Single Image Depth Estimation via Deep Learning

Wei Song
Stanford University
Stanford, CA

Abstract

The goal of the project is to apply direct supervised deep learning to the problem of monocular depth estimation of still images. We have done experiments with two different types of deep neural network architecture for depth estimation, and performed regression on the sparse coding of depths with appropriate objective functions. Our best model achieved an average error in the ln space of 0.0891 for the training and 0.26 for the testing set. This preliminary result demonstrates the feasibility of using deep neural network for still image depth estimation.

1 Introduction

Despite the fact that humans can easily infer the depths of scenes presented in pictures, monocular depth estimation has long been a difficult problem in computer vision research. This is mainly due to the fact that without high level understanding of the real world, local appearance alone is insufficient to resolve depth ambiguities. So far, many of the state-of-the-art approaches apply probabilistic graphical models to enforce local consistencies among the depth pixels and apply some other high-level information for resolving ambiguities. Some recent research done at Stanford such as the one described in [1] applied super-pixel-based Markov Random Field (MRF) while using a predefined-set of semantic labels (e.g. sky, tree) as the prior for the depths; in [2], the authors also applied MRF and incorporated the likelihood of the 3D structure of the scenes.

In this project, we take a different approach by directly performing training on the whitened images while using some representation of the depths as the labels without introducing any other features or prior knowledge. Specifically, we will be using deep neural network for this task. Deep neural networks have shown great potential in many computer vision applications such as the winning model in the 2012 LSVRC contest for image classification [3]. Such impressive results demonstrate deep learning's capability of capturing high level concepts entirely from the images. Therefore, it is likely that it can also resolve depth ambiguities when trained with sufficient amount of data. Another motivation for this project is for us to experiment training a shared network for both depth estimation and objection detection and investigate if the performance of either can be improved as a result of this multi-task learning – which has shown to work well in NLP literature as demonstrated in [5].

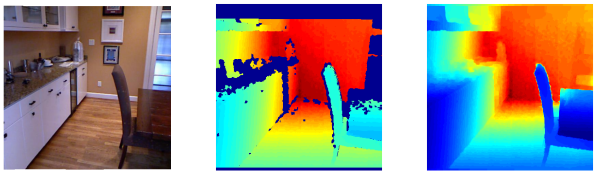

Figure 1: Projected depth map before and after pre-processing.

2 Dataset

We use the NYU Depth Dataset V2 for both training and testing [4]. It contains 407,024 raw frames of indoor scenes captured by Kinect. The combined raw data totals 428GB. Each raw frame has an RGB image and an unprojected raw depth map. Upon performing the projection of depths to the original image, significant portions of the depth maps are left empty, which requires additional processing. One of the better default depth-filling algorithm provided by [4] is too slow to pre-process all the raw data. Thus, we used the alternative tool that applies cross-bilateral filters to the depth map. However, this alone is unable to fill every empty pixel and as a final step, we apply a greedy algorithm that uses the closest depth value along each axis to fill the remaining depth pixels. Figure 1 illustrates the output of combined pre-processing. Since this dataset is still relatively small for a typical deep learning task, after downsampling each image and its depth map to 120×120 pixels, we crop out a 100×100 subimage for every 3 pixels such that we can generate 49 samples per image. This gives us a dataset of size 20 millions. Since within each indoor scene, there are many similarities among the frames, we ensured that our training set – which contains approximately 80% of the samples – do not overlap with scenes appeared in the test set. All image samples are whitened before feeding into our networks.

3 Deep Learning Framework

In this section we provide a high level overview of the two deep network architecture that we are using as well as the infrastructure that they run on. Since many variants of the

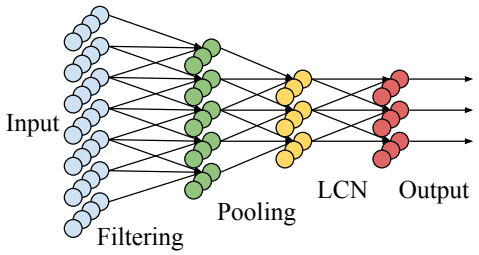

Figure 2: An illustration of one of the layers of RICA

networks have been experimented, the descriptions here are kept as generic as possible.

