# Error Diagnosis of Deep Monocular Depth Estimation Models

Jagpreet Chawla[1], Nikhil Thakurdesai[1], Anuj Godase[1], Md Reza[1], David Crandall[1] and Soon-Heung Jung[2]

***Abstract*— Estimating depth from a monocular image is an ill-posed problem: when the camera projects a 3D scene onto a 2D plane, depth information is inherently and permanently lost. Nevertheless, recent work has shown impressive results in estimating 3D structure from 2D images using deep learning. In this paper, we put on an introspective hat and analyze state-of-the-art monocular depth estimation models in indoor scenes to understand these models' limitations and error patterns. To address errors in depth estimation, we introduce a novel *Depth Error Detection Network (DEDN)* that spatially identifies erroneous depth predictions in the monocular depth estimation models. By experimenting with multiple state-of-the-art monocular indoor depth estimation models on multiple datasets, we show that our proposed depth error detection network can identify a significant number of errors in the predicted depth maps. Our module is flexible and can be readily plugged into any monocular depth prediction network to help diagnose its results. Additionally, we propose a simple yet effective *Depth Error Correction Network (DECN)* that iteratively corrects errors based on our initial error diagnosis.**

## I. INTRODUCTION

Monocular depth estimation is an important problem in robotics and computer vision. Depth maps can be used to understand the 3D structure and relative positions of objects in a scene for applications including autonomous driving [1], visual odometry [2], [3], augmented reality [4], sensor fusion [5], and many others. Estimating depth from a monocular image is an inherently ill-posed problem, since 3D information is irretrievably lost when the camera projects to a 2D image.

Nevertheless, visual cues such as shadows, highlights, defocus, and silhouettes can be exploited to approximately recover the depth map of a scene. Machine learning-based approaches such as Make3D [6], and more recent techniques based on deep learning [7], [8], have shown significant promise. These techniques take a variety of approaches. For example, instead of directly estimating depth, BTS [9] estimates the parameters of local planes at various scales. The model is trained using only ground truth depth, as the local plane parameters are learned implicitly by the network. PlaneRCNN [10], another state-of-the-art technique, estimates planar surfaces in addition to estimating depth for non-planar areas. The final depth map is then produced by combining these two types of outputs.

[1]Luddy School of Informatics, Computing, and Engineering, Indiana University, Bloomington, IN 47408, USA {jchawla, nthakurd, abgodase, mdreza, djcran}@iu.edu

[2]Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea zeroone@etri.re.kr

This work was supported by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government (21ZH1200, The research of the fundamental media·contents technologies for hyper-realistic media space).

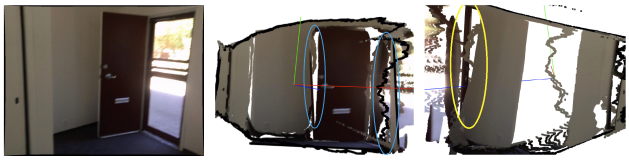

Fig. 1: Sample errors in predicted depth using PlaneRCNN [10]. From left to right: A sample input, and two different views of the generated 3D output. The blue area denotes segmentation issues around the boundaries of the plane, while the yellow area shows planes that are supposed to be adjacent and connected (best viewed in color).

Of course, these papers (and the countless others on monocular depth estimation) present quantitative results that characterize measures of error with respect to ground truth. However, these quantitative error metrics can be surprisingly opaque, making it difficult to choose among algorithms for any given application. For example, multiple algorithms could yield results with exactly the same mean squared depth error, but the error patterns could be completely different: one could have all depths under or over-estimated by some offset, another could have most depth values exactly accurate but with a few extreme outliers, while another could accurately estimate the depth of object surfaces but give inaccurate estimates for object boundaries. Despite having the same quantitative errors, these three algorithms would have very different performance in a real-world application.

In this paper, we propose a technique to analyze methods in monocular depth estimation and to *spatially identify and characterize* likely errors in their output. We evaluate this technique on three diverse approaches to monocular 3D estimation, PlaneRCNN [9], Eigen et al. [7], and BTS [9], and experiment on two different datasets, NYUDv2 [11] and ScanNet [12]. We find that our error diagnostic tools can *identify erroneously predicted depth locations* in monocular depth estimation methods. Additionally, we propose a simple yet effective iterative method to correct the likely errors, showing that it can improve the depth map estimates. Figure 1 demonstrates some of the errors when the same scene is visualized from different viewpoints.

