are the patches, each extracted form stereo image pairs. These two images pass through the two parallel convolutional neural networks. The two sets of features extracted in the parallel pipeline are then concatenated and fed into fully connected network for depth map prediction.

```
th>
th> model
nn.Sequential {
  [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> output]
  (1): nn.ParallelTable {
    input
      |`-> (1): nn.Sequential {
      |      [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9) -> (10) -> (11) -> (12) -> output]
      |      (1): nn.SpatialConvolutionMM(3 -> 300, 7x7)
      |      (2): nn.ReLU
      |      (3): nn.SpatialMaxPooling(2,2,2,2)
      |      (4): nn.SpatialConvolutionMM(300 -> 200, 7x7)
      |      (5): nn.ReLU
      |      (6): nn.SpatialMaxPooling(2,2,2,2)
      |      (7): nn.View
      |      (8): nn.Dropout(0.500000)
      |      (9): nn.Linear(5000 -> 400)
      |      (10): nn.ReLU
      |      (11): nn.Linear(400 -> 840)
      |      (12): nn.Sigmoid
      |    }
      |`-> (2): nn.Sequential {
      |      [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9) -> (10) -> (11) -> (12) -> output]
      |      (1): nn.SpatialConvolutionMM(3 -> 300, 7x7)
      |      (2): nn.ReLU
      |      (3): nn.SpatialMaxPooling(2,2,2,2)
      |      (4): nn.SpatialConvolutionMM(300 -> 200, 7x7)
      |      (5): nn.ReLU
      |      (6): nn.SpatialMaxPooling(2,2,2,2)
      |      (7): nn.View
      |      (8): nn.Dropout(0.500000)
      |      (9): nn.Linear(5000 -> 400)
      |      (10): nn.ReLU
      |      (11): nn.Linear(400 -> 840)
      |      (12): nn.Sigmoid
      |    }
       ... -> output
  }
  (2): nn.JoinTable
  (3): nn.Linear(1680 -> 400)
  (4): nn.ReLU
  (5): nn.Dropout(0.500000)
  (6): nn.Linear(400 -> 1681)
  (7): nn.Sigmoid
}
```

Figure 9: Architecture

The network parameters are optimised using Stochastic gradient decent. The targets are unrolled into a one dimensional vector which are then compared to the predicted outputs. As said above, the loss is calculated by using mean squared error.

6

RESULTS

6.1 DATA PREPROCESSING

The network is trained on KITTI dataset [5]. This dataset was chosen because of its size. As the network size increases, it needs more and more data to train therefore with large dataset we will be able to produce even larger number of patches. Moreover this data consists of outdoor images therefore it is very challenging. It consists of a set pairs of images with corresponding disparity maps. This disparity map is created using laser scanner. Disparity is present only in the lower half of the image so images are cropped to consider only the regions where disparity is given. Then the stereo images are converted from RGB to YUV format.

These stereo pairs and their corresponding depth maps are then divided into patches of 41×41. The network is trained using these patches.

6.2 TRAINING

Input to the network is given in the form of {Left_Image_Patch, Right_Image_Patch} and target is defined as unrolled version of the Depth_Image_Patch. Training is done using mini-batches of size 250 with SGD using a learning rate of 0.1. First the network was trained with $80,000$ image patch pairs. The results showed that there is still need to train more as there lot of noise in the output. See figure 10

Upon increasing the number of patches for training to 3 lakhs we get the results shown in figure 11

With the training error given in figure 12. We can see that the training error has converged however there is still scope for improvement.