3.1 RICA Network

This architecture is similar to Google Brain's unsupervised learning network as described in [6]. In a nutshell, each layer in the network contains three sub-layers: a local receptive filtering layer where each neuron connects to a sub-region of its input layer, a pooling layer that where each neuron computes some non-linear function of the input field, and a Local Contrast Normalization (LCN) layer as shown in Figure 2. Typically, two or three such layers are stacked together followed by an output layer. We perform direct supervised learning (or fine tuning) on the data without unsupervised pre-training. The output layer is also different as it will be discussed later. Other hyper-parameters such as the input size, filtering size, number of depths maps, pooling sizes and pooling strategies also vary.

3.2 Krizhevsky's Network

This is also known as the "Drednet". One of the key differences between the RICA and the Krizhevsky's network is that unlike RICA, the filters are all convolutional. That is, the weights of the filters in each layer are tied. This significantly reduces the number of parameters need to be learned per map, which allows a higher number of maps generated at each layer [3]. The layers of Krizhevsky's network is not as ordered as RICA's mentioned earlier. The first five layers are mixed combination of pooling, LCN and filtering layers of various sizes, while the remaining two layers are densely connected layers. Another important aspect of Drednet is the activation unit. Typically, a smooth differetiable function such as the sigmoid or tanh functions are used for activation. In Krizhevsky's network, Rectified Linear Units (ReLUs) are used, which simply takes the maximum between the input and 0. Unsurprisingly, ReLUs are very efficient, and in fact yield better results for many vision related tasks [3]. One regularization technique applied is random dropouts, which randomly omits neurons during training such that the dependencies among neurons are reduced. We use this network as the baseline and tweak many of the hyper-parameters for experiments.

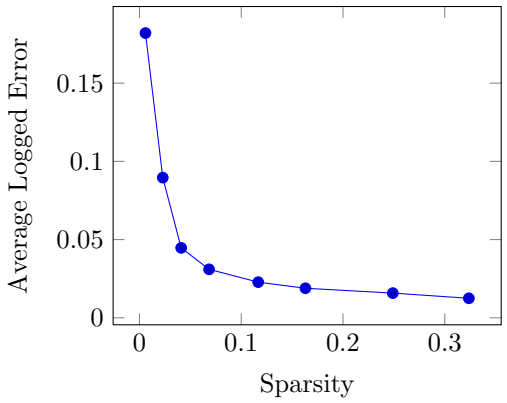

Figure 3: Average absolute error in log space for reconstruction vs the sparsity (i.e. percentage of non-zero values of the embeddings) by varying β

3.3 Infrastructure

All of the training and evaluation of the deep networks are done using the distributed GPU infrastructure as described in [7]. At a high level, each machine has 4 high-end GPUs, and the intercommunication between the machines rely on InfiniBand to match the desired throughput. Most of the networks used in this project uses 1-2 machines or 2-8 GPUs. The configurations of the networks are written in Python, but at the low-level, most of the computational intensive tasks are written in CUDA. MPI is used to coordinate the workers.

