Camera-Independent Monocular Depth Prediction using Metric Space Inputs

Jeshwanth Pilla

Advisor: Helisa Dhamo
Supervisor: PD Dr. Federico Tombari

Chair for Computer Aided Medical Procedures
Department of Informatics
Technical University of Munich

Abstract. Depth Prediction from a single RGB image suffers from the problem that a network trained on images from one camera does not generalize well to images taken with a different camera model. Hence, in this work, we try to propose a new approach that applies metric space transformation to image pixels before training and thereby considering of camera parameters of the dataset. Our experiments show that neural networks tend to perform better if metric coordinates are known compared to one unknown, indicating the improvement in networks generalizing to images of different camera models.

1 Introduction

Depth Estimation is a computer vision task in which we try to estimate depth from a 2D image. It usually involves a single RGB image as input and produces depth map as output. This has been a very active area of research since it could help in the improvement of many other computer vision tasks. Some of these tasks are reconstruction of indoor scenes for better understanding of structured interpretation from 3D cues [1], recognition tasks like RGB-D indoor scene labelling [2], semantic segmentation tasks like semantic labelling of indoor scenes without low-level segmentation or superpixels [3], pose estimation tasks like predicting dense correspondences for one-shot human pose estimation [4]. The real-world applications of these methods could be in self-driving cars [5], grasping in robotics [6], robot-assisted surgery, automatic 2D-to-3D conversion in film [7], and shadow mapping in 3D computer graphics [8]. It also helps in situations where direct depth sensing is not available. To compute the depth estimation, typically we rely on traditional methods like Structure-from-Motion (SfM)[9], Markov Random Field (MRF)[10] and Superpixel methods [11][12].

Recently, deep neural networks have led to significant progress, but such methods are yet to broadly impact depth perception from a single image taken in unconstrained settings. Multiple views of a scene can also be used as an implicit source of training data but may not always yield high-quality learned depth.

The RGBD-sensor cameras like Microsoft Kinect which provide both colour and depth information are used to collect the images for RGB-D datasets and training on these images is assumed to generalize well to other RGB-D camera images. But the experiment results from the recent neural network methods show that they perform well because of the usage of a test set from the images with the same camera as training data. This indicates the implicit camera dependency of neural network models and a strong bias in the benchmark results of the recent state of art methods.

Therefore, we present an approach addressing this camera dependency by making use of intrinsic parameters of the camera. We show that by concatenating the metric coordinates to the RGB images it enhances the generalization capability of neural networks by helping them to learn the dependency of depth from the camera parameters. This resulted in performance improvement of the neural network model compared to the baseline model in the current state of art methods.

2 Related Work

Some of the recent approaches use multi-scale deep convolution neural networks (CNN) that produces a pixel-wise prediction of a depth map from a single RGB image. Often these networks are initialized with models like AlexNet, ResNet etc. pre-trained on ImageNet dataset. It all started with work of Eigen et al. [13] where they used two deep networks i.e. a coarse-scale network that predicts the depth of scene at a global level and a fine-scale network for refining it locally at a finer resolution.

The other work by Laina et al. [14] which improves over the typically fully connected layers with a single-scale fully convolutional model involving residual up-sampling blocks to reduce the total number of model parameters. This followed by Liu et al. work [15] where they combined CNN with the continuous conditional random field (CRF) to model the relations of neighbouring super-pixels and learn the unary and pairwise potentials, but it does not exploit any geometric cues.These networks are generally trained on NYU [16] or KITTI [17] datasets which usually required to train on the particular dataset to achieve this state-of-art performance.

In parallel to supervised learning techniques, unsupervised image-to-depth learning approaches are also being proposed, where the only supervision is obtained from a monocular video. Zhou et al.[18] proposed the first fully differentiable deep neural network approach for unsupervised learning of depth and ego-motion and showed it outperforms prior approaches which used depth sensors as supervision. Casser et al.[19] also presents a similar approach for estimating monocular depth and camera motion which explicitly models 3D motions of moving objects, and adapts to new environments by learning with an online refinement of multiple frames.

Chen et al. [20] have introduced a new dataset consisting of images in the wild annotated with relative depth and claimed to improve single-image depth

perception in the wild through these annotations. This work is inspired from the issue of current RGB-D datasets that are collected in constrained settings and with limitation of scene diversity. Li and Snavely also have tried introducing a large depth dataset called MegaDepth [21] which is generated from Internet photo collections via SfM and multi-view stereo (MVS) methods.

