Fully Convolutional Network for Depth Estimation and Semantic Segmentation

Yokila Arora [*]
ICME
Stanford University
yarora@stanford.edu

Ishan Patil [*]
Department of Electrical Engineering
Stanford University
iapatil@stanford.edu

Thao Nguyen
Department of Computer Science
Stanford University
thao2605@stanford.edu

Abstract

Scene understanding is an active area of research in computer vision that encompasses several different problems. This paper addresses two of those problems: we use a fully convolutional network architecture to perform, in a single pass, both depth prediction of a scene from a single monocular image, and pixel-wise semantic labeling using the same image input and its depth information. We optimize the first task on L2 and berHu loss, and the latter on negative log likelihood loss per pixel. Our model encompasses residual blocks and efficient up-sampling units to provide high-resolution outputs, thus removing the need for post-processing steps. We achieve reasonable validation accuracies of 49% and 66% in the semantic labeling task, when using 38 and 6 classes respectively.

1. Introduction

Predicting depth is crucial to understanding the physical geometry of a scene. The more challenging problem is learning this geometry from a single monocular image in the absence of any environmental assumptions, due to the ambiguity of mapping color intensity or illumination to a depth value. Developing an accurate real-time network for generating pixelwise depth regression is an ill-posed task, but a crucial one for automated systems where depth sensing is not available. In addition, other tasks in computer vision can also benefit greatly from having depth information, as we have shown with semantic segmentation in our project.

We adapted our model from the one proposed by Laina et al. [10] and implemented a joint architecture in PyTorch for both depth estimation and semantic segmentation tasks. The inputs to our model consist of RGB-D images from the NYU Depth v2 dataset and their corresponding ground-truth depth maps, whereas the outputs contain a predicted depth map and semantic labels (for 6 and 38 most frequent labels in the aforementioned dataset) for each input image. The model is able to learn directly from data without any specific scene-dependent knowledge. The absence of post-processing steps and fully connected layers helps reduce the number of parameters of our model as well as the number of training examples required, while still ensuring reasonable performance. We also combine the concepts of up-convolution and residual learning to create up-projection units that allow more efficient up-sampling of feature maps, which are essential to increasing the resolution as well as accuracy of the output image. In this paper, we will analyze the influence of different variables (loss function, learning rates, etc.) on the performance of the model, in addition to the results it generates on standard benchmarks.

2. Related Work

Convolutional Neural Networks (CNNs) are being widely used for tasks like image recognition, object classification and natural language processing. Eigen et al. [3] were the first to use CNNs for depth estimation. They present a multi-scale deep network, which first predicts a coarse global output and then a finer local network. A recent paper by Eigen et al. [2] extends this model on two other tasks, namely surface normal estimation and semantic labeling, and achieves state-of-the-art results on all three tasks. They develop a general network model using a sequence of three-scales, based on AlexNet [9] and the Oxford VGG network [14].

In [4], Farabet et al. propose a multi-scale convolutional network for scene labeling from the raw input images by

[*]Equal contribution

classifying into regions centered around each pixel. These regions of multiple sizes are encoded in the form of dense feature vectors, which are then used to obtain labels using post processing techniques (like superpixels and CRFs). Couprie et al. [1] use a similar architecture to perform multi-class segmentation, but using both input (RGB) image and corresponding depth (D) image.

Laina et al. [10] explore different CNN networks (AlexNet, VGG-16 and ResNet) for the down-sampling part and try to improve the receptive field. They develop a network based on ResNet-50 [7] which makes the network deeper without vanishing gradient problem, making use of residual and skip layers as well as batch normalization. Further, they apply up-convolutional blocks to improve accuracy. Their model attains state-of-the-art results on all models for depth estimation.

Considering these architectures, we also use CNN for both depth estimation and semantic labeling tasks. Furthermore, we seek to reduce the number of parameters and post-processing steps required by gradually down-sampling the input (using the ResNet architecture), followed by multiple rounds of efficient up-convolution, to output depth predictions in a single pass through the network. The latter is inspired by Laina et al. [10]. We extend their model for depth estimation to perform the task of semantic labeling as well. As done by Eigen et al. [2] and Couprie et al. [1], we use both input (RGB) and ground truth depth (D) images for the latter task.