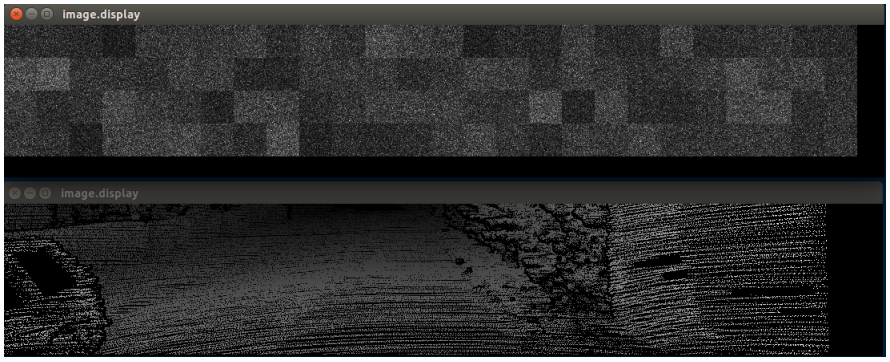

Figure 10: Top image is the predicted depth map
Bottom image is the ground truth

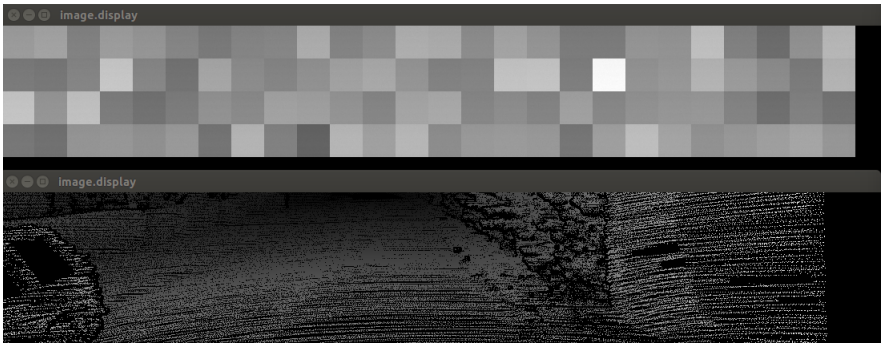

Figure 11: Top image is the predicted depth map
Bottom image is the ground truth

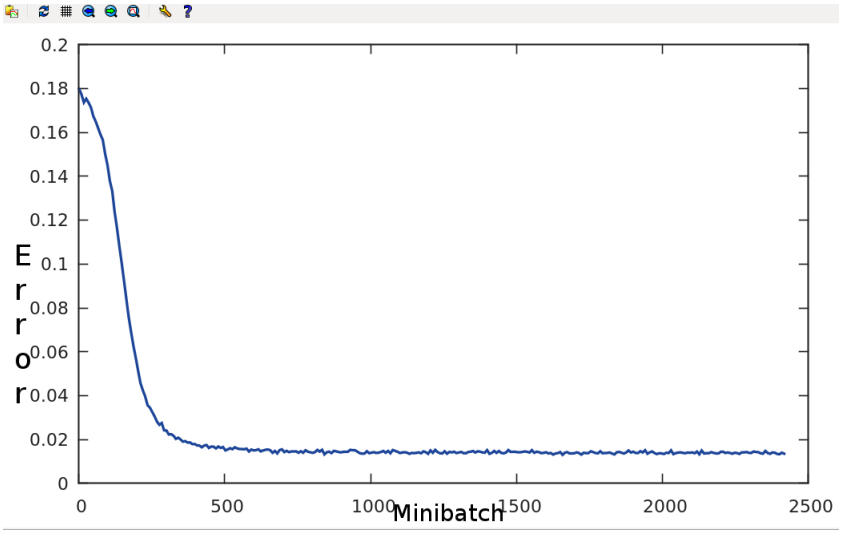

Figure 12: Training error vs Minibatches

7

FUTURE WORK

The next possible avenue for improving the results is increasing the number of convolution layers from two to three or four. Increasing the network depth will help to learn more complex hypothesis. And hence bring down the error. As we are not padding input image patch so after every convolution the size of the image decreases. Then after pooling it decreases even more. So we need to increase the size of the image patch to around 90×90. This will create a problem because as the input size will increase, the computation will also increase so we need GPUs to do the computation. Torch can be easily integrated to GPU with CUDA but this is out of the scope of the project.

BIBLIOGRAPHY

[1] F. Besse, C. Rother, A. Fitzgibbon, and J. Kautz. Pmbp: Patchmatch belief propagation for correspondence field estimation. *International Journal of Computer Vision*, 110(1):2–13, 2014.

[2] 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*, pages 2366–2374, 2014.

[3] S. K. Gehrig, F. Eberli, and T. Meyer. A real-time low-power stereo vision engine using semi-global matching. In *Computer Vision Systems*, pages 134–143. Springer, 2009.

[4] A. Geiger, M. Roser, and R. Urtasun. Efficient large-scale stereo matching. In *Computer Vision–ACCV 2010*, pages 25–38. Springer, 2011.

[5] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[6] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun. 3d traffic scene understanding from movable platforms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(5):1012–1025, 2014.

[7] F. Güney and A. Geiger. Displets: Resolving stereo ambiguities using object knowledge.

[8] P. Heise, S. Klose, B. Jensen, and A. Knoll. Pm-huber: Patchmatch with huber regularization for stereo matching. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2360–2367. IEEE, 2013.

[9] A. Hosni, M. Bleyer, M. Gelautz, and C. Rhemann. Local stereo matching using geodesic support weights. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 2093–2096. IEEE, 2009.

[10] B. Julesz. Binocular depth perception of computer-generated patterns. *Bell System Technical Journal*, 39(5):1125–1162, 1960.

[11] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(9): 920–932, 1994.

[12] K. Konda and R. Memisevic. Unsupervised learning of depth and motion. *arXiv preprint arXiv:1312.3429*, 2013.

[13] L. Ladickỳ, P. Sturgess, C. Russell, S. Sengupta, Y. Bastanlar, W. Clocksin, and P. H. Torr. Joint optimization for object class segmentation and dense stereo reconstruction. *International Journal of Computer Vision*, 100(2):122–133, 2012.

[14] X. Mei, X. Sun, W. Dong, H. Wang, and X. Zhang. Segment-tree based cost aggregation for stereo matching. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 313–320. IEEE, 2013.

[15] A. Saxena, J. Schulte, and A. Y. Ng. Depth estimation using monocular and stereo cues.

[16] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[17] D. Wei, C. Liu, and W. T. Freeman. A data-driven regularization model for stereo and flow. In *3D Vision (3DV), 2014 2nd International Conference on*, volume 1, pages 277–284. IEEE, 2014.

[18] J. Žbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. *arXiv preprint arXiv:1409.4326*, 2014.

[19] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.