Though recent internet derived large scale wild datasets helped in achieving better generalization, they are also biased towards the outdoor landmarks. There is a loss in accuracy during the testing time because CNN implicitly learns the camera. Intrinsic parameters are usually not made use of, in these deep learning pipelines. Our work is primarily to address this issue. While we are working on this, another work by Facil et al. came out called CAM-Convs [22] which also aimed to use these camera parameters to allow networks to learn calibration-aware patterns. Though it was in the same direction, the implementation is different from ours. It involved precomputing pixel-wise centred coordinates and field-of-view maps and feed them along with input features with focal length normalization. From their experiments, they pointed out that current state-of-art methods have degraded performance when trained with images from different cameras. This reinforces our assumption about the camera dependency and idea of using camera intrinsic parameters with image features help in generalizing to images of different cameras.

3 Methodology

Our work is an extension to the existing state-of-art methods for Depth Prediction. We like to propose a new approach that applies metric space transformation to image pixels and train an existing fully Convolution Neural Network model on a widely used depth dataset. And then we try to compare the evaluation results of this proposed approach over the one without this transformation. So we picked Laina et al[14] network for this implementation and the network architecture is explained in detail in section 3.1. And we have explained our metric transformation approach in section 3.2

3.1 CNN Architecture

Our work is heavily reliant on Laina et al work [14] because we have used the same CNN architecture along with our idea of metric space transformation to image pixels.

We have set the input size of 304 x 228 pixels and predict an output map that will be at approximately half the input resolution i.e. 160 x 128. We initialized with ResNet[23] weights pre-trained on the ImageNet dataset for facilitating better convergence. ResNet[23] involves skip layers that help with the issue of vanishing gradient and it also has a large receptive field which helps in capturing the input image of higher resolutions compared to AlexNet or VGG.

This is a fully convolutional network with a series of convolutions and pooling layers which helps higher-level neurons to capture more global information. This

network also involves the Up-Convolution Blocks i.e. unpooling layers which are followed by a convolution and successively by ReLU activation layer. These up-convolutions are extended to create up-sampling residual blocks by adding a projection connection from the lower resolution feature map to the result. This is called an up-projection block. Chaining up-projection blocks enhance feature map sizes and allow high-level information to pass more efficiently. This is further enhanced by convolving the original feature map with 4 differently composed filters to avoid zero multiplications that arise from the unpooling operations and leading to a decrease in training time. Followed by these up-projection blocks, dropout is applied and succeeded by final convolutional layer yielding the prediction.

The same proposed Loss function is used i.e reverse Huber (berHu) which supposedly performs better than normal L2 loss. It puts more weight towards pixels with high residuals because of L2 and considers the impact of smaller residuals gradients because of L1.

The model architecture [14] is shown in the Fig. 1 below

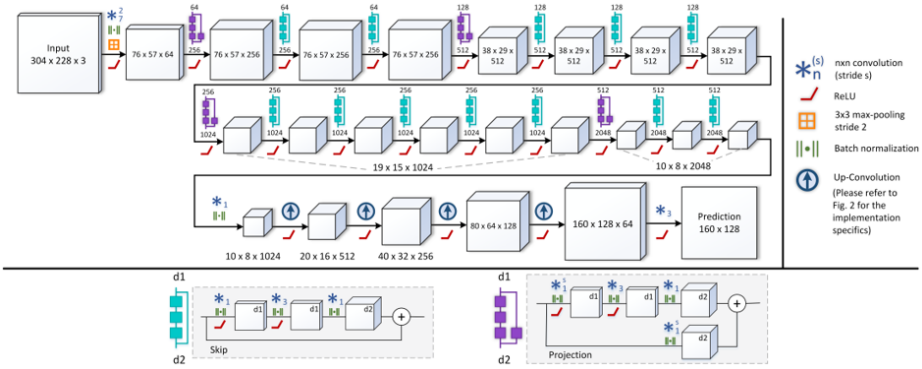

Fig. 1. Fully Convolutional Neural Network Architecture.[14]

3.2 Metric Transformation

In this operation, we precompute pixel-wise coordinates and feed them along with the input features to our CNN network. The primary motivation for this transformation is to allow the network to learn and predict depth patterns that depend on camera calibration.

