Convolutional Neural Networks for Depth Estimation on 2D Images

Austin Sousa
asousa@stanford.edu

David Freese
dfreese@stanford.edu

Abstract

The human brain possesses the remarkable ability to infer depth when viewing a two-dimensional scene, even with a single-point measurement, as in viewing a photograph. However, accurate depth mapping from a single image remains a challenge in computer vision. Applications for single-image depth estimation extend from autonomous vehicles, to satellite data processing, to re-presenting two-dimensional movies in 3D. While in practice depth can be measured using stereoscopic imaging or laser rangefinding, in many cases additional measurements are unavailable. However, automated methods suffer from hardware and computation cost.

Humans can very-easily understand the 3D structure of a photographed scene, even without stereoscopic measurements (just close one eye). Recently Convolutional Neural Networks (CNNs) have made extensive progress in mimicking the human visual system, and can provide a means to infer depth cues within a single image.

Herein we attempt to implement the recent work by Fayao Liu et al[5], using a CNN with adjacency regularization.

1. Introduction

3D scene reconstruction is a problem that has been studied in depth over many years, however, it is a difficult problem because of the inherent ambiguity that exists in images due to the effects of camera projection. Each image could mathematically represent an infinite number of combinations of images along any linear scale of coordinates. The ability of humans to infer a depth scale within images is largely due to their ability to extract and weight different depth cues within the image [9], based predominately on prior knowledge of scales of objects within the image. This system is not something that is easily mimicked by algorithms, as they cannot currently mimic the learned object recognition and object relationships that the brain is able to do. However, there has been significant work to create approaches and appropriate assumptions to make reasonable inferences from a single monocular image. Liebowitz et al. studied this problem by assuming some degree of parallelism and orthogonality within architectural images to both calibrate the camera used to take the image as well as infer the relative scale of points within the image [4]. This approach allowed the authors to generate three dimensional models of architectural scenes where it is a reasonable to assume that the ground is orthogonal to the buildings within the image. Saxena et al. extended this concept to apply supervised learning for Markov Random Field (MRF) models of depth cues within images to infer depth within an image assuming that the image was composed of small, connected planes [8]. In this case, patches of the image were assumed to provide some information about the image, while neighboring regions and straight lines were assumed to provide information about the variation that is seen across the depth field.

With the introduction of CNNs to the computer vision field, they have begun to be applied outside the classification regime to continuous problems. A prominent work in this area was that completed by Eigen et al. [2]. This work used interleaved coarse and fine CNNs to provide a global estimate of the structure of the image that was then adapted by the properties of the local regions. Building off of this work, Liu et al. demonstrated work that is at a fundamental level, quite similar to that presented by Saxen et al. in terms of assuming connections between neighboring superpixels. In their work, Liu et al. demonstrate a method of training two networks to maximize a log likelihood function by minimizing two energy functions: one for the euclidian distance between the actual and predicted depth, and a second for the cost between neighboring superpixels based on an arbitrary number of metrics.

2. Theory and Architecture

We begin with the architecture proposed by Liu et al[5], wherein N_t training images are first segmented into N_p superpixels. A windowed region is then taken from around the centroid of each superpixel, creating N_p Patches. Each of these patches is labeled with a target depth. The total set of $(N_t N_p)$ patches comprises our training set.

The training set is then applied to a CNN, configured for regression using a Euclidean loss. This CNN is referred to as the *Unary* branch.

Simultaneously, the patches from each pair of adjacent superpixels are passed through a similarity function, referred to as the *Pairwise* branch.

The outputs of the unary and pairwise branches are then passed to a modified loss function, which encourages similar-looking adjacent patches to share similar predicted depths. This step is referred to as the *Graphcut*, or Continuous Random Fields (CRF) minimization block.

A schematic of the system used by [5] is shown in figure 1.

The Conditional Random Fields loss function used in [5] is given by:

$$-\,LogPr(y|x) = -log\frac{1}{Z(x)}exp\{-E(y,x)\} \tag{1}$$

where the energy function, E, is given by:

$$E(y,z) = \sum_p (y_p - z_p)^2 + \sum_{p,q} \lambda_{pq}(y_p - y_q)^2 \tag{2}$$

The first summation is the squared distance between the evaluated depths, y_p, and the true depths, z_p. The second summation is a measurement of the similarity between superpixel p and its adjacent superpixels, q. λ is a matrix (and is a learned parameter), which maps various similarity functions – euclidean distance by pixel, euclidean distance by color histogram, and binary pattern difference) – to a single similarity value. This parameter has the effect of encouraging transitions in depth to occur only at dissimilar pixels.