4 Sparse Coding of Depths

Initially, we defined our objective function as the squared L_2 norm between the logged depth values and the prediction. However, we obtained very poor results for both RICA and Krizhevsky's networks. In particular, it seems that neither of the two networks is learning the right objective as shown in Section 6. Thus, we decided to change our representation of the depths. Instead of asking the network to predict the logged depth values for every pixel, we encode the depth maps through sparse coding such that each depth map is represented as four vectors where each vector represents the sparse embedding of a quadrant of the depth map. We used the method and code as described in [8]. The variant of sparse coding we used can be formulated as follows,

$$\begin{aligned} \text{minimize}_{B,S} \quad & \frac{1}{2\sigma^2}\|X - BS\|_F^2 + \beta \sum_{i,j} \|S_{ij}\|_1 \\ \text{subject to} \quad & \sum_i B_{ij}^2 \leq c, \forall j = 1, ..., n \end{aligned}$$

where X is the original data, B is the dictionary, and S is the sparse representation. Holding either B or S fixed makes the problem convex, but it's not convex when both can be optimized. The authors in [8] proposed an efficient algorithm that alternates the optimization of B and S while holding the other variable fixed.

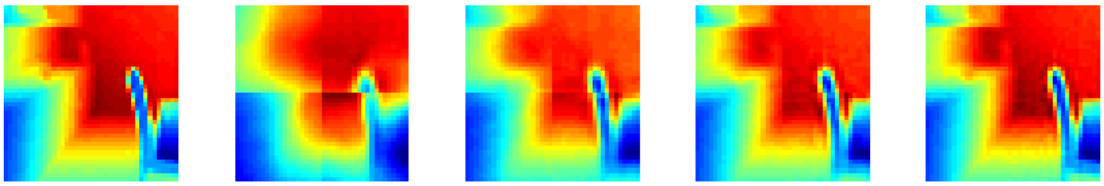

Figure 4: Image on the far left is the ground truth. $\beta = 1, 0.1, 0.01, 0.001$ are used for the four reconstruction. The sparsities of their representations are 1.5%, 5.2%, 14%, 28%, respectively.

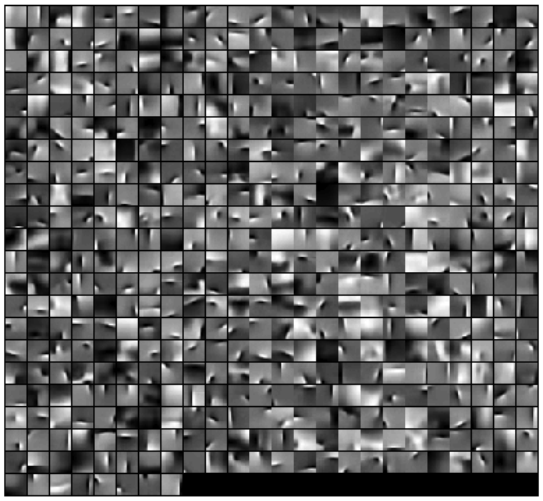

Figure 5: Our 512-basis sparse coding dictionry learned from 4 miilion 20×20 depth patches.

Models	Avg L_2 logged Test Err
Krizhevsky's baseline	3.12
No Dropout	3.24
Three 2048 dense layers	3.08

Table 1: Some failed but saved results of models based on Krizhevsky's network not using sparse coding.

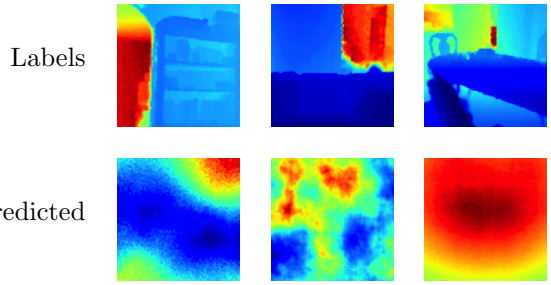

Figure 6: Poor results when not using sparse embedding as labels. Vast majority of the output by both networks did not resemble any pattern of the target labels.

For our dataset, we downscale each jittered depth map to 40×40. We are willing to downsample to such small scale because the texture in depth maps are relatively low compared to typical images. We then subtract a previously computed mean of all downsampled depth maps and break it down into four 20×20 patches – one for each quadrant. For each patch, we encode it using the dictionary that we learned from approximately 4 million patches generated in a similar fashion. Figure 4 illustrates the effect of varying the L1 regularization constant β, which regulates the amount of sparsity during reconstruction. Note that permitting higher sparsity yields almost a perfect reconstruction of the downsampled ground truth depth as shown in figure 3. For our problem, we use $\beta = 0.1$, which yields an average sparsity around 5% and 0.04 average logged difference for all the patches. This strategy transforms the representation of each depth map of a jittered image to a vector of size 2048 with approximately 100 non-zero values when using a sparse dictionary size of 512.

The reason we use sparse coding over other methods such as PCA is that sparse coding allows over-complete basis, which gives more expressive power to the dictionary while maintaining low sparsities. Our learned dictionary of size 512 is shown in Figure 5. Note that this may look different from a typical dictionary learned on whitened images. One of the reasons is that in order to reconstruct the original depth map, the dictionary must retain the depth values within the basis vectors. As a result, there aren't as many patches with mostly gray regions.

5 Objective Function

Using sparse coding to represent the depth maps poses another problem: since the representations are very sparse, when using the common choice of L_1 or L_2 norm as the objective function, the networks often learn to output predictions close to the zero vectors – which in fact do produce very low objective values because we subtract all depth maps by a mean depth map. This is especially the case for RICA network. Inspired by a very recent paper as described in [9] where the authors of the paper faced a similar problem when using 0-1 vector to represent the presence of objects, we attempted to fix this issue by using the following objective function

$$\min_{\Theta} \sum_{i} \left\| (Diag(1\{s^{(i)} \neq 0\}) + \lambda I)^{1/2} (h_{\Theta}(x^{(i)}) - s^{(i)}) \right\|_2^2$$

Here, $s^{(i)}$ is the sparse vector of size n representing the embedding of the depths of the i-th sample. The indicator function converts $s^{(i)}$ to $\{0,1\}^n$ such that a field in the output is 1 if and only if the corresponding field in $s^{(i)}$ is non-zero. Thus, by using smaller values of λ, we are

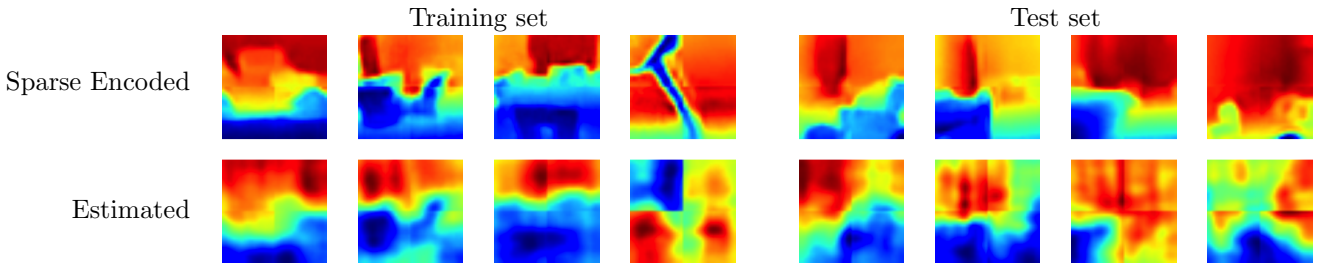

Figure 7: Results obtained using our best model. First columns of each dataset are depths maps that achieved very low error, while the fourth one has very high error.