For this computation, we assumed pinhole camera model which has the following parameters. The intrinsic parameters are focal length f_x and f_y, principal point coordinates c_x and c_y and the image resolution size width w and height h. The metric coordinates X and Y at each pixel x' and y' can be calculated with

the below matrix operation. The python code can be found in the Appendix 6.2.

$$\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{1}$$

Further, we also take multiple random cropping of the image and resize them again to suit accordingly to the architecture input size. The cropping could result in the shift of the principal point and change in the size of the resolution. The resizing of the image could change the focal length of the camera parameters. These operations are performed in an attempt that the network trains to different resolutions and creating an illusion of different cameras which helps in generalizing cameras unseen during training. The metric coordinates are sensitive to these changes and hence they must be computed again with respect to camera parameters.

For example, the below computation is performed to calculate the new principal point and focal length in x.

$$offset\_cx = \frac{orig\_height - crop\_height}{2} \tag{2}$$

$$new\_cx = old\_cx - offset\_cx \tag{3}$$

$$new\_fx = old\_fx * scale \tag{4}$$

where

$offset\_cx$ = offset of height due to cropping
new_cx = modified principal point in x
old_cx = original principal point in x
$orig\_height$ = original height of the RGB image
$crop\_height$ = height of the RGB image after cropping
new_fx = modified focal length in x
old_fx = original focal length in x
$scale$ = the scale by which image is resized

The python code for the computations of modified principal point and focal length can be found in the Appendix 6.1. These computed metric coordinates are concatenated to the three channels (RGB) of the image and then fed to the network.

4 Experiments

We used the NYU Depth Labelled Dataset V2 as collected and analysed by Silberman and Fergus [16] and train on a single NVIDIA GeForce GTX TITAN with 12GB of GPU memory. This dataset has 795 training images and 654 test images.

We have used data augmentation to increase the number of training samples and check its impact on the performance. It showed best results when the input images and corresponding ground truth are transformed using small rotations, the swap of red and green channels, brightness change and horizontal flips with a 0.5 chance, with values inspired from Eigen et al. [13]. Finally, we model small translations by random crops of the augmented images down to the chosen input size of the network.

We have also attempted to introduce this idea of cropping and re-sizing, which is intended to trick the network and make it consider these images are of different cameras. Hence, we perform cropping to three different image resolution i.e. 90%, 80% and 75% of the original resolution and then scale it to target input resolution i.e. 304 x 228. The care was taken to maintain the same aspect ratio while cropping. This process is kept consistent both for training and testing. As mentioned before, we believe this method could help the models to generalize for the unseen cameras.

The original training images of size 640 x 480 pixels are resized to 304 x 228 pixels to suit the input size of the architecture. NYU dataset images has white border which needs to be handled before resizing. The ground truth is checked for invalid depth values and masked out during both training and testing, which also resulted in improved performance. The weight layers of the architecture are initialized by the pre-trained ResNet weights. The fully convolutional network from [14] is trained on RGB images to predict the corresponding depth maps. We train our model with a batch size of 16 for 100 epochs. The learning rate is 0.001 for all layers.

We evaluate on the test subset of 654 images and the output resolution of the network is 160 x 128. The ground truths are downsampled to output size and compared against the predicted depth maps. The evaluation is performed with the same network without finetuning and following the same method of cropping and resizing.

In our experiments, we have also tried out learning rate scheduler. We used the exponential decay method from the TensorFlow with decay rate as a ratio of learning rate and the number of epochs. We observed that it makes a great impact in improving the accuracy of the model, though our main objective is to compare the impact of camera parameters.

In the Table 1, we compare the results obtained by the proposed metric model compared to the baseline model without the pixel coordinates.

Method (on NYU v2)	rel	rmse	$\log_{10}$
Baseline model	0.3728	1.3699	0.1769
Proposed Metric Model	**0.3671**	**1.3523**	**0.1720**

Table 1. Comparison of proposed model against the model without metric coordinates evaluated on NYU v2 dataset

For the quantitative evaluation, the general error metrics which have been used in prior state-of-art works are computed on our experiment results and the lower the better. So we could see from the values in the table that, proposed metric model outperformed baseline model in all metrics indicating an improvement in depth estimation.