Of note is that, in the absence of the similarity sidechain ($\lambda = 0$), the energy function reduces to a simple Euclidean loss:

$$E(y,z) = \sum_p (y_p - z_p)^2 \tag{3}$$

Similarly, in the absence of multiple similarity metrics, ($\lambda = I$), the problem reduces to a conventional graphcut problem [1].

3. Implementation

Our intent was to develop this topology in the following phases:

1. Train the unary classifier, following the same topology as [5], and using transfer learning from PlacesCNN [10] and a Euclidean loss

2. Remove the Euclidean loss and summing nodes, and retrain the network using a 1-vs-all Softmax loss

3. Implement the fine-tuning routine using a standalone graphcut package [1] and a single similarity function

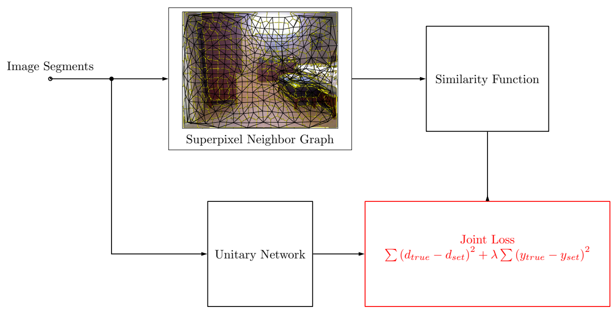


Image Segments
Superpixel Neighbor Graph
Similarity Function
Unitary Network
Joint Loss
$\sum(d_{true} - d_{set})^2 + \lambda\sum(y_{true} - y_{set})^2$

Figure 1: Block diagram of the depth regression system used by Liu et al [5]

4. Time-permitting, implement the graphcut loss and multiple-similarity sidechain of [5], and compare to single similarity results

The system was to be implemented using the Caffe [3] CNN environment, on a well-equipped PC running Ubuntu 14.04, and using an nvidia GTX670 GPU.

Two datasets are commonplace in the literature – the NYUv2 Indoor Scenes set [6] and the Make3D outdoor scenes set[7, 8]. NYUv2 includes 1,449 individual 640x480 photos, each with a depth map of the same size, and a standard training / test split. The set includes indoor scenes only, from apartments, offices, and classrooms. Make3D includes 534 images and depth maps of much-higher resolution (1704x2272), featuring outdoor scenes from around the Stanford campus. For both sets, ground truth was recorded using a Microsoft Kinect. We chose to work with the NYUv2 set initially, for two reasons – the data appeared to be more-easily preprocessed, and that the Make3D set truncates infinite depth to 81 meters. Using indoor scenes only eliminated the need to worry about infinite depth conditions.

3.1. Preprocessing

Prior to training the network within Caffe, there are several preprocessing steps, shown in Figure 2, that are required. These steps, which are described in detail later are segmentation of the image into superpixels, pixelation of the depth data on a log scale, and, finally, extraction of image patches around the centroid of each superpixel.

3.1.1 Segmentation

Segmenting each image was accomplished using the SLIC algorithm, as implemented in SKimage. For the training set, we segmented each initial 640x480 image into approximately 400 superpixels. The SLIC algorithm targets but does not guarantee a specific number of superpixels. As

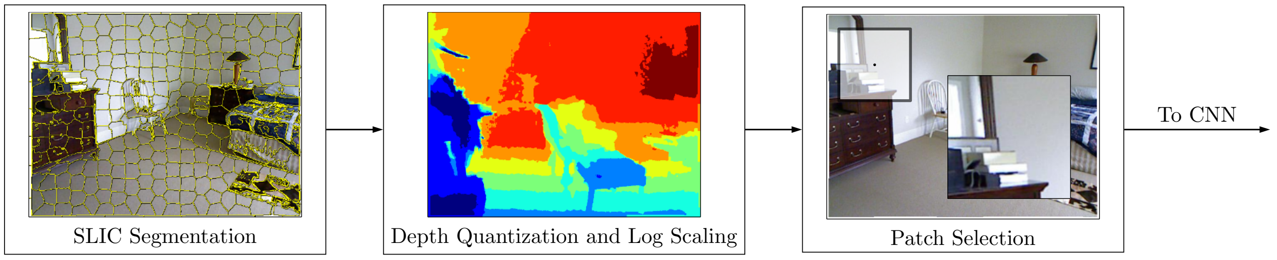


Figure 2: The preprocessing steps required to produce data necessary for training within Caffe

shown in [5], increasing the number of superpixels on training incurs a substantial computation penalty, with diminishing returns. Approximately 300 superpixels appeared to be a minimum for a visually-pleasing map. After segmentation, we select a square patch around the superpixel centroid. Following [5], this patch was sized to 167x167 pixels; however, time allowing, the patch size (in relation to test image size and number of superpixels) would be a revealing parameter to sweep.

3.1.2 Log Scaling

Each superpixel was paired to a single depth value to use as a ground truth. We examined three methods of selecting this value – the true depth at the superpixel centroid; the mean over the superpixel body; and the mean over the entire windowed patch. The resulting depth maps can be seen in Figure 3. The selected depth was then discretized using 16 logarithmically-spaced bins, spanning 0.7 to 10 meters. There were two motivations to discretizing the depth field. The first is that working in a log domain helps improve scale invariance, as is used in both [5] and [2]. The second, and more practical, motivation is that the Caffe data layer frameworks only accept integers for ground truth labels, despite being able to work internally as floats. The number of bins was selected to match the size of the Softmax nonlinearity within the unary CNN.

3.1.3 Set Normalization

Initial training using achieved a promising 30% accuracy on both the test and training sets. After visualizing the results on actual images, however, it was clear that the CNN learned to always select a single depth, in this case the peak of the distribution of depths, which can be seen in Figure 4. Following, we examined the distribution of depths within the training set, which were found to be highly nonuniform. Our solution was to create a new training set with optimally-distributed depth values, by finding the highest-occuring depth class, and filling in the remaining

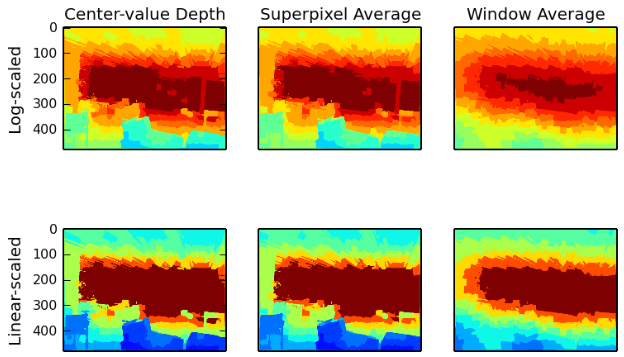


Figure 3: Log scaling and depth averaging. The left column shows the map using center-valued depth selection; the middle column via averaging over each superpixel; and the right, averaging over an entire window section.

depths with randomly-selected examples from their respective classes. Figure 4 shows the initial and final distributions of depth labels, as well as the equivalent number of epochs for each depth class (i.e. the average number of times the training images for each class are repeated).

Finally, each 167x167-pixel patch in the set is rescaled to fit the 227x227-pixel inputs of the pre-trained network (from PlacesCNN, itself a derivative of the standard ImageNet CNN [10]) The final training set comprises 310,845 unique patches, which after normalization result in 676,320 patches, each labeled with an integer depth from 1 to 16. The set occupies approximately 2 GB of hard drive space.

Figure 5a shows the full-frame mean image of the NYUv2 dataset; Figure 5c shows the mean image of the training set patches. Note the vertical and horizontal structures present in the mean image; these are due to a 10-pixel white border present in the NYUv2 image data, which assures each image has a constant framesize. While we would expect to see a diffuse field, this is in fact the mean of all patches that the CNN will see during training.

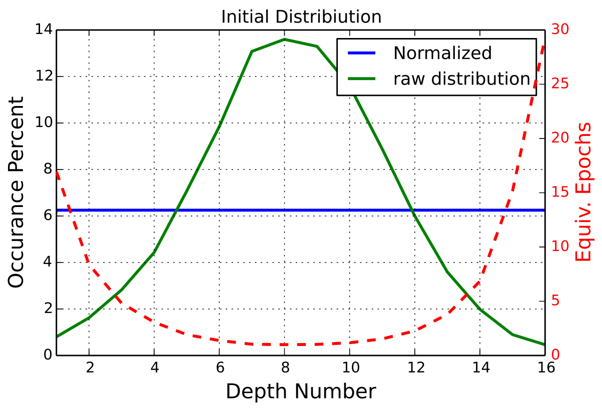
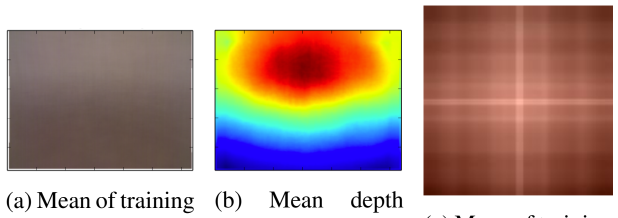


Figure 4: Training set distribution, before and after normalization. The red dashed line shows the equivalent epoch number for each epoch of the entire set.



(a) Mean of training images (b) Mean depth field (c) Mean of training patches

Figure 5: Mean images of the training set

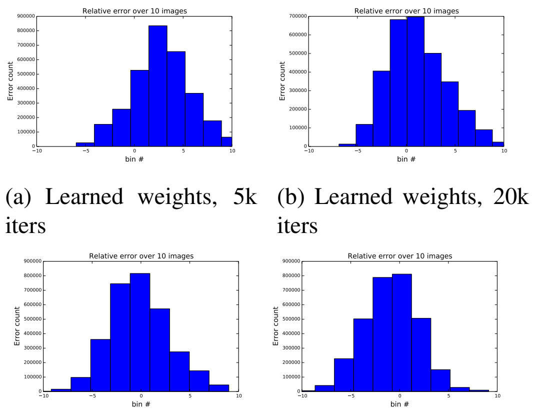
4. Training with Euclidean Loss

Figure 9 shows the topology of the unary CNN using a Euclidean loss. Note that the first five layers are pre-trained, and are used as a feature-extractor. The following layers are similar to that used by [5]; however we add an additional fully-connected layer, as in [10]. Each fully-connected layer is followed by a simple ReLU nonlinearity, and is trained with random dropout (50%). Finally, the output is passed through a Softmax nonlinearity, and is summed by the last layer, resulting in a single real-valued output for each patch at the input.

Figure 7 compares the training loss of this network, using center and superpixel-averaged depths. Note that the superpixel-averaged version trained more-quickly for the same training parameters (SGD solver, loss rate=0.001, step size =5000, gamma = 0.5), which we initially took to indicate a cleaner correlation between patch and depth.

However, figure 8 shows a typical depth prediction from the convnet, trained for 1 epoch (5000 iterations), on center-depth versus superpixel-averaged training sets. Note that the center depth version exhibits far greater variance in depth, which indicates that perhaps depth averaging occludes key features.

When using a Euclidean loss, Caffe offers no provision for automatic training / test accuracy. In order to quantify the error beyond simple visual inspection, we classify 10 randomly-selected images from the test set, and inspect the difference between target and predicted depths. Figure 6 shows a typical distribution of errors using the center-depth Euclidean CNN. While the fractional error is decent, one glaring issue is is noticeable in the results: the classifier generally predicts differences in depth in the appropriate regions, but frequently fails to get the correct polarity – that is, far regions are mapped to near regions, and similarly nearby regions are mapped to further distances. This is likely due to the sign-invariance of the Euclidean loss: $(y_p - y_t)^2$.



(a) Learned weights, 5k iters (b) Learned weights, 20k iters

(c) Assigned weights, 5k iters (d) Assigned weights, 20k iters

Figure 6: Error histograms for the Euclidean regressor 6a, 6b: using the learned weights (blue trace in figure 10), and 6c, 6d, with the hardcoded weights.

4.0.4 Summing-layer Weights

Our first step in examining the performance of the classifier was to visualize the weights of the output stage. Our intuition was that the Softmax nonlinearity, which outputs probabilities ranging from 0 to 1, should operate as a multiple classifier - that is, channel 5 should have high probability on depths of 5, and zero elsewhere. In this case, the ideal output weights would be integer values from 1 to 16. However, the network instead learned only two nonzero parameters, approximately equal in polarity, as shown in figure 10. With these weights sent through the Euclidean loss layer, the network is essentially taking the difference in probability of just two channels, while ignoring contributions from the remaining 14.

We made several attempts to modify the Euclidean network, including presetting the output weights to the expected ramp; using a different activation; and using no acti-

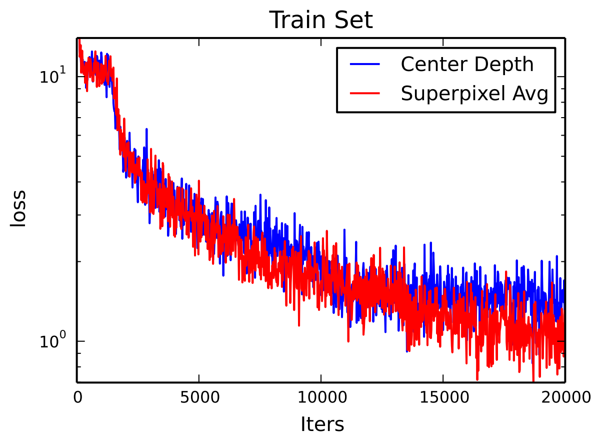


Figure 7: Training loss, using a Euclidean loss function, using two different methods of calculating the ground truth (center pixel value vs. superpixel average)

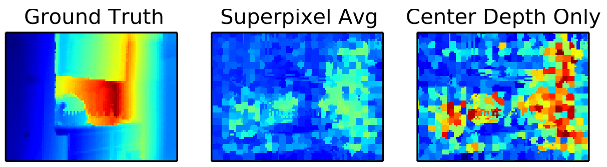


Figure 8: Example predictions using the Euclidean-loss CNN after 5000 iterations. Left is the true depth; the center column was trained on average depths across each superpixel; the right was trained on the center depth only.

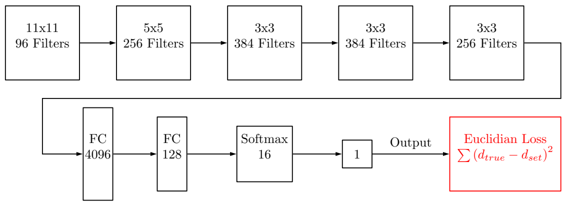


Figure 9: Block diagram of the unary CNN, using a Euclidean loss.

vation function at all. In all three cases, performance using the Euclidean loss was similar – the classifier generally selected unique regions in the appropriate places, but failed to accurately estimate the true depth.

5. Softmax and 1-vs-all Classification

We next considered re-training the CNN as a multiple classifier, rather than a regressor. In this form, the last sum-

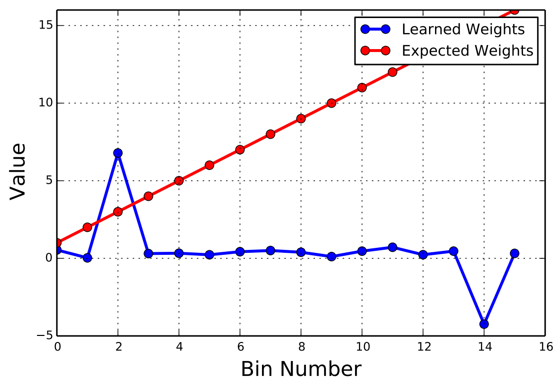


Figure 10: Learned vs Expected weights of the summing layer, using a Euclidean loss.

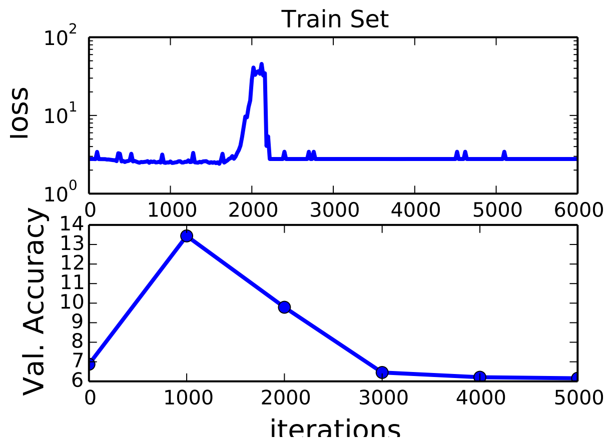


Figure 11: Poor learning performance using a Softmax multiple classifer loss function. Note the instability and subsequent flatlining of the loss function at 2000 iterations

ming layer was removed, resulting in 16 outputs, ranging from 0 to 1, for each test or train patch. We were optimistic about this architecture for two primary reasons – one, that Caffe (and CNNs in general) are better-suited to classification tasks than regression tasks; and two, our selected graphcut package, PyGCO, required marginal probabilities for each prediction.

Unfortunately, we never achieved a stable classifier. Figure 11 shows a typical training run, using center-value depths, 16 depth bins, and a Softmax loss. In this case the solver went unstable, before asymptoting to a constant value 2.77. Despite trying different SGD parameters, and even exploring ADAGRAD and Nesterov solvers, this architecture would never achieve performance greater than 14%, or two times the nominal random performance.

6. Further Work

At this stage, without a reliable multiclass CNN, we have found it difficult to continue and implement the graphcut routine. Our current implementation fails to meet the baseline performance of [5] and [2]; so much so that we believe there are fundamental errors within our preprocessing system. Time-permitted, there are several points remaining to consider:

1. Investigate the effect of window sizing, as well as data augmentation in training
2. Confirm that there is a human-recognizable correlation, albeit likely small, between our selected patches and the true depth
3. Confirm that the Caffe package is correctly-handling preprocessing steps – mean subtraction, channel flipping, etc.
4. Further investigate different loss functions and nonlinearites
5. Develop a more-robust and automated error analysis system, to better-quantify performance beyond human inspection

Regrettably, much of our time was spent configuring a GPU-processing environment, and learning to work with the Caffe framework, which is currently in its infancy. At this point we have developed the infrastructure necessary to experiment, only to have reached our time limit. In hindsight, working with a less-encapsulated CNN framework such as MatConvNet would have been beneficial, as it would have required far less data preparation as with Python and Caffe.

7. Concluding Remarks

At the completion of this stage of this project, several points come to mind: First, that depth prediction is an inherently ill-posed problem – that is, while a classifying CNN such as AlexNet is designed to detect the presence of an object (cat / not cat) etc, and PlacesCNN is designed to recognize a scene (park, church, cafe), depth prediction is far less-defined. An object seen by a neural net could be located at several spaces within the depth plane, and it is unsettling, at the least, to hope that a trained CNN will learn to recognize depth cues – shadows, horizon lines, etc – rather than the presence of objects or textures indicating depth. For example, tables and chairs were frequently mapped to a constant depth, which matched that of the hardwood floors also found within our training set. Similarly, flat black objects such as televisions and computer monitors were frequently mapped to the furthest distance, possibly due to our zero-padding step in preprocessing.

However, the architecture of using a primary CNN, combined with a sidechain similarity network and graphcut routine is compelling for many problems of field extrapolation - for instance, colorization of black-and-white images, or inferring temperature or other such parameters from color images. Once implemented, it would be a rewarding exercise to apply this architecture to these problems.

Our project code, models, and example results are available on GitHub: `github.com/asousa/DepthPrediction/`

References

[1] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions of Pattern Analysis and Machine Intelligence (PAMI)*, 23(11):1222–1239, 2001.

[2] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *CoRR*, abs/1406.2283, 2014.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[4] D. Liebowitz, A. Criminisi, and A. Zisserman. Creating architectural models from images. In *Annual Conference of the European Association for Computer Graphics (Eurographics)*, volume 18, pages 39–50, 1999.

[5] F. Liu, C. Shen, and G. Lin. Deep convolutional neural fields for depth estimation from a single image. *Journal of Foo*, 14(1), 2014.

[6] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[7] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *NIPS*, http://www.cs.cornell.edu/ asaxena/learningdepth/, 2005.

[8] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Transactions of Pattern Analysis and Machine Intelligence (PAMI)*, 30(5):824–840, 2009.

[9] M. J. Young, M. S. Landy, and L. T. Maloney. A perturbation analysis of depth perception from combinations of texture and motion cues. *Vision Research*, 33(18):2685–2696, 12 1993.

[10] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. *Advances in Neural Information Processing Systems (NIPS)*, 27, 2014.