More specifically, we make the following contributions. First, we introduce a Depth Error Detection Network (DEDN) to help diagnose model errors. Our method can locate pixels with likely erroneous depth estimates. Additionally, we propose numerical measures to quantify properties of incorrectly predicted depth locations. Second, we evaluate DEDN on two datasets (NYUDv2 and ScanNet) in single-

view and multi-view settings and using different depth prediction methods. We show it is generic and can be applied to any depth predictor. Third, we introduce a Depth Error Correction Network (DECN) to iteratively correct errors detected by DEDN.

## II. Related work

### *Introspective Capability of Machine Learning Models*

Understanding what a model does not know is a critical part of many applications. Grimmett et al. [13], [14] raised concerns about the limitations of existing metrics (e.g., precision and recall) for evaluating classification. They showed that classifiers like SVMs and LogitBoost are overconfident about their predictions. For high-stakes applications like autonomous driving, the authors emphasized the need for an introspective capability and introduced entropy measurement-based uncertainty estimates during classification. With this novel notion of the introspective capability of a classifier, they demonstrated that the Gaussian process classifier is better suited to some decision-making robotic systems. Berczi et al. [15] reached a similar conclusion with Gaussian processes for a robotic terrain assessment system.

More recently, Gal et al. [16] proposed Monte Carlo dropout for Bayesian approximation to model uncertainties in deep learning. Lakshminarayanan et al. [17] suggested deep ensemble-based uncertainty estimates and demonstrated their value over Bayesian approximation.Kendall et al. [18] identified different types of uncertainties that could arise during decision making, and explicitly encoded them into a Bayesian deep learning model. They demonstrated that this novel model is capable of identifying uncertainties for monocular depth estimation. The first type of uncertainty, *aleatoric uncertainty*, is inherent to the observations, such as depth for a distant object or at occlusion boundaries. The second type, *epistemic uncertainty*, is inherent to the model, e.g., the uncertainty of model parameters, and can be resolved through better models or more training data. More recently, Posetels et al. [19] proposed a sampling-free strategy for estimating the epistemic uncertainty.

### *Error Diagnostic Measures and Metrics*

Error diagnostics have been explored for various image understanding tasks [20], [21] including monocular depth prediction [22]. Boyla et al. [20] introduced a tool called TIDE for identifying different sources of errors in object detection and segmentation. Cadena et al. [22] analyzed the limitations of the existing evaluation metrics — e.g., mean absolute error, root mean square error, etc. — to characterize depth prediction performance. They also proposed a new measure that addresses some deficiencies in earlier metrics. Hekmatian et al. [23] address depth completion from sparse point-clouds of LiDAR sensors. Their error map prediction module, which is explicitly designed to model depth errors, is jointly trained with the depth map predictor. In contrast, we propose two error diagnostic modules to identify and correct errors in sequential stages, decoupled from the existing depth prediction network. Also, while [23] aimed to produce dense depth maps from sparse input point clouds, our method is designed to spatially quantify the errors produced by an existing depth prediction network first, and then to provide a mechanism to improve the predicted depth map. We evaluate on three different depth prediction models and two datasets.

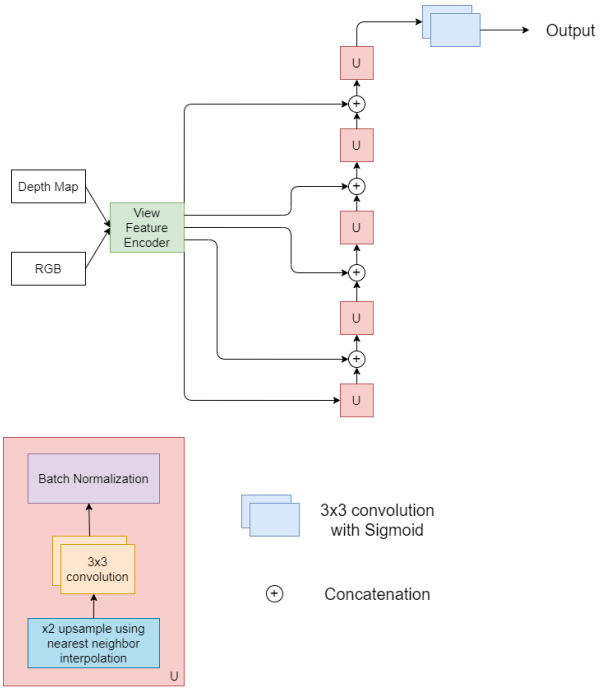


Fig. 2: Single-view Depth Error Detection Network (DEDN), consisting of a View Feature Encoder (Figure 3) that takes the Depth Map and RGB image as input. The U blocks denote Upsampling.

## III. Method

We propose error diagnosis methods for the depth maps produced by deep neural network-based techniques. First, we propose an error diagnostic method – *Depth Error Detection Network (DEDN)* – that can identify locations of erroneous depth predictions in the output of DNN-based depth estimation models. We devised two types of DEDNs to achieve this goal, one for single-view images (Sec III-A.1) and another for multiple views (Sec III-A.2). We also introduce a technique for correcting the detected errors (Sec III-B).

### *A. Depth Error Detection Network (DEDN)*

Our DEDNs receive the output of a depth estimation model and try to predict which pixels have incorrect estimates. More precisely, our first DEDN receives the predicted depth map, along with its corresponding RGB pixel value, from an existing method as input pair, and then quantifies the degree of inconsistencies in that predicted depth map. We refer to this as our *single-view DEDN* (Figure 2) since it focuses solely on the input frame without exploiting error patterns around surrounding frames. Our second error detection network – *multi-view DEDN* (Figure 4) – exploits additional adjacent frames for better error diagnosis.

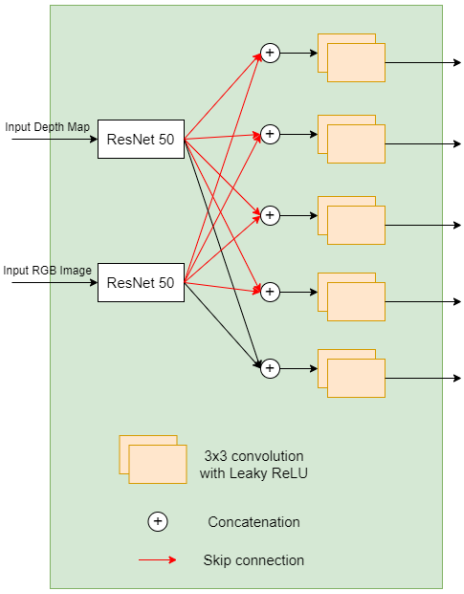


Fig. 3: Our View Feature Encoder consists of two ResNet-50's, for the Input Depth Map and the Input RGB Image. The output feature maps are concatenated together.

*1) Single-view DEDN:* We designed an encoder-decoder network architecture for detecting depth errors. Our encoder can be thought of as a feature extractor acting on an input pair – a predicted depth map (from an existing depth estimation method) and its corresponding RGB image. The decoder then assumes the role of an error predictor that formulates its task as per-pixel error classification. The low-resolution feature maps produced by the encoder need to be upsampled. Inspired by U-Net [24], the decoder module uses skip connections from the encoder at each level to aid with upsampling as shown in Figure 2.

*View Feature Encoder:* As shown in Fig. 3, the View Feature Encoder takes a Predicted Depth Map along with a corresponding RGB image as input. We call this the "view" feature encoder as this same encoder can take multiple views as input. For the single-view case, we just have one view, while for the multi-view case, we can use this same module for mutliple views *without adding* to the complexity of our model (in terms of number of network parameters). We discuss the multi-view case in more detail in Section III-A.2. We use ResNet50v2 [25] pre-trained on Imagenet [26] as an encoder for both types of inputs – predicted depth map and RGB – although other network architectures could be used. Using a pre-trained network has the significant advantage that the encoder already produces some useful features, providing a better starting point. Moreover, using an encoder trained on a different dataset also promotes generalization. from depth input as well.

Since we want to find features from both the RGB image and the predicted depth map and then detect discrepancies to find possible depth errors, ideally we would want a similar set of features. To achieve this goal, we performed an extra pre-training step for our depth encoder leveraging the idea of *cross modal distillation* to transfer knowledge from one image modality to another [27]. Past work [27] showed that transfer of supervision from one modality (e.g., RGB) to another results in significant improvement in downstream tasks that use the second modality (e.g, depth). By leveraging the concept of cross-modal distillation, we take features of a network trained in one modality and train the second network to produce the same features given the corresponding paired image in a different modality. More precisely, to pre-train the depth encoder for extracting features from depth input, we first trained it to produce the same set of features as a pre-trained RGB encoder. The depth encoder and RGB encoder have the same architecture except for a different number of input channels. Using mean squared error as a loss function, we trained the encoder module on ScanNet [12] as it contains a large set of paired RGB and depth images for all of our experiments. Once we trained our depth encoder, we used it as pre-trained encoders to train the complete depth error detection model.

*Decoder:* The decoder consists of several convolutions and upsampling steps, with skip connections from encoders. At each step, the output of two encoders are concatenated, followed by a 2D 3x3 convolution. This is concatenated with the output of the previous decoding step followed by upsampling with nearest neighbor interpolation, 2D 3x3 convolution, and batch normalization. All convolutions use leaky ReLU activation, except for the final output which is a 2D 3x3 convolution with sigmoid activation. See Figure 2.

*Loss function:* We formulated the error prediction as a per-pixel classification task, where the goal is to categorize the predicted depth estimations into correct, over-estimated, or under-estimated. To assign each pixel to a category, we check the absolute difference between the estimated depth $\mathbf{D}$ and the ground truth depth $\mathbf{D}^*$, as follows:

- **Correct** if the estimate is within a threshold $t$ of the ground truth, $|d_{i,j} - d^*_{i,j}| \leq t$.
- **Under-estimate** if the difference is more than $t$ and the predicted depth value is less than the ground truth, $d_{i,j} - d^*_{i,j} < -t$.
- **Over-estimate** if the difference is more than $t$ and the predicted depth value is more than the ground truth, $d_{i,j} - d^*_{i,j} > t$.

Then the cross-entropy loss of our error classification is,

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{3} m_{i,j} \cdot c_j \cdot y_{i,j} \cdot \log(\bar{y}_{i,j})$$

where $N$ is the total number of pixels in the input image, $m_{i,j}$ is the binary mask for each pixel, $c_j$ is the weight assigned to that particular class, $\bar{y}_{i,j}$ is the probability that the estimated depth for pixel $i$ belongs to class $j$ as calculated by the DEDN and $y_{i,j}$ is the ground truth probability for pixel $i$ belonging to class $j$.

The training dataset of our DEDN model is inherently imbalanced and the degree of imbalance depends on the depth prediction – e.g., a very good depth predictor has many more correct pixels than incorrect ones. We handle

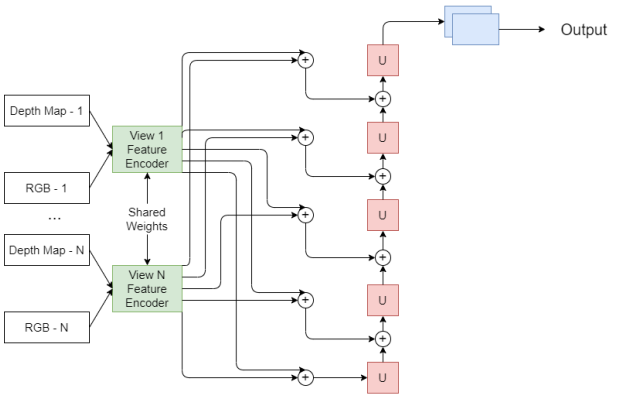


Fig. 4: Multi-view Depth Error Detection Network (DEDN) is based on single-view DEDN but with multiple View Feature Encoders. Moreover, there is an extra concatenation step to gather information from multiple views before doing the upsampling. The "U" blocks denote Upsampling blocks.

the imbalance by using a weighted loss where the weight of each pixel is inversely proportional to the number of samples of that pixel's class in the dataset, so that the class with the greater amount of samples is weighted less and vice versa. Pixels with missing depth are ignored in the loss computation by assigning them a weight of zero to ensure that they do not negatively affect the training of our DEDN.

*2) Multi-view DEDN:* We follow a similar architecture for the multi-view DEDN. Our View Feature Encoder module allows us to take any number of views as input (Figure 4). The weights of the encoder are shared between all the views, so increasing views does not add to the model's complexity. We use two views by taking an adjacent frame in addition the frame for which we are detecting depth errors. Our encoder can be thought of as a Siamese network with two branches, with each frame fed into its own branch. Our intuition for incorporating an additional frame is to provide additional cues which could help identify errors which are inherently challenging to address from a single view, such as occlusion, clutter, or other appearance variations. In the two-view case, given the left RGB image, left depth map, right RGB image, right depth map, the output embeddings from the Siamese network are then concatenated channel-wise and passed through a 3x3 convolution before sending them to the decoder (Figure 4). The same concatenation technique can be used for an arbitrary number of views. Again, we used the same loss function as single-view DEDN to classify errors into categories *under-estimated* and *over-estimated*.

### B. Depth Error Correction Network (DECN)

One application of DEDN is to refine the depth map by trying to correct the errors it has identified. We can do this by incrementally increasing depths of pixels that are predicted to be under-estimated, and decreasing the depths of those predicted to be over-estimated. Once each pixel is classified

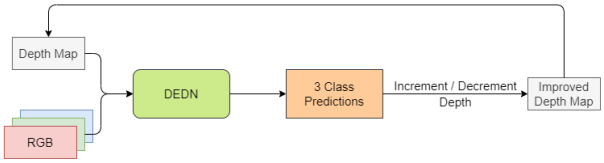


Fig. 5: Depth Error Correction Network (DECN). We can plug in the Single-view or Multi-view version of DEDN.

using the DEDN, we only consider pixels predicted as an error with high confidence (greater than 0.7) and adjust the depth values by incrementing or decrementing them based on the direction, by a fixed small amount (0.01 meters in our experiments). These adjustments yield a new depth map, and the process can be repeated multiple times as shown in Figure 5. We can plug in either version of our DEDN – Single-view or Multi-view.

## IV. Experiments

We experimented with three deep neural network-based monocular depth estimation methods using our error diagnostic modules on two different indoor datasets.

### A. Depth Prediction Models

**Eigen** [7] introduced an early DNN to estimate depth using a stack of two neural networks, one to estimate a coarse depth map using global context, and another to refine this prediction locally. **Plane RCNN** [10] detects planes and their parameters along with a depth map. The final depth map can be produced by combining per pixel depth for non-planar regions and plane parameters for detected planar regions. **BTS** ("From Big to Small") [9] employs a novel local planar guidance layer to produce depth cues at various scales using a local planar assumption. The final depth map is estimated by using these depth cues as input to final convolution layers.

### B. Datasets

**NYUv2** [11] dataset contains over 120,000 RGB-D images gathered from 464 scenes using a Microsoft Kinect. We use the official train-test split with 249 scenes for training and 215 scenes for testing. After aligning and synchronizing the RGB and depth data, we have 24,231 images for training and 654 images for testing. Two common artifacts from Kinect-collected datasets are holes with missing depth information and edge erosion [28]. **ScanNet** [12] is a large RGB-D dataset containing 1,513 indoor scenes and 2.5 million views. It contains 3D camera pose information, surface reconstruction, and semantic segmentation. ScanNet used a Structure sensor [29], which is a portable 3D sensor designed similar to Microsoft Kinect v1. ScanNet is also affected by artifacts such as holes with missing depth information and edge erosion [28]. We conduct evaluations on the respective datasets on which the model was trained. We do not explore cross-dataset evaluation or domain adaptation.

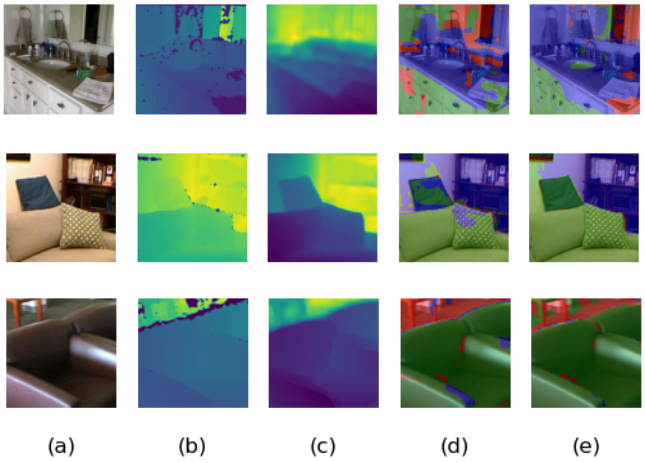


Fig. 6: Sample visualizations of our single-view error detection. Each row denotes error diagnosis on (top to bottom) Eigen [7], BTS [9], and Plane-RCNN [10]. The columns denote: (a) RGB input, (b) ground truth depth, (c) predicted depth from the model, (d) errors in the output as predicted by our model, (e) actual errors in the output compared to ground truth. For (d) and (e), *red* means under-estimated, *green* means correct, and *blue* means over-estimated.

### C. Detecting Errors from the Depth Prediction Models

In order to understand how well our model has learned to detect pixels which have incorrect depth predictions (either under-estimated or over-estimated), we use two metrics – precision and recall – for both the under-estimated and over-estimated classes. We would ideally like a high precision with a reasonable recall (e.g., greater than 50%). This is because we do not want the network to output too many false positives (low precison) as this would hurt the performance of our DECN. At the same time, we do not want many false negatives (low recall) either, as this would mean that we are missing out on correcting those pixels.

*Random Baseline:* To the best of our knowledge, our error diagnosis work (DEDN) is first of its kind. We formulate the problem of error diagnosis of depth prediction into a three-class classification problem: i) under-estimated, ii) correct, and iii) over-estimated. No prior work has tried to classify errors in similar manner, which makes it difficult to compare our results with others. Instead we compare error diagnostic performance of our DEDN against a random baseline.

Each model (Eigen, BTS or Plane-RCNN) has its own distribution for the three classes. For example, BTS has 56% correct, 17% under-estimated, and 27% over-estimated pixels, while Eigen has 35% correct, 37% under-estimated, and 28% over-estimated. To make comparisons fair, we use a random baseline for each model, where pixels are randomly assigned to the classes according to the distribution of model we are using. To calculate the precision and recall for the random baseline, we generate 10 random class predictions each of size 224 x 224. This gives about a half million pixel samples (10 x 224 x 224).

| Model | Under-estimated Precision (↑) | Over-estimated Precision (↑) | Under-estimated Recall (↑) | Over-estimated Recall (↑) |
|---|---|---|---|---|
| Random Baseline | 0.3152 | 0.2775 | 0.3707 | 0.2806 |
| Eigen [7] single-view | 0.4965 | 0.4199 | 0.3585 | 0.4331 |
| Eigen [7] multi-view | **0.8862** | **0.8246** | **0.9292** | **0.7235** |
| Random Baseline | 0.5045 | 0.1427 | 0.5091 | 0.1495 |
| Plane-RCNN [10] single-view | 0.5910 | 0.2290 | 0.5250 | 0.2760 |
| Plane-RCNN [10] multi-view | **0.6824** | **0.5538** | **0.8509** | **0.3794** |
| Random Baseline | 0.0892 | 0.2011 | **0.1730** | 0.2691 |
| BTS [9] single-view | 0.1919 | 0.2431 | 0.0141 | 0.2925 |
| BTS [9] multi-view | **0.5235** | **0.6934** | 0.1484 | **0.7177** |

TABLE I: Error detection results. Higher numbers are better. Under-/over-estimated pixels have estimated depths less/greater than the ground truth, respectively. Numbers in **bold** are best and underline are second best.

*Single-view Results:* The value of $t$ used is 0.2 meters for Eigen and 0.1 meters for BTS and Plane-RCNN. .We achieve precisions of 0.4965 and 0.4199, and recalls of 0.3585 and 0.4331 for the under-estimated and over-estimated classes, respectively, when trained on the Eigen model, as shown in Table I. For Plane-RCNN, we achieve a high precision of 0.5910 and a reasonable recall of 0.5250 for the under-estimated class. We beat the random baseline on all 4 metrics for Plane-RCNN and 3 out of 4 metrics for Eigen and BTS.

Eigen [7] is the oldest of the three models used and hence has more distinct errors than the newer models. Our results reflect this fact, as DEDN is able to classify errors more accurately for Eigen than for the other two models (considering the average precision on both the error classes). BTS is the current state-of-the-art model and hence has the least distinct error patterns, which may explain why our model cannot detect its errors nearly as accurately.

*Multi-view Results:* For multi-view experiments, we used a value of $t = 0.15$ metres for all three models. Table I shows that we achieve a very high precision of 0.8862 and 0.8246 for the under-estimated and over-estimated classes for Eigen. Even for BTS and Plane-RCNN, we achieve high precisions of 0.5235 and 0.6934, and 0.6824 and 0.5538, respectively, for the two error classes. Our results beat the random baseline by very large margins on all of the metrics for Eigen and Plane-RCNN, and on 3 out of 4 metrics for BTS. We note that the multi-view results for each model are better than the corresponding results for single-view, which supports our hypothesis that using multiple views (two in this case) will aid the model to find errors more effectively.

Our architecture is very general and can be easily extended to more views for better accuracy.

### D. Correcting Errors from the Depth Prediction Models

We try to iteratively improve the depth predictions using the predictions from our DEDN as explained in Section III-B. We present results on 4 different widely-accepted error metrics: Accuracy under a threshold ($\delta < thresh$), Absolute Relative Error (AbsRel), Root Mean Square Error (RMSE), and log10; please refer to [9], [22] for definitions.

**Single-view**

| Model | | Metrics | | | |
|---|---|---|---|---|---|
| | | $\delta < 1.25$ (↑) | AbsRel (↓) | RMSE (↓) | log 10 (↓) |
| Eigen [7] | Before | 0.6095 | 4.1154 | 0.8364 | 0.1233 |
| | After | **0.6096** | **3.9943** | **0.8297** | **0.1224** |
| PlaneRCNN [10] | Before | 0.8560 | 0.1260 | 0.2522 | 0.0544 |
| | After | **0.8655** | **0.1233** | **0.2403** | **0.0523** |
| BTS [9] | Before | **0.8958** | **0.1071** | **0.3853** | **0.0454** |
| | After | 0.8916 | 0.1087 | 0.3989 | 0.0472 |

**Multi-view**

| Model | | Metrics | | | |
|---|---|---|---|---|---|
| | | $\delta < 1.25$ (↑) | AbsRel (↓) | RMSE (↓) | log 10 (↓) |
| Eigen [7] | Before | 0.4109 | **0.3518** | 1.0646 | 0.1447 |
| | After | **0.4329** | 0.3706 | **1.0568** | **0.1410** |
| PlaneRCNN [10] | Before | 0.8477 | 0.1208 | 0.3319 | 0.0539 |
| | After | **0.8645** | **0.1191** | **0.3035** | **0.0497** |
| BTS [9] | Before | **0.8882** | **0.1087** | **0.3852** | **0.0462** |
| | After | 0.8750 | 0.1151 | 0.4076 | 0.0513 |

TABLE II: Accuracy of depth maps produced by single-view (top) and multi-view (bottom) DECN.

*Single-view Results:* Our DECN improved the depth map predictions on most of the metrics for Eigen and Plane-RCNN (Table II). For Eigen and Plane-RCNN, DECN improves the performance on all 4 metrics. DECN did not improve the BTS model, presumably due to the low precision and recall for the single-view DEDN. We used 15 iterations for all the three models. We measured improvement in depth prediction in successive iterations but did not notice any improvement after the 15th iteration. Note that the "Before" values are slightly different than the ones reported in the original papers since we have re-run their experiments.

*Multi-view Results:* For the multi-view case, DECN improves on Eigen for 3 out of 4 metrics and on Plane-RCNN for all 4 metrics (Table II). Our model does not improve for BTS, again probably because of the very low under-estimated recall achieved for multi-view DEDN for BTS. We used 20 iterations for Eigen, 10 for BTS, and 15 for Plane-RCNN.

### *E. Model Runtime*

Our model uses a ResNet50 U-Net. The running time of a single-view DEDN is 0.05 seconds (20 *fps*). Single-view DECN requires about 0.76 per frame. For the multi-view case, we pass multiple views through the encoder sequentially, which adds overhead and increases the DEDN time to 0.08 seconds (12.5 *fps*) and DECN to 1.15 seconds. We used an NVIDIA Titan Xp GPU for all experiments; a newer GPU would increase speed significantly.

## V. CONCLUSIONS

We present techniques for error diagnosis to analyze monocular depth estimation models. The *Depth Error Detection Network (DEDN)* locates erroneous depth pixels in single-view and multi-view settings. Our *Depth Error Correction Network (DECN)* improves predicted depth maps, and we show that it improves the results of Eigen and PlaneRCNN. Future work will evaluate using more than two views, and on outdoor scenes.

**Acknowledgements.** This work was supported by the Electronics and Telecommunications Research Institute (ETRI) funded by the Korean government (21ZH1200, The research of the fundamental media contents technologies for hyper-realistic media space).

## REFERENCES

[1] Y. Wang, W. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Weinberger, "Pseudo-lidar from visual depth estimation," in *CVPR*, 2019.
[2] H. Zhan, C. S. Weerasekera, J. W. Bian, and I. Reid, "Visual odometry revisited: What should be learnt?" in *ICRA*, 2020.
[3] N. Yang, R. Wang, J. Stueckler, and D. Cremers, "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry," in *ECCV*, 2018.
[4] W. Lee, N. Park, and W. Woo, "Depth-assisted real-time 3d object detection for augmented reality," in *ICAT*, 2011.
[5] J. M. Fácil, A. Concha, L. Montesano, and J. Civera, "Single-view and multiview depth fusion," *IEEE RAL*, 2017.
[6] A. Saxena, S. H. Chung, and A. Y. Ng, "Learning depth from single monocular images," in *NeurIPS*, 2006.
[7] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *ICCV*, 2015.
[8] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *3DV*, 2016.
[9] J. Lee, M. Han, D. Ko, and I. Suh, "From Big to Small: Multi-scale local planar guidance for monocular depth estimation," in *arXiv*, 2019.
[10] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz, "PlaneRcnn: 3d plane detection and reconstruction from a single image," in *CVPR*, 2019.
[11] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.
[12] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3D reconstructions of indoor scenes," in *CVPR*, 2017.
[13] H. Grimmett, R. Paul, R. Triebel, and I. Posner, "Knowing when we don't know: Introspective classification for mission-critical decision making," in *ICRA*, 2013.
[14] H. Grimmett, R. Triebel, R. Paul, and I. Posner, "Introspective classification for robot perception," *IJRR*, 2016.
[15] L. Berczi, I. Posner, and T. Barfoot, "Learning to assess terrain from human demonstration using an introspective gaussian-process classifier," in *ICRA*, 2015.
[16] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *ICML*, 2016.
[17] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *NeurIPS*, 2017.
[18] A. Kendall and Y. Gall, "What uncertainties do we need in bayesian deep learning for computer vision?" in *NeurIPS*, 2017.
[19] J. Postels, F. Ferroni, H. Coskun, N. Navab, and F. Tombari, "Sampling-free epistemic uncertainty estimation using approximated variance propagation," in *ICCV*, 2019.
[20] D. Bolya, S. Foley, J. Hays, and J. Hoffman, "TIDE: a general toolbox for identifying object detection errors," in *ECCV*, 2020.
[21] D. Hoiem, Y. Chodpathumwan, and Q. Dai, "Diagnosing error in object detectors," in *ECCV*, 2012.
[22] C. Cadena, Y. Latif, and I. Reid, "Measuring the performance of single image depth estimation methods," in *IROS*, October 2016.
[23] H. Hekmatian, J. Jin, and S. Al-Stouhi, "Conf-net: Toward high-confidence dense 3d point-cloud with error-map prediction," in *arXiv*, 2019.
[24] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *MICCAI*, 2015.
[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CVPR*, 2016.
[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
[27] S. Gupta, J. Hoffman, and J. Malik, "Cross modal distillation for supervision transfer," in *CVPR*, 2016.
[28] A. Kadambi, A. Bhandari, and R. Raskar, "3D depth cameras in vision," in *Computer Vision and Machine Learning with RGB-D Sensors*, 2014.
[29] "Structure sensor." [Online]. Available: https://structure.io