

(CV PA3) SDR: Supervised Depth Refinement

20225092 Huigyoeng Son

Abstract

본 프로젝트에서는 주어진 (Sparse Depth, RGB Image, Surface Normal)으로부터 정확한 전체 Depth를 예측하는 **SDR(Supervised Depth Refinement)**을 구현하였다. SDR model은 Sparse와 RGB를 입력으로 받아 Depth와 Normal을 출력하며, Depth Refinement를 수행하기 위해 Sparse Ground Truth와 Normal Ground Truth의 Supervision하에 가중치를 학습한다. Baseline model은 HoleFiller, UNet, Depth2Normal 모듈로 구성되어있으며, 이 중 학습가능한 모듈은 UNet으로 Sparse depth loss와 Normal loss를 줄이는 방향으로 학습된다. Baseline model에 대해 성능을 개선하기 위해 우리는 **ArchBoost(Architecture Boost)**와 **DataBoost(Data-driven Boost)**를 추가적으로 고안하여 사용하였다. ArchBoost는 (i) Smooth Hole-filling (ii) Learnable Depth2Normal (iii) Auxiliary Depth Loss의 구조적인 개선을 통해 성능을 향상시키고, DataBoost는 (i) Robustness를 위한 Transfer Learning과 (ii) Sample Data Augmentation의 Data-driven 접근을 통해 성능을 향상시킨다. 해당 프로젝트에 사용한 코드는 다음의 github에서 확인할 수 있다: <https://github.com/gyoenge/cvpa3-depth-refinement>.

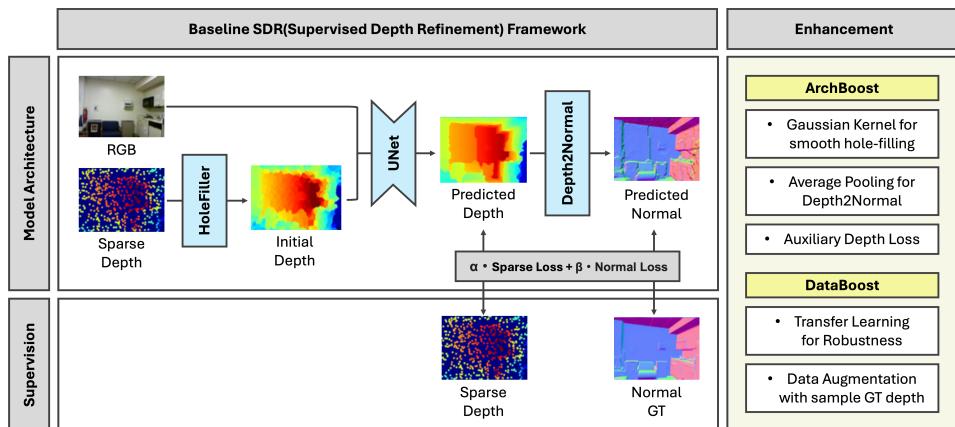


Figure 1: Overview framework of Supervised Depth Refinement

1 Introduction

1.1 SDR(Supervised Depth Refinement): Our Task

Depth Completion은 불완전한 깊이 정보(예: Sparse Depth Map)를 입력으로 받아, 해당 장면의 Dense Depth Map을 복원하는 Computer Vision Task이다. 이때 입력으로 RGB 이미지를 함께 사용하는 경우가 많으며, 전통적인 접근 방법으로는 Guided image filtering, Joint bilateral filtering, Optimization-based refinement 등이 있다. 우리가 풀고자하는 SDR(Supervised Depth Refinement)은 Depth Completion Task의 일종으로, RGB-D(RGB image + Sparse Depth)으로부터 Sparse Depth와 Normal Map GT(Ground Truth)의 Supervision하에서 Dense Depth Map을 복원하는 Task이다. SDR은 특히 Normal Map을 Supervision 신호로 활용하여 구조적으로 더 정밀한 깊이 복원이 가능하다는 점에서 기존 Depth Completion보다 확장된 형태라 할 수 있다. 우리는 Optimization 기반의 접근 방식을 사용한다.