In the Fig. 2, we qualitatively compare the accuracy of the predicted depth maps using the metric model with that of baseline model. As you could observe

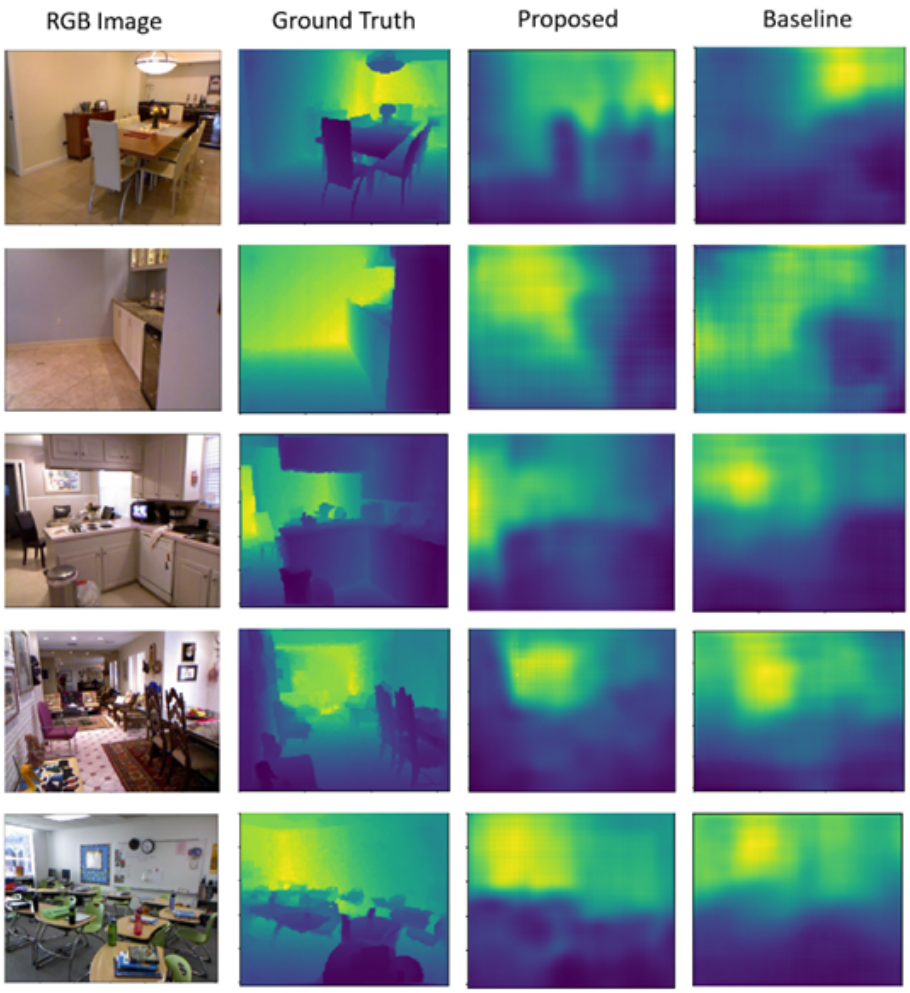

Fig. 2. Qualitative results on NYU Depth images showing predictions using our metric model and the baseline model

from the predicted maps above, our metric model is better at detecting the

overall structure and edges. It is also to be noted that we could achieve these results without even training on the entire raw dataset as in related work. Hence it is evident that calibration information being fed is creating an advantage for prediction performance.

We have also evaluated on another dataset called Sun RGB-D[24]. The dataset contains RGB-D images from NYU depth v2[16], Berkeley B3DO[25], and SUN3D[26]. We picked this dataset because it has images from different cameras and gives an opportunity to test our idea that our metric model generalizes to the datasets of different cameras. Since each dataset has different resolution and camera parameters, the metric coordinates need to be carefully computed separately for each of its datasets. The intrinsic parameters of the datasets are mentioned in Appendix 6.3 for reference.