3. Dataset

We evaluate our model on a subset of NYU Depth v2 dataset [11], one of the largest RGB-D datasets for indoor scene reconstruction. The images in the dataset represent environments that we often interact with in everyday life - scenes of offices, stores, rooms of houses - most of which have uneven lighting. There are 1449 scenes in total with 1449 corresponding ground-truth depth maps and 894 different object classes. However, related works often focus on only a subset of those classes. As mentioned in [2], there are three standard subsets of label classes, namely 4, 14 and 40, that are used for the task of semantic labeling. The 4-class labels defined by Silberman et al. [13], give a high-level categorization of labels into the sets "floor", "structure", "furniture" and "props"". On the other hand, Couprie et al. [1] use the 13-class labels for segmentation. The more common approach now is to use the most dominant 40-class labels, as described by Gupta et al. [6], which include more abstract objects like "wall", "floor", "desk", "book shelf", etc. We use the standard 40-class labels, but instead of the 3 categories "other furniture", "other props", "other struct", we group all other labels into one class which we call "other". Along with these 38-labels, we also test our model on the most frequent 6-class labels, namely "wall", "floor", "bed", "chair", "cabinet" and others.

4. Methodology

In this section, we describe in detail the different components of our network, starting with the overall architecture, to the additional components that help to boost efficiency, ending with our choice of loss function.

4.1. Depth Estimation

4.1.1 Convolutional Neural Network

We use a single CNN architecture to output depth estimation, adapted from [10], with the hypothesis that having combined information from neighboring pixels in the same local region is useful for the task of depth prediction. Our network uses transfer learning, keeping most of the downsampling and early up-convolutional layers fixed. The weights for those layers are taken from a pretrained TensorFlow model provided in [10].

At the start, a series of convolution and pooling layers reduce the resolution but at the same time capture global information such as edges, corners, blobs etc. Near the end of the network where we want to transition to a high-resolution output, we replace the typical fully connected layers as used in the model by Eigen et. al. [2], [3], which are expensive in terms of number of parameters, with more efficient residual up-sampling blocks as elaborated in the following section, and end with up-sampling the feature map using bilinear interpolation to scale the feature map to the size of ground truth depth map in order to calculate regression loss.

In addition, our approach relies heavily on residual learning, particularly through the skip and projection layers. Therefore, we are able to build a deep network relatively unaffected by the vanishing gradient problem. As seen in the top half of the architecture in Figure 3, our model accepts input size $304 \times 228 \times 3$ and output prediction map of size 640×480 .

4.1.2 Up-projection

Unpooling layers, such as by mapping each entry value into the top left corner of a 2×2 kernel (with other values in it being zeros) followed by a 5×5 convolution (as illustrated at the top of Figure 1), is a standard way of increasing the size of the inputs, doubling in this case. However, that means a lot of computation would deal with zero values in the up-sampled feature map. Up-projection block (Figure 2) addresses this issue by breaking the 5×5 convolution into smaller filters with receptive fields of sizes 3×3, 3×2, 2×3 and 2×2. The elements of the four resulting filter maps undergo interleaving (the bottom of Figure 1) to form the final output of this block. (Unlike TensorFlow, PyTorch

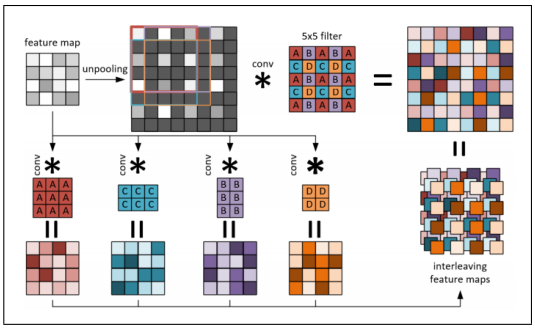


Figure 1. More efficient up-convolution using 4 different filters and interleaving [10]

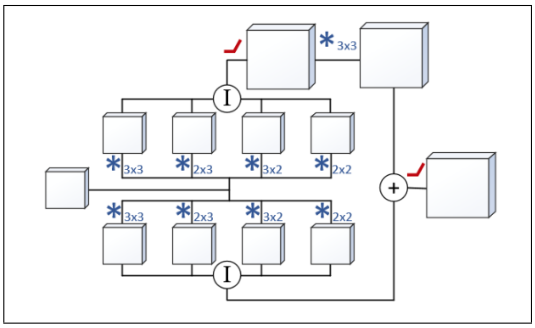


Figure 2. Fast up-projection layer [10]

does not have any built-in function for interleaving, thus we had to construct our own method to do so.)

4.1.3 Loss Function

Regression problems such as depth estimation can make use of the standard L2 loss function for optimization. However, the reverse Huber (berHu) loss function as proposed by [15] and also employed by the initial paper [10] has been proven to have several advantages over L2. It is defined as-

$$B(x) = \begin{cases} |x|, & \text{if } |x| \leq c \\ \frac{x^2+c^2}{2c}, & \text{otherwise } |x| > c \end{cases}$$

BerHu loss provides flexibility in handling two different types of pixel values present in the outputs: those with small residual have greater impacts on the final loss (due to the absence of division by 2c), while those with large residual account for the L2 terms in the final loss and thus produce large gradient updates for the corresponding weights.

4.2. Semantic Segmentation

4.2.1 Network Architecture

For this task, similar to Section 4.1.1, we use a fully convolutional network, but pass both RGB images and their corresponding ground-truth depth maps as input to the network. For each of these depth maps, we rescale it from the default size of 640×480 to 304×228, and create a channel dimension where we copy the depth information thrice to form a $304 \times 228 \times 3$ image for input.

For each training example, we first obtain two separate feature maps (each of size $160 \times 128 \times 64$) from the fourth (last) up-projection layer of the model in [10], shown in Figure 8, one produced from the input RGB image and the other produced from the ground truth depth map. Concatenating these two feature maps along the channel dimension gives a $160 \times 128 \times 128$ output which is in turn fed into an up-convolution block which doubles the spatial dimensions while reducing the number of channels by half, giving a $320 \times 256 \times 64$ output. We then pass this through 2 more convolutional filters, reducing the number of channels gradually to 38, in the case that we want to predict among 38 class labels, or 6 if we just want to predict among 6 class labels. The up-sampling layer (bilinear interpolation) is used to resize the resulting feature map to a $640 \times 480 \times 38$ (or $640 \times 480 \times 6$) final output. The full network architecture is as depicted in Figure 3.

4.2.2 Segmentation Loss

We use pixel-wise softmax classifier to predict a class label for each pixel for semantic labeling, as done by Eigen et al. [2]. The pixel-wise softmax classifier is based on the negative log likelihood, which is defined as:

$$L(C, C^*) = -\frac{1}{n}\sum_i C_i^* log(C_i)$$

where C is softmax of the predicted semantic label map (i.e. $C_i = \frac{e^{z_i}}{\sum_C e^{z_{i,c}}}$) and C^* is the ground truth label.

The tensor output from the up-sampling layer ($640 \times 480 \times 38$), after applying softmax activation, is fed as input to the 2d negative log likelihood loss function, which compares these 38 (or 6) "one-hot" feature maps to a map that has ground truth target labels ($0 \leq$ labels $< c$, where c is the number of classes being considered) for each pixel.

4.3. Transfer Learning

One of the reasons why the semantic segmentation task is somewhat easier than the depth estimation one is that the softmax loss is much more stable and easier to optimize than the L2 or berHu Loss. Therefore, due to time constraint, we just predict the depth map from Laina et al.'s model [10] by using their pre-trained weights up to the last up-projection layer. After that stage we split the model into two branches, one that predicts the depth (and continues using the pre-trained weights of Laina et al.'s model) and another that predicts the semantic labels (developed and trained by us using transfer learning), as shown in Figure

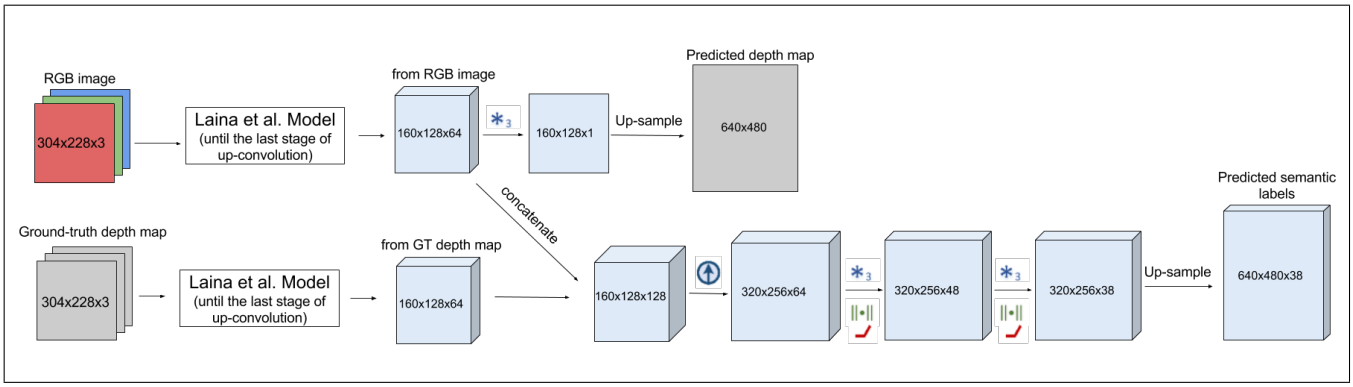


Figure 3. Network architecture for 38 class labels

3. It should be noted that the RGB images and the corresponding ground-truth depth maps are passed through two identical but different modules of Laina et al.'s model and hence the depth prediction task doesn't take any input information from the ground-truth depth maps, but the semantic segmentation task incorporates information from both inputs.

5. Experimental Setup

5.1. Model conversion

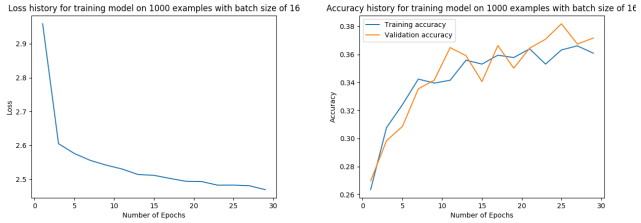
Laina et al.'s [10] model and the corresponding pre-trained weights are available in Tensorflow and MatConvNet frameworks. Since we choose PyTorch as our framework of choice, we decide to implement their entire model in PyTorch. One of the challenges we face while performing this model conversion is writing an implementation of "interleaving" feature maps in the up-projection block, which has been explained in Section 4.1.2. The original implementation uses TensorFlow's *dynamic_stitch* method to achieve this directly but PyTorch doesn't have any equivalent functionality. Thus, we code up this function and to check that it works, we load the pre-trained weights into our model and feed input images to acquire corresponding depth maps.

5.2. Data Augmentation

To avoid overfitting the model to the training images, we also implement Data Augmentation which includes random flips, random crops, scaling and random rotations. Since PyTorch, as of now, doesn't support co-transforms, i.e. transforms that are applied simultaneously to both input and target images, we include code to do that from [12].

5.3. Training

Initially, we plan to replicate results from Laina et al. [10] by transfer learning weights for the up-projection blocks and freezing ResNet pre-trained weights for the down-sampling part of their network. However, we face problems in optimizing both the L2 and berHu regression loss and since we are not able to learn effectively from just a small subset of the NYU Depth Dataset, given that the authors in [10] use 9k examples and [2],[3] use almost 95k, 120k respectively for this task. Thus we decide to explore the potential of this model for semantic segmentation, a task that has not been explored by the authors themselves. Given the pre-trained weights for the depth estimation task, and our new branch for semantic segmentation (as described in Section 4.2.1), we apply transfer learning on this combined model with loss function and ground truth target labels for each pixel of each of the training images as mentioned in Section 4.2.2. We split our dataset of 1449 images, into 1000 training, 300 validation and 149 test images. Optimal learning rate (0.05) is found by first over-fitting the model on just 20 examples, and the learning rate is decayed by a factor of 0.5 whenever the loss starts to plateau. Final training is done on 25 epochs on 1000 examples with batch size of 16 on NVIDIA Tesla K80 GPU. After that, we also fine-tune the entire model (by allowing backpropogation through all model parameters) to increase the accuracy by $2-3\%$ [[8]].



(a) (b)

Figure 4. (a) Loss curve, (b) Accuracy curves for transfer learning semantic labels for 38 classes

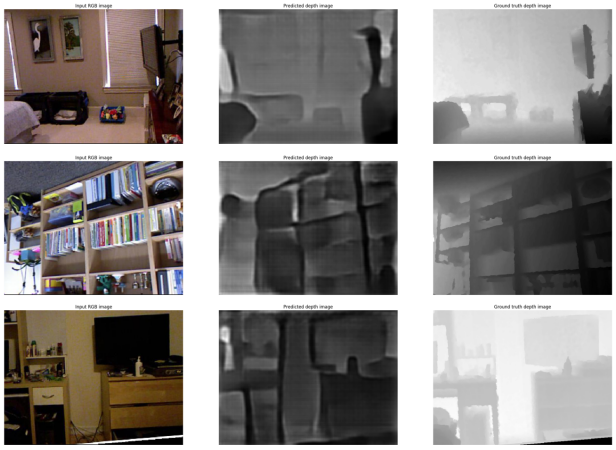


Figure 5. Comparison of input RGB images, predicted depth images and ground-truth depth maps. The three samples consist of, in order, a standard image, a flipped image and a rotated image.

6. Results

6.1. Depth Estimation

Since we haven't explicitly trained our model for this task but directly use pre-trained weights, for now, we will just visualize out results and show how they compare to the ground truth. As a quantitative evaluation of error, we provide Scale-Invariant Error for each prediction, which is defined as:

$$D(y, y^*) = \frac{1}{2n} \sum_{i=1}^{n} (\log y_i - \log y_i^* + \alpha(y, y^*))^2$$

where y is the predicted depth map and y^* is the ground truth, each with n pixels indexed by i, and $\alpha(y, y^*) = \frac{1}{n} \sum_i (\log y_i^* - \log y_i)$ is the value of α that minimizes the error for a given (y, y).

Those results actually help verify our correct implementation of model conversion from TensorFlow into PyTorch, particularly for the "interleaving" module as mentioned in Section 5.1.

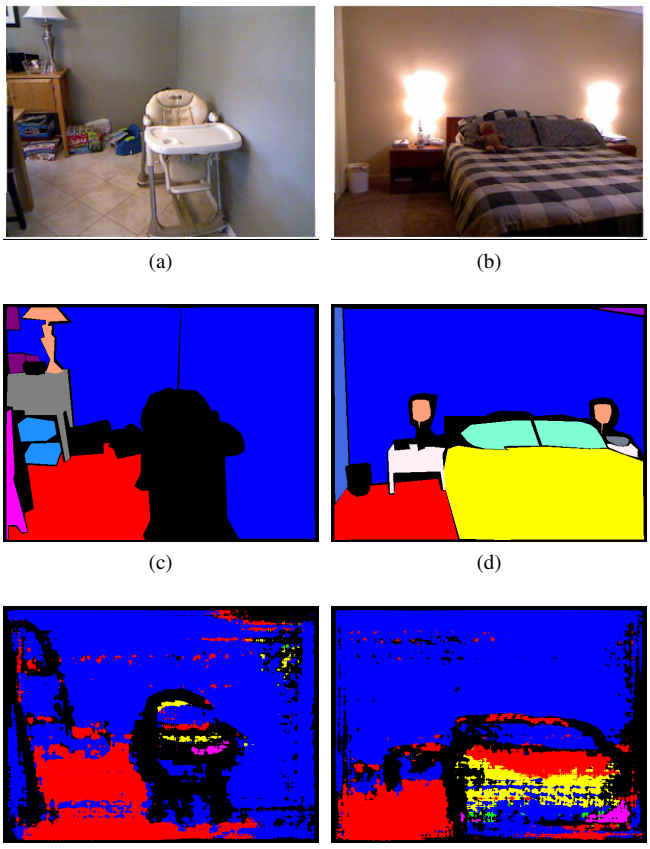
Figure 5 shows 3 sample input RGB images, corresponding predicted depth images and the ground-truth depth maps from the dataset.