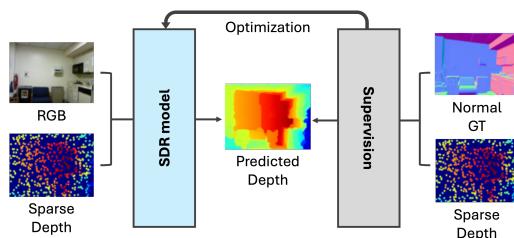


Figure 2: SDR Task Definition

2 Method

2.1 Baseline

Hole Filling

Hole Filling은 입력으로 주어진 Sparse Depth의 빈 공간을 채워 Initial Depth로 변환하는 과정이다. Baseline model은 Mean Convolution 연산을 반복하여 수행하는 방식으로 Hole Filling을 수행한다. 여기서 Mean Convolution 연산이란, 모든 값이 1인 Convolution Filter를 적용한 후 Filter 내 Valid Pixel 수만큼 나누는 연산을 말한다. 해당 연산을 반복 수행하면 Sparse Depth의 초기점 값들이 이웃으로 점차 전파되어 자연스러운 Initial Depth를 구할 수 있게 된다.

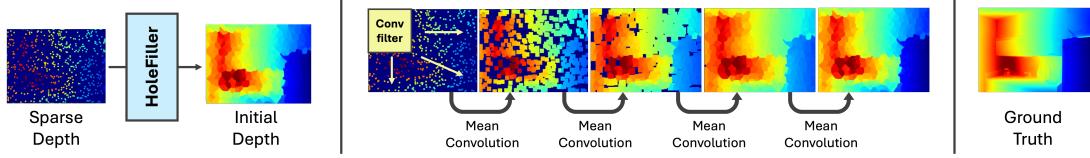


Figure 3: Hole Filling Process

위의 Fig 3은 이러한 Hole Filling 과정을 나타낸다. 여기서 우리는 Mean Convolution 수행에 따른 Depth 이미지 크기의 변화(감소)를 방지하기 위해, 매 수행마다 적절한 padding을 주어 이미지 크기가 일정하게 유지되도록 구현하였다. Convolution Kernel(Filter) Size와 Mean Convolution Iteration은 실험을 통해 결정할 수 있었다.

UNet: Depth Refining

Initial Depth는 UNet을 통해 Final(Predicted) Depth로 Refinement된다. UNet[1]은 본래 Image Segmentation Task에 사용되는 Convolutional Network으로, 대칭적인 U자형 Encoder-Decoder 형태의 구조를 가지며, high-resolution feature와 low-level context를 효과적으로 결합해 정밀한 pixel-wise 예측이 가능한 네트워크이다. Fig 4는 우리가 사용한 UNet Architecture를 나타낸 것이다. Network는 Encoder에 대응되는 Contracting path와 Decoder에 대응되는 Expansive path로 구성된다. Contracting path는 전형적인 Convolutional network의 구조를 따르는데, 2번의 3x3 Unpadded Convolution + ReLU 연산과 2x2 Stride2 Maxpooling 연산을 반복적으로 수행하도록 구성되어 있다. Expansive path는 2x2 Up-Convolution(feature channel을 절반으로 출력) 연산 결과와 Contracting path에서의 대응되는 Feature map을 Concatenation한 후, 2번의 3x3 Convolution + ReLU 연산을 반복적으로 수행하도록 구성되어 있다. 기본적인 Step 수와 Internal Feature Channel 크기는 원본 논문과 같으나, 다른 점은 Double Conv (파란 화살표) 과정에서 이미지 크기가 변하지 않도록 하는 것과, Concatenation을 수행할 때 Crop으로 크기를 맞추는 대신 Padding으로 크기를 맞추는 방식을 사용한다는 점이다. 또한, Input Feature Channel은 4로 RGBD를 포함하도록 하였고, Output Feature Channel은 1로 Depth Map을 표현하도록 구성하였다. 특히 우리는 Pytorch UNet 구현[2]을 따랐고, 아래의 Fig 4는 특히 우리가 사용한 Sample의 이미지 크기에 맞춰 구조와 inference 과정을 표현한 것이다.