Model and Obj Function	SC MSE Train	SC MSE Test	Abs log Train Err	Abs log Test Err
RICA, $\lambda = 0.001$	0.0986	0.108	2.588	2.641
RICA, $\lambda = 0.01$	0.0718	0.0847	1.290	1.221
RICA, $\lambda = 0.03$	0.0380	0.0574	0.781	0.788
RICA, $\lambda = 0.05$	0.0364	0.0557	0.724	0.737
RICA, $\lambda = 0.07$	0.0457	0.0577	0.825	0.819
Drednet, $\lambda = 0.05$	0.0461	0.0515	1.105	1.089
Drednet, $\lambda = 0.1$	0.0280	0.0404	0.631	0.652
Drednet, $\lambda = 0.5$	0.0144	0.0313	0.211	0.327
Drednet, L_2	0.0210	0.0366	0.246	0.379
Drednet, Varying Obj	**0.0127**	**0.0276**	**0.0891**	**0.260**

Table 2: Results for models that produced non-zero outputs. Here, SC is the mean squared error of sparse code, and the absolute log errors are absolute average absolute difference between each depth pixel in ln space. The best model was produced by increasing λ during training and then switching to L_2 near the end.

effectively putting more relatively weights on the non-zero fields of the sparse vector, which in turn penalizes the network for outputting zero vectors. As λ approaches infinity, this is analogous to the squared L_2 objective function. Note that the indicator function can be replaced with any other non-negative functions. In particular, we tried to use the absolute and squared values of the sparse vectors. However, they did not prevent the network from outputting near zero vectors, and we believe the main reason is that many of the non-zero values in the sparse representation are relatively small. Thus, when using these values as the relative weights, most values still have the same penalization as compared to the zero fields.

6 Early Results

As mentioned earlier, the results for directly estimating the logged values of the depth maps have been quite poor. Due to a disk failure in AI Lab in late October, much of the evaluations of those models and our input data is lost, and everything has to be re-downloaded and re-processed. Since we've decided to use sparse coding as our new representation and re-running the experiments are rather expensive, we did not go back and prepare the old data input again to regenerate those models. However, we do have some saved results demonstrating the poor performance of those models and justifying why we have decided to switch to sparse coding. Table 1 shows the results of some variations of the Krizhevsky's network. Note that the actual number of models tested far exceeds what's shown here, but the evaluation data is lost and none had been able to achieve any reasonable output. Figure 6 also shows some randomly sampled saved outputs by both RICA and Krizhevsky's network. Clearly, We can observe that depths estimation is not being properly learned at all when performing regression on the labels directly – at least for all the models we've experimented.

7 Results with Sparse Coding

Using sparse coding to represent the depths and with the new objective function we defined earlier, we are finally be able to obtain some models that can begin to estimate reasonable depth maps for images in both training and testing dataset. Figure 7 shows some good and bad output produced by our best models. Despite that our models still have trouble estimating the depths of more complex shapes, they can capture the basic structures of many different scenes. It's worth noting that even though each quadrant is represented by a separate sparse vector, the network is able to produce embeddings that generate smooth boundaries between the quadrants for most samples.

Table 2 shows a subset of models we tried by changing the objective function and other appropriate hyper-parameters. Note that we did not include the outcome of models that output results close to the zero vectors. The reason being that although numerically their errors are comparable to the models we've shown, qualitatively speaking, they tend to generate the same depth map for most images, which is an unacceptable behavior. For the RICA networks, we used map size 8 and filter size 10 with 2 stacked layers. For Krizhevsky's

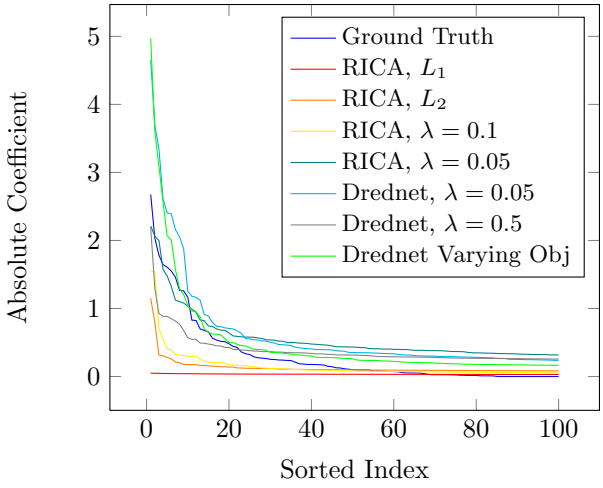

Figure 8: Absolute values of the largest 100 coefficients in the sparse representation of each model sorted in descending order.