6.2. Semantic Segmentation

As explained earlier, we perform semantic segmentation task on 6-class and 38-class labels, and compute the negative log likelihood loss (Section 4.2.2). The starting value of this loss for 6-class prediction was close to $log_e(6)$ and that for 38-class prediction was close to $log_e(38)$, as expected.

For our evaluation, the error metric on class predictions considered is per pixel class prediction accuracy. We show both our training and validation accuracy plots as well as the loss curve as a function of the number of epochs. For the experiments that we have done until now, we have been able to achieve 40.96% and 66.91% accuracy on the validation set, as shown in Table 1. Even though this is less than what was achieved by [2] on a different model but for the same two tasks, it should be noted that the authors in that paper have trained the model on 95k input images as compared to our transfer learning based approach. Also, the 4-class labels considered in [2] are different than ours. While we consider the dominant 6-classes, they have grouped all 894 object labels into 4 broad categories.

Figure 6 shows our results for semantic segmentation on 38 class labels. Figure 7 shows results for 6 class labels. Some quick observations can be made such as the artifact due to the upsampling layer, and prediction of the "wall" class almost correctly owing to its highest frequency of oc-



(a) (b) (c) (d) (e) (f)

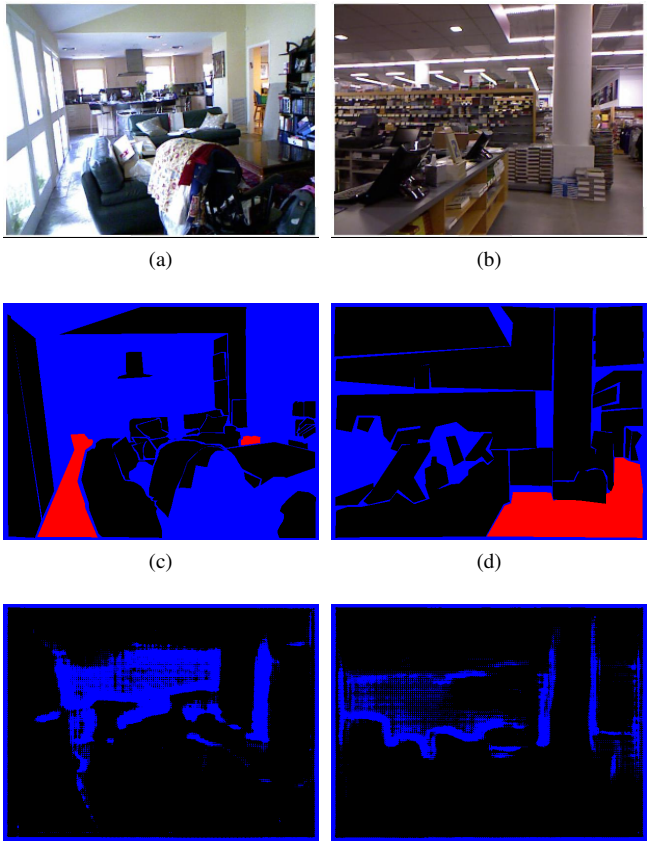
Figure 6. (a,b) Sample RGB images, (c,d) the corresponding colored labeled images and (e,f) predicted labeled images, for 38-class labels, from the NYU Depth Dataset [11]

Our model	Val. Acc.	Loss
6-class	66.91%	1.17
38-class	40.96%	2.44

Table 1. Experimental results

currence, even more than the "unknown" class, in the labeled dataset.

For now, we conclude that our model lacks the complexity required to achieve values of accuracy higher than these ones since the loss doesn't decrease more than 2.44 from an initial starting value of $log_e(38) = 3.637$ and 1.17 from an initial starting value of $log_e(6) = 1.79$, for 38 and 6 classes prediction respectively. Another fact which bolsters this conclusion is the higher value of validation accuracy compared to the training accuracy as seen in Figure 4.



(a) (b) (c) (d) (e) (f)

Figure 7. (a,b) Sample RGB images, (c,d) the corresponding colored labeled images and (e,f) predicted labeled images, for 6-class labels, from the NYU Depth Dataset [11]

7. Conclusion & Future Work

Overall we demonstrate how a fully convolutional architecture based on residual learning removes the need for fully connected layers while still giving highly accurate depth estimation, and how depth information can be an useful feature for other computer vision tasks such as semantic segmentation. The architecture used in this paper has many advantages: residual and skip layers that capture global information and pass it on much further down in the network, efficient up-sampling blocks that increases output accuracy, training with relatively few examples and parameters, and a single-scale network unlike other related approaches that require post-processing to refine coarser predictions.

Given more time and resources, we would want to explore how depth information can potentially help improve the performance of other computer vision tasks besides segmentation, such as object detection. Another option is to alter the number of layers - increasing the number of up-convolutions for instance - and see how the accuracy of the predictions is affected. We can also try to apply our existing model to other datasets with different characteristics, an example being KITTI [5] for outdoor scenes.

8. Appendix

Github repo to our PyTorch implementation (https://github.com/iapatil/depth-semantic-fully-conv)
Figure 4 shows the full convolutional residual network used in [10].

References

[1] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*, 2013.

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

[3] 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.

[4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2013.

[5] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[6] S. Gupta, P. Arbelaez, and J. Malik. Perceptual organization and recognition of indoor scenes from rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 564–571, 2013.

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

[8] J. Johnson. `https://gist.github.com/jcjohnson/6e41e8512c17eae5da50aebef3378a4c`.

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

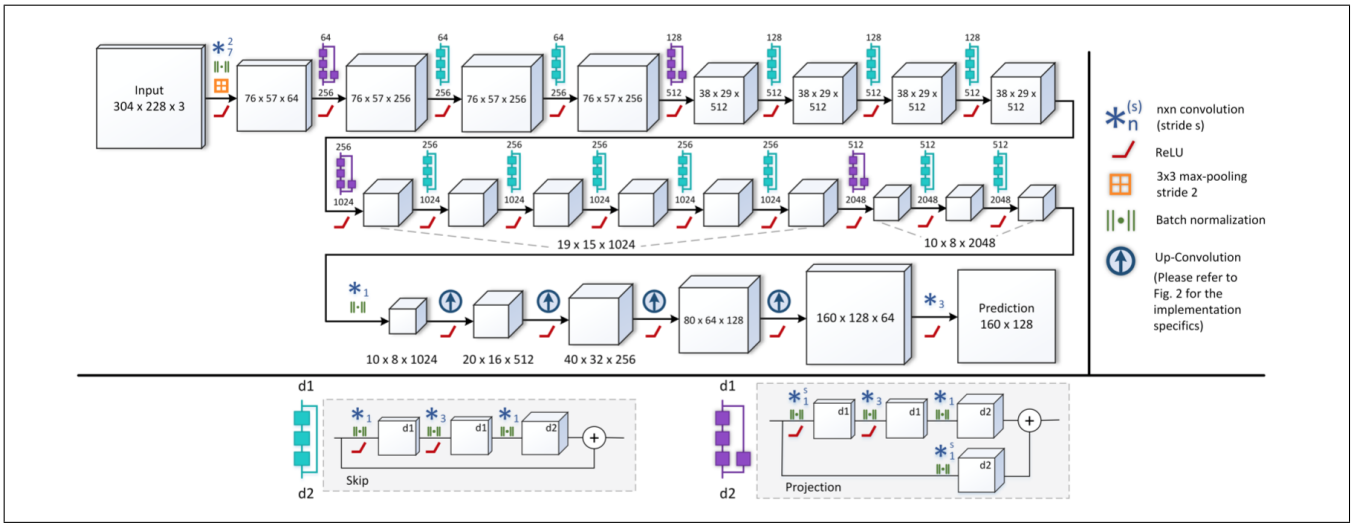


Figure 8. Network structure proposed by [10]

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

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

[12] C. Pinard. Flownetpytorch. `https://github.com/ClementPinard/FlowNetPytorch`, 2017.

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

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

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