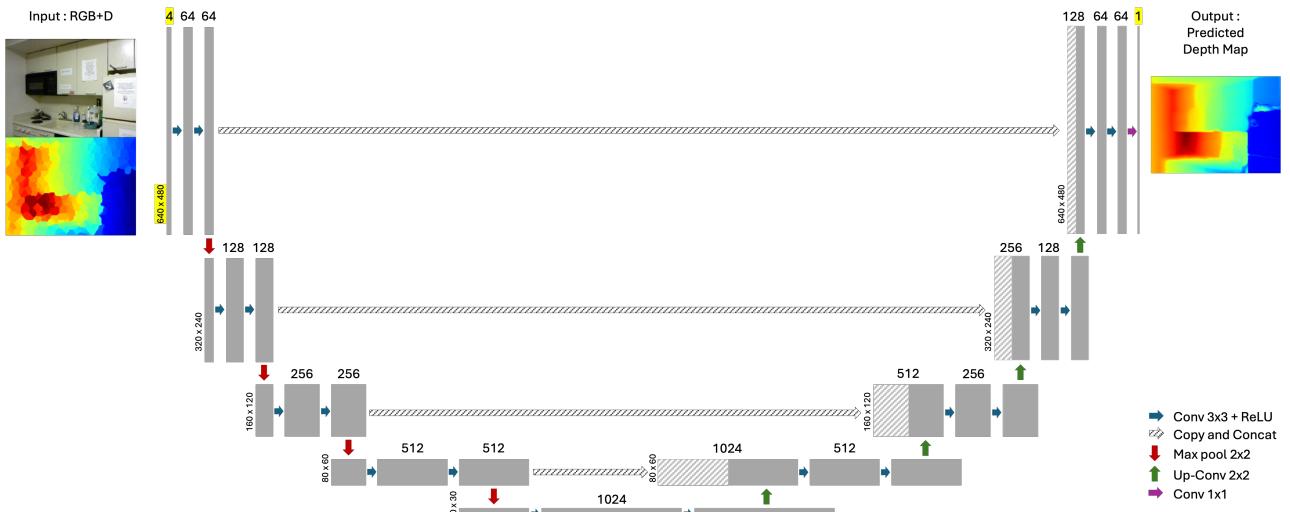


Figure 4: UNet Architecture

UNet은 원래 Segmentation Task을 수행하기 위해 고안된 모델이지만, Out Class의 수를 1로 두면 Depth Map Regression 문제로 대응할 수 있다. 따라서 SDR(Supervised Depth Refinement) Task의 Depth 추정에도 사용할 수 있다. 해당 UNet 모델은 학습이 가능하며 Normal Map GT와 Sparse Depth의 Supervision을 받아 학습되는 핵심 부분이다.

Depth to Normal

Depth to Normal 과정은 Predicted Dense Depth를 Normal Map으로의 계산을 포함한다. Baseline에서 Depth to Normal은 학습가능한 과정이 아니며, 오로지 수치 계산에 의존하고 있다. 먼저 Depth Map에 Camera Intrinsic Matrix의 역행렬을 곱하면 Point Map (Camera Coordinate 상의 3차원 좌표)를 계산할 수 있는데, Point Map에서 모든 점들에 대해 각각 Vertical vector (V)와 Horizontal vector (H)의 Cross product를 수행한 Vector를 최종적으로 정규화함으로써 Surface Normal Map을 구할 수 있다. 이러한 과정은 아래 Fig 5에 나타나 있으며, 추후 Optimization에서 Backpropagation 수행을 위해 해당 계산 모듈은 Pytorch nn.Module로 구현되었다.

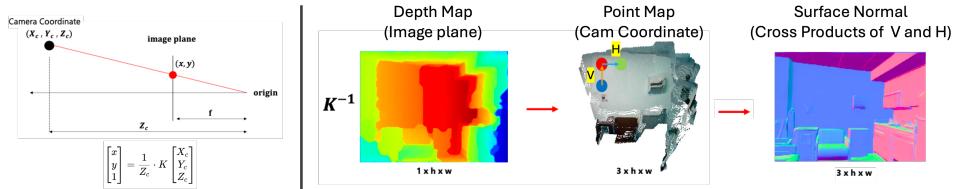


Figure 5: Depth to Normal

Supervised Optimization

HoleFiller, UNet, Depth2Normal의 세 모듈을 종합하여 최종적으로 SDR Baseline model을 구현할 수 있었다. UNet이 Dense Depth를 적절히 예측하도록 학습시키기 위해, 결과 값인 Dense Depth와 Surface Normal에 대해 Ground Truth와 비교하여 차이를 감소시키는 방향으로 Supervised Learning을 수행하였다. 따라서 이때 전체 Loss function은 $\alpha \cdot \text{Sparse Loss} + \beta \cdot \text{Normal Loss}$ 으로 구현된다. 우리는 Sparse Loss로 MSE Loss Function을 사용하였고, Normal Loss로 F1 Loss Function을 사용하였다. MSE Loss는 depth의 정확한 수치 회귀에, L1 Loss는 normal의 벡터 방향 차이를 직접 줄이는 데 적합하며, 두 supervision은 각각 거리 정확도와 기하 구조 보존을 동시에 강화한다.

2.2 ArchBoost: Architecture Boost

Smooth Hole Filling

Baseline의 Hole Filling은 단순한 Average Convolution Filter를 사용하기 때문에, 결과적으로 Initial Depth 내부에 영역간 경계가 블록처럼 생기는 현상이 나타난다. 그러나 이러한 경계는 Ground Truth의 경계와는 차이가 존재하며, 오히려 학습을 불안정하게 만들 수 있는 요소라고 판단되었다. 따라서 조금 더 부드럽게 빈 공간을 채우는 방법을 사용하기로 했다. 우리는 2차원 정규분포를 따르는 Gaussian Filter를 사용하여 Smooth Hole Filling을 구현하였다.

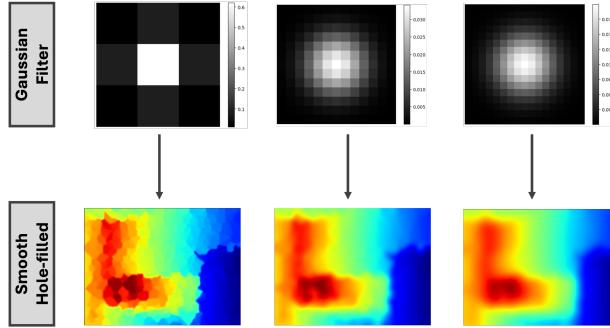


Figure 6: Smooth Hole Filling

위 Fig 6은 Gaussian Filter 크기에 따른 Smooth Hole Filling 예시 결과이다. 작은 Gaussian Filter를 사용할 때는 Baseline의 Hole Filling과 유사한 결과가 나타난다. 더 부드러운 Gaussian Filter를 적용함에 따라 Initial Depth는 더 부드럽게 나타나며, 기존의 블록처럼 경계가 생기는 현상이 완화된다. 자세한 결과는 Section 3의 Result에서 다룬다.

Average Pooling Depth2Normal

Baseline의 Depth2Normal은 기본적으로 각 점에서 한 쌍의 Vertical, Horizontal 벡터들로부터 하나의 normal을 추정하는 방식이다. 한 쌍에 의존하기 때문에 작은 오차에 의해서도 큰 변화가 생길 수 있으며, 따라서 학습이 불안정해지는 요소가 될 수 있다고 생각했다. 이러한 불안정성은 특히 Normal Map에서 값들이 급격히 변하는 Edge들이 많을 때 더 심하게 나타난다. 따라서 우리는 일차적으로 구한 Normal Map에 대해, Average Pooling을 적용하여 픽셀마다 커널 내의 이웃 Normal들의 값을 참조하여 최종 Normal Vector를 갖도록 하였다.

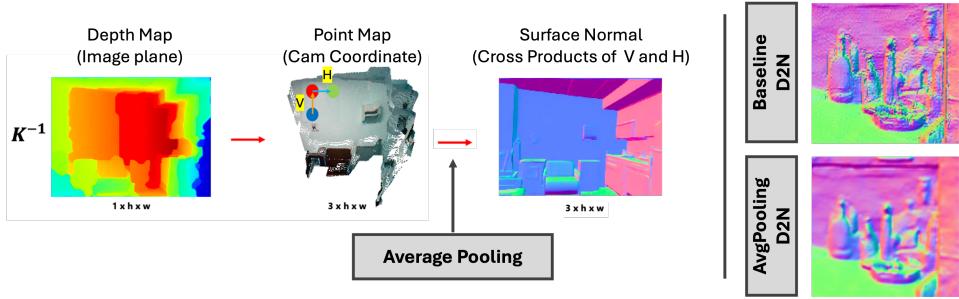


Figure 7: Average Pooling Depth to Normal

결과적으로 위 Fig 7과 같이 Average Pooling을 적용하였을 때, Edge 오차에 의해 깨지는 현상이 완화되는 것을 확인할 수 있다. 뿐만 아니라, 이러한 Normal 추정으로 학습되는 Depth 또한 부드럽게 형성되는 효과를 얻을 수 있다. 자세한 결과는 Section 3의 Result에서 다룬다.

Auxiliary Depth Loss

Baseline은 Sparse MSE Loss와 Normal L1 Loss를 결합하여 손실함수를 정의한다. 우리는 여기에 추가적으로 Auxiliary Depth Loss를 결합함으로써 더 부드럽고 정확한 Depth 추정과 안정적인 학습이 가능하도록 하였다. Auxiliary Depth Loss는 (i) Depth가 항상 양수가 되도록하는 Minus Penalty Loss와 (ii) Edge가 아닌 곳에서 Depth의 변화가 부드럽게 일어나도록 하는 Edge-aware Smoothness Loss를 결합하여 사용하였다.

$$\begin{aligned} \text{ArchBoost Total Loss} &= \alpha \text{ Sparse MSE Loss} + \beta \text{ Normal F1 Loss} + \lambda \text{ Auