Depth Estimation from a Single Image Using a Deep Neural Network

Milestone Report

Rawan Alghofaili

March 16, 2015

1 Introduction

By using the intrinsic and extrinsic camera parameters, Multi-view Stereo has been applied to accurately estimate depth maps. Although this can produce impressive results, the camera parameters(e.g. focal length, disparity and baseline) are necessary for the estimation. This means that the depth can not be estimated unless we have prior knowledge about the image's origin. In addition, two slightly different views of a particular scene are needed in order to reconstruct its depth map.

As interest grows in deep neural networks and with the introduction of readily available depth sensors such as Microsoft Kinect, depth estimation from single view images has become an open and interesting research problem in the computer vision community.

In this project I trained a convolutional neural network to estimate depth from a single image. It will be explained later in this report how the Places [5] pretrained convolutional network will be utilized to solve this problem. Further architectural implementation details will be explained as well.

2 Problem Statement

In this project, I've aimed to train a deep convolutional neural network to produce estimated depth maps of single view images. I used a neural network architecture inspired by [1], which I will explain in more detail in the following section. As you can see from 1, I used features extracted from PlacesCNN[5] (a convolutional network trained to classify indoor and outdoor scenes) to train the network.

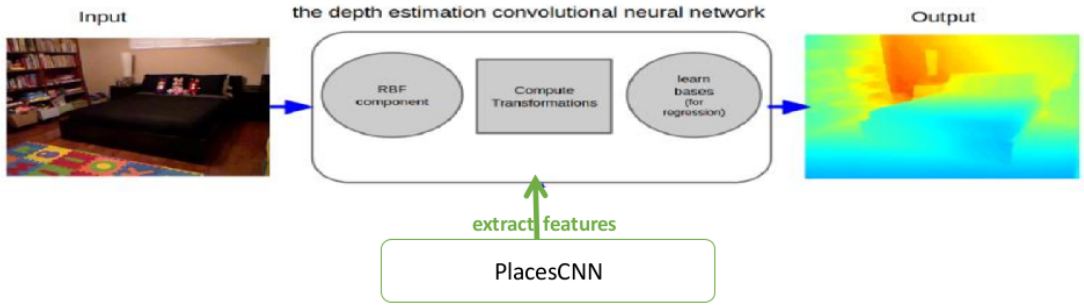


Figure 1: The architecture of the convolutional network. Input: a single view image; Output: a depth map.

3 Related Work

In [4] they used depth maps of indoor scenes produced by a Microsoft Kinect to successfully classify scenes. I used the NYUv2[4] benchmark as a ground truth measure of depth for training.

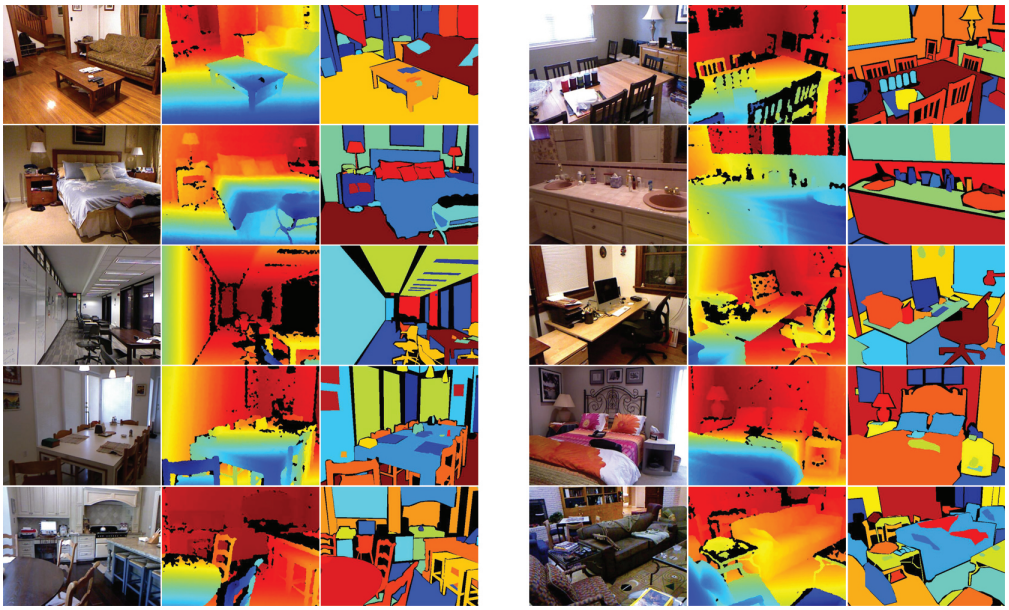


Figure 2: RGB images,their corresponding depth images and labeled images taken from the NYUv2 dataset

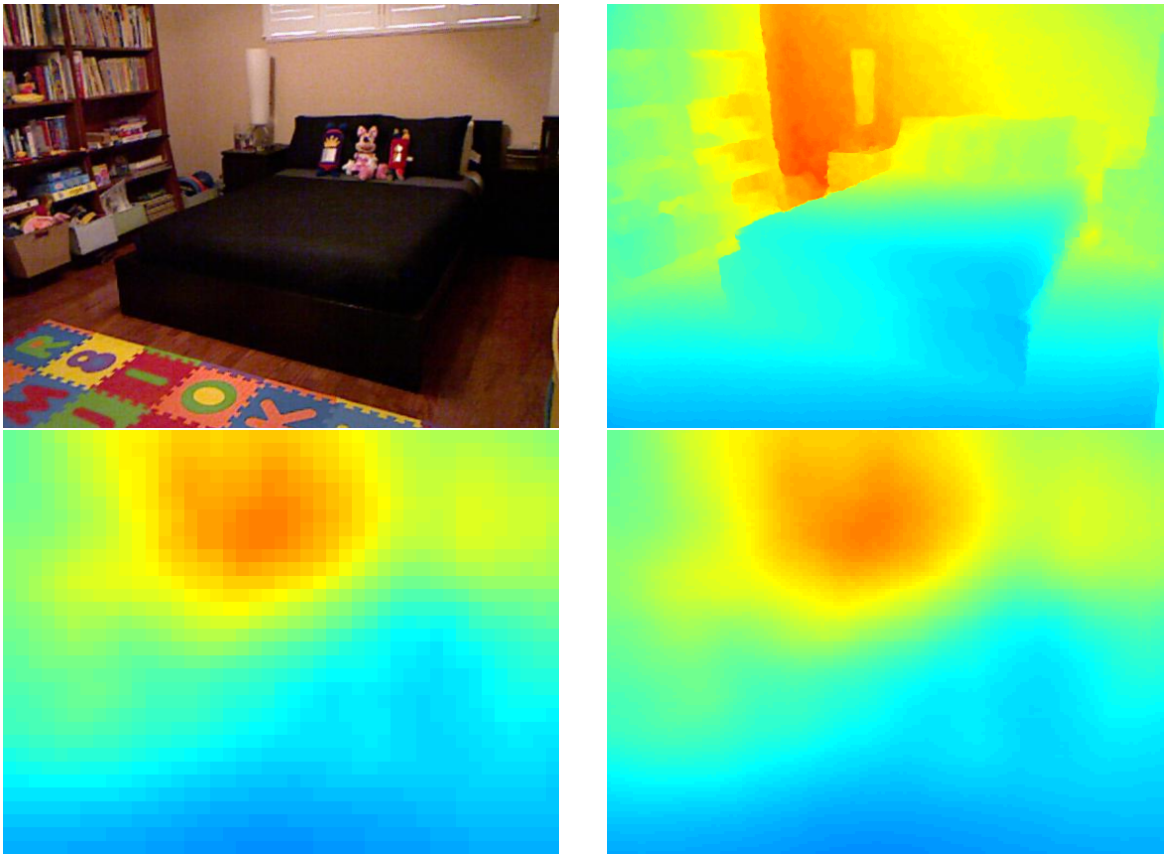


Figure 3: Results from [1]. Top left : input image, top right : ground truth, bottom left : coarse prediction, bottom right : refined prediction.

From [1] and [4] we concluded that scene information give us a good coarse depth estimate. Meaning, knowledge of the scene will provide us with global cues about the image's spatial structure such as the location of floor and ceiling edges. And by using image features extracted from the PlacesCNN[5] pool5 layer we can exploit scene information in estimating our depth map.

[1] also deduced that further local refinements of the global estimate can give us a high accuracy depth map. As in [1] I will as well attempt to use PlacesCNN[5] as a coarse estimate. However, I will use a deep neural network to learn the depth dictionary and transformation T instead of the coupled regression approach presented in the paper.

4 Method

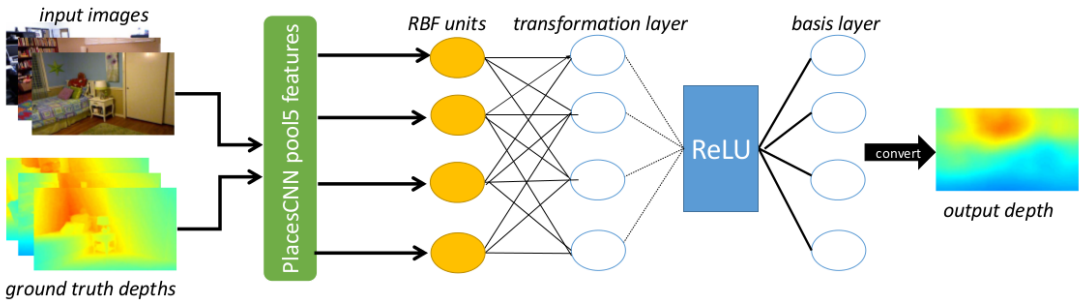


Figure 4: The architecture of the convolutional network in more detail.

Given a training set comprised of feature vectors extracted from PlacesCNN[5] after inputting NYUv2[4] single view images, I've trained a convolutional neural network to produce depth estimates using center feature vectors randomly chosen from the training set.

As previously stated this convolutional neural network architecture is inspired by [1]'s approach of using RBF kernels in estimating depth. The *RBF units* in Figure 4 refer to $\phi_j(x) = exp(-||f(x) - f(c_j)||^2/2\sigma^2)$ defined in [1]. Both the transformation and bases have been translated into fully connected convolutional neural net layers, *transformation layer* and *basis* layer respectively.

5 Implementation

5.1 Layer set up

Caffe [2] was used as a skeleton for implementation. The *RBF unit* ($\phi_j(x) = exp(-||f(x) - f(c_j)||^2/2\sigma^2)$) from [1] was implemented as a caffe "Blob" or layer, where the center c_j is, as previously mentioned, the fixed feature vector extracted from the Places CNN.

5.2 Feedforward

The feedforward operation was relatively trivial to implement once the caffe skeleton code was understood. By overriding the *ForwardCPU()* method in the Blob with my implementation of $\phi_j(x)$ here referred to as the function RBF():

for *each center* c_i **do**
| output[i] = RBF(c_i, feature vector)
end
where $i \in \{1, 2, ..., n\}$ and n is the number of centers(feature vectors extracted from places)

5.3 Backpropogation

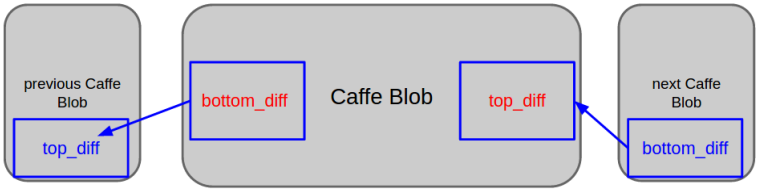


Figure 5: Gradient storage in Caffe

In caffe, the gradient for each Blob is computed and stored in *bottomdiff*. The contents of *bottomdiff* are automatically copied to *topdiff* of the previous later. The gradient from the next layer (*topdiff* in the current Blob) is multiplied by the output of RBF() and added to *bottomdiff* of the current Blob:

for *each center* $c_i \in$ *previous Blob* **do**
| bottomdiff += RBF(c_i, feature vector)*topdiff
end

6 Testing

Unit tests were written to test the layer setup, feedforward and backpropagation. Layer setup was tested by creating a Blob and verifying its dimensionality. Feedforward was tested by injecting an input vector of ones and ten center feature vectors of ones as well. In order for feedforward to produce the correct results each center should output a value of one. This is because the l2-norm of $x = [1, 1, ..., 1]$ and center $c_i = [1, 1, ..., 1]$ should equal to 0. Because $e^0 = 1$ each center should output one in the feedforward operation.

As for the backpropogation unit test, caffe provides a GradientChecker utility which I've used to my advantage. Using finite differencing it estimates a gradient and compares it to the gradient computed by backpropogation. As you can see in Figure 6, the layer passed all three tests as well as tests conducted by caffe.

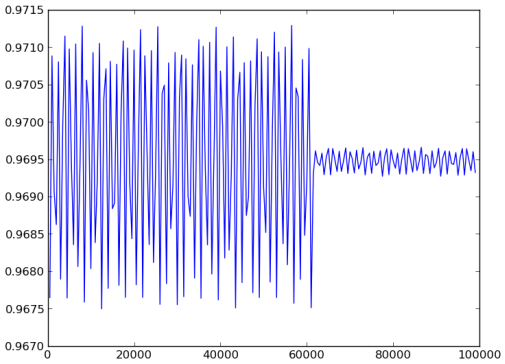
```
[----------] 3 tests from ThetaLayerTest/1, where TypeParam = caffe::DoubleCPU
[ RUN      ] ThetaLayerTest/1.TestForward
[       OK ] ThetaLayerTest/1.TestForward (0 ms)
[ RUN      ] ThetaLayerTest/1.TestSetUp
[       OK ] ThetaLayerTest/1.TestSetUp (0 ms)
[ RUN      ] ThetaLayerTest/1.TestGradient
[       OK ] ThetaLayerTest/1.TestGradient (3 ms)
[----------] 3 tests from ThetaLayerTest/1 (3 ms total)
```

Figure 6: RBF component *ThetaLayer* passing unit tests

7 Results

7.1 First Experiment

I trained my model using a portion of the NYUV2[4] data set of depths, their features extracted from PlacesCNN's pool5 layer[5]. I used both the center crop and mirrored crop pool5 features, this gave me 1590 training examples.



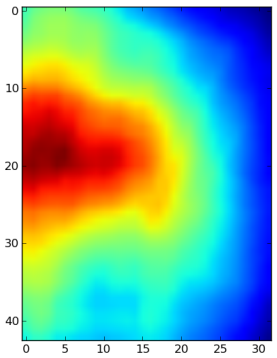


Figure 7: *Left:* training using the raw images. The horizontal axis represents the iteration number, and the vertical axis is the RMSE(Root Mean Square Error) of the validation set; *Right:* a visualization of the depth produced from a validation example

You can see from Figure 7 that the training wasn't going so smoothly. I visualized the depth estimates of some of the validation examples and they seemed visually similar. I concluded that the network might be learning the mean depth map instead of individual depth estimates.

7.2 After Subtracting the Mean

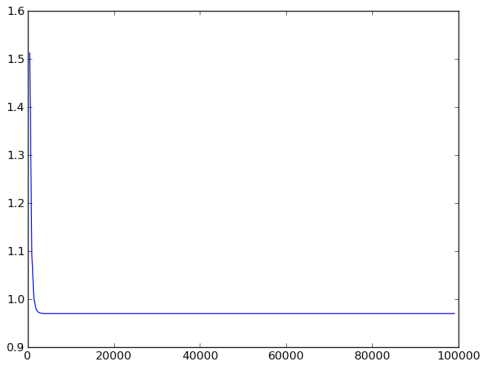


Figure 8: Training after mean depth subtraction. The horizontal axis represents the iteration number, and the vertical axis is the RMSE(Root Mean Square Error) of the validation set; *Right:* a visualization of the depth produced from a validation example

As you can see from Figure 8, the training is less jittery. However, the validation error is still two low.

8 Further Work

After scripting further unit tests to verify the *ThetaLayer* output, I've found that it's reading the centers incorrectly. I'm implementing a different strategy to input the centers which will hopefully improve the training error. I've also found a bug lurking in my data pre-processing step. The mean was not being subtracted from the depths correctly. The bug has been handled and the preprocessing tested and after the *ThetaLayer* is corrected, I will be re-training the model.

The next step after seeing an improvement in the RMSE, would be to merge this convolutional neural net with Places and start finetuning.

References

[1] Baig, M., Torresani, L.: *Coarse-to-fine Depth Estimation from a Single Image via Coupled Regression and Dictionary Learning.* arXiv preprint arXiv:1310.1531, 2013.

[2] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: *Decaf: A deep convolutional activation feature for generic visual recognition.*

[3] Seitz, S., Curless, B., Diebel, J., Scharstein, D., Szeliski, R.: *A comparison and evaluation of multi-view stereo reconstruction algorithms.* In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition. (2006).

[4] Silberman, N., Derek Hoiem, Fergus, R.: *Indoor segmentation and support inference from rgbd images.* In ECCV, 2012.

[5] Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., Oliva, A.: *Learning Deep Features for Scene Recognition using Places Database.* Advances in Neural Information Processing Systems 27 (NIPS), 2014.