We evaluated on the test subset of 4992 images and they are cropped accordingly to suit the input and output size of the network as above. The evaluation is performed with the same network following the same method of cropping and resizing. The results obtained are shown in the Table 2

Method (on Sun RGB-D)	rel	rmse	$\log_{10}$
Baseline model	0.5936	4451.19	2.862
Proposed Metric Model	**0.5394**	**4451.12**	**2.856**

Table 2. Comparison of proposed model against the model without metric coordinates evaluated on Sun RGB-D dataset

It can be observed that error measures are slightly higher than usual, but it is because the model is trained on a labelled dataset of NYUv2 of just 654 images. And we evaluated images on dataset different from the training set and also captured with different cameras. We could see that metric model is performing slightly better than baseline model which strengthens our argument that metric coordinates are helpful in networks generalizing to different datasets and cameras unseen during training.

5 Conclusion

Our work introduces a new approach that helps depth estimation networks to be camera independent. It shows that our proposed model generalizes over the camera intrinsics and learn the dependence of the image features from the calibration parameters. It would be interesting to evaluate other state-of-art networks with this approach on more diverse datasets and solidify our idea of camera independence models.

6 Appendix

6.1 Camera Parameters after crop and resize

```
def crop_and_resizeIntrinsic(crop_height, crop_width,
                             orig_height, orig_width,
                             fx_rgb, fy_rgb, cx_rgb, cy_rgb
                             , scale):

    offset_cx = (orig_height - crop_height)/2
    offset_cy = (orig_width - crop_width)/2
    new_cx_rgb = cx_rgb - offset_cx
    new_cy_rgb = cy_rgb - offset_cy
    new_fx, new_fy ,new_cx, new_cy = resizeIntrinsic(
                                 crop_height, crop_width,
                                 fx_rgb, fy_rgb, new_cx_rgb
                                 , new_cy_rgb, scale)
    return new_fx, new_fy ,new_cx, new_cy
```

```
def resizeIntrinsic(height, width, fx, fy, cx_rgb, cy_rgb,
                             scale):

    center_x = float(width-1) / 2
    center_y = float(height-1) / 2
    orig_cx_diff = cx_rgb - center_x
    orig_cy_diff = cy_rgb - center_y
    scaled_height = round(scale * height)
    scaled_width = round(scale * width)
    scaled_center_x = float(scaled_width-1) / 2
    scaled_center_y = float(scaled_height-1) / 2
    new_fx = scale * fx
    new_fy = scale * fy
    new_cx = scaled_center_x + scale * orig_cx_diff
    new_cy = scaled_center_y + scale * orig_cy_diff
    return new_fx, new_fy ,new_cx, new_cy
```

6.2 Calculate Metric Cordinates

```
def findMetricCordinates(fx_rgb, fy_rgb, cx_rgb, cy_rgb,
                             width_px, height_px):

    cam_matrix = np.array([[fx_rgb,0,cx_rgb],[0, fy_rgb,
                                 cy_rgb],[0,0,1]])
    inverse_cam_matrix = np.linalg.inv(cam_matrix)
    nx, ny = (width_px, height_px)
    x = np.linspace(0, nx-1, nx)
    y = np.linspace(0, ny-1, ny)
    xv, yv = np.meshgrid(x, y)
    zv = np.ones((ny,nx))
    pixel_cord = np.stack((xv,yv,zv), axis=2)
    metric_cord = np.empty(shape=(ny,nx,3))
```

```
    for i in range(ny):
        for j in range(nx):
            metric_cord[i][j] = np.dot(inverse_cam_matrix,
                                       pixel_cord[i][j])
    return metric_cord[:,:,:2]
```

6.3 Intrinsic Camera Parameters of the datasets

```
def sun3Dintrinsic(cam):

    if cam == 'b3dodata':
        fx_rgb = 520.532000
        fy_rgb = 520.744400
        cx_rgb = 277.925800
        cy_rgb = 215.115000
        height = 415
        width = 545
        size_afterCrop = [[374,491], [332, 436], [311, 409]]
                                        # 90%, 80%, 75% crop
    elif cam == 'NYUdata':
        fx_rgb = 518.857901
        fy_rgb = 519.469611
        cx_rgb = 284.582449
        cy_rgb = 208.736166
        height = 415
        width = 545
        size_afterCrop = [[374,491], [332, 436], [311, 409]]
    elif cam == 'align_kv2' or cam == 'kinect2data':
        fx_rgb = 529.500000
        fy_rgb = 529.500000
        cx_rgb = 365.000000
        cy_rgb = 265.000000
        height = 518
        width = 714
        size_afterCrop = [[436,643], [414, 571], [388, 535]]
    elif cam == 'lg' or cam == 'sa' or cam == 'shr':
        fx_rgb = 693.744690
        fy_rgb = 693.744690
        cx_rgb = 360.431915
        cy_rgb = 264.750000
        height = 519
        width = 665
        size_afterCrop = [[467,598], [415, 532], [389, 499]]
    elif cam == 'sh':
        fx_rgb = 691.584229
        fy_rgb = 691.584229
        cx_rgb = 362.777557
        cy_rgb = 264.750000
```

```
        height = 519
        width = 665
        size_afterCrop = [[467,598], [415, 532], [389, 499]]
    elif cam == 'sun3ddata':
        fx_rgb = 570.342205
        fy_rgb = 570.342205
        cx_rgb = 310.000000
        cy_rgb = 225.000000
        height = 429
        width = 575
        size_afterCrop = [[386, 517], [343, 460], [322, 431]]
    elif cam == 'xtion_align_data':
        fx_rgb = 570.342224
        fy_rgb = 570.342224
        cx_rgb = 291.000000
        cy_rgb = 231.000000
        height = 429
        width = 575
        size_afterCrop = [[386, 517], [343, 460], [322, 431]]

    crop_cx_rgb = cx_rgb - 8 #white border cropping
    crop_cy_rgb = cy_rgb - 6

    return fx_rgb, fy_rgb ,crop_cx_rgb, crop_cy_rgb, height,
                                    width, size_afterCrop
```

References

1. N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," *Computer Vision ECCV 2012 Lecture Notes in Computer Science*, p. 746760, 2012.
2. X. Ren, L. Bo, and D. Fox, "Rgb-(d) scene labeling: Features and algorithms," *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
3. D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.
4. J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon, "The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation," *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
5. Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving," *International Conference on Learning Representations*, 2020.
6. J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," *Conference on Robot Learning (CoRL)*, 2018.
7. J. Xie, R. Girshick, and A. Farhadi, "Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks," *Computer Vision ECCV 2016 Lecture Notes in Computer Science*, p. 842857, 2016.

8. D. Mwiti, "Research guide for depth estimation with deep learning," Feb. 2020. [Online]. Available: https://heartbeat.fritz.ai/research-guide-for-depth-estimation-with-deep-learning-1a02a439b834
9. R. Szeliski, "Structure from motion," *Texts in Computer Science Computer Vision*, p. 303334, 2010.
10. A. Saxena, M. Sun, and A. Ng, "Make3d: Learning 3d scene structure from a single still image," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, p. 824840, 2009.
11. M. Liu, M. Salzmann, and X. He, "Discrete-continuous depth estimation from a single image," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.
12. X. Guo, K. Nguyen, S. Denman, C. Fookes, and S. Sridharan, "Single image depth prediction using super-column super-pixel features," *2017 IEEE International Conference on Image Processing (ICIP)*, 2017.
13. D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Advances in Neural Information Processing Systems 27*, 2014.
14. I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," *2016 Fourth International Conference on 3D Vision (3DV)*, 2016.
15. F. Liu, C. Shen, and G. Lin, "Deep convolutional neural fields for depth estimation from a single image," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
16. P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," 2012.
17. J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger, "Sparsity invariant cnns," *2017 International Conference on 3D Vision (3DV)*, 2017.
18. T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
19. V. Casser, S. Pirk, R. Mahjourian, and A. Angelova, "Unsupervised monocular depth and ego-motion learning with structure and semantics," 06 2019.
20. W. Chen, Z. Fu, D. Yang, and J. Deng, "Single-image depth perception in the wild." *CoRR*, vol. abs/1604.03901, 2016.
21. Z. Li and N. Snavely, "Megadepth: Learning single-view depth prediction from internet photos," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.
22. J. M. Facil, B. Ummenhofer, H. Zhou, L. Montesano, T. Brox, and J. Civera, "Cam-convs: Camera-aware multi-scale convolutions for single-view depth," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
23. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.
24. S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
25. A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell, "A category-level 3d object dataset: Putting the kinect to work," *Consumer Depth Cameras for Computer Vision*, p. 141165, 2013.

26. J. Xiao, A. Owens, and A. Torralba, "Sun3d: A database of big spaces reconstructed using sfm and object labels," *2013 IEEE International Conference on Computer Vision*, 2013.