network, we used filter size of 12, 5, 3, 3, and 3 for the first 5 layers with max-pooling on the 1st, 2nd and 5th. The pooling size used is 2. The last two dense layers each have 4096 neurons, and the number of maps for the first 5 layers are 100, 260, 384, 384 and 256. The initialization of both networks are models trained for the image classification tasks. Our results show that Drednet performs better than RICA, which is unsurprising due to the larger number of parameters used in Drednet. Relatively speaking, our new objective function seems to make very significant impact on the RICA network. In particular, using λ value that's close to the average sparsity of the embeddings produced the best model. However, this isn't the case for Drednet and using larger λ values seem obtains better results. Our best model was trained by modifying the Drednet's objective function from low $\lambda = 0.05$ to 0.5, and then switched to squared L_2 as training progresses. This indicates that the initialization of the networks for different objective function impacts the convergence of the networks.

Since our network outputs the sparse coding of the quadrants of the depth map, it is also interesting to analyze this output before decoding them to depths values. For a fixed random scene, by taking the absolute values of the coefficients of the sparse representation then sorting them in descending order, we obtained Figure 8, which displays the largest 100 coefficients for each model. While this may not be representative of all embeddings, we can observe some expected trends. Both L_1 and L_2 for RICA produced embeddings that are very close to the zero vectors, and smaller λ values for both networks result in a fatter tail compared to the larger λ values.

8 Conclusion and Future Work

Our project has demonstrated the feasibility of using deep neural networks to directly estimate the depth values of single still images without using any hierarchical or hand-engineered features. Our best models are able to estimate reasonable depth maps for most scenes in training and testing set.

There is certainly room for further improvements. Due to the long training time and resource required for each model (e.g. Drednet needs 8 GPUs and days to converge), we weren't able to experiment the effect of varying too many hyper-parameters as we'd like to but they should also be tried in the future. We are also considering turning off max-poolings since we may not actually want the translation invariance offered by it for this particular task, though this would require even more VRAM and GPUs to hold the network.

The relatively large difference between the training and testing error suggests overfitting of the training data. There are several ways to remedy this. One is to further downsample the output space or use smaller dictionary for sparse coding. Alternatively, we could also introduce more dropout and regularization in out networks. The effect of varying the objective function during training should also be further analyzed and tested with more trials on different models.

Acknowledgement

I'd like to thank Stanford's Artificial Intelligence Laboratory for providing me with the necessary machines and tools to carry out large-scale deep learning. Specifically, I'd like to thank Brody Huval for guiding me throughout the entire project. He has also helped pre-processing the NYU dataset, helping me setting up initial experiments and making necessary changes to the codebase for the depth prediction task. I'd also like to thank Adam Coates for continuously providing feedback on our results.

References

[1] B. Liu, et al. "Single image depth estimation from predicted semantic labels." *CVPR 2010*.

[2] A. Saxena, Ashutosh, et al. "Make3D: Depth Perception from a Single Still Image." *AAAI 2008*.

[3] A. Krizhevsky, et al. "Imagenet classification with deep convolutional neural networks." *NIPS 2012*.

[4] N. Silberman, Nathan, et al. "Indoor segmentation and support inference from RGBD images." *ECCV 2012*.

[5] R. Collobert and J. Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning." *ICML 2008*.

[6] Q.V. Le, et al. "Building high-level features using large scale unsupervised learning". *ICCV 2012*.

[7] A. Coates, et al. "Deep learning with COTS HPC systems." *ICML 2013*.

[8] Lee, Honglak, et al. "Efficient sparse coding algorithms." *NIPS 2006*.

[9] C. Szegedy, et al. "Deep Neural Networks for Object Detection." *NIPS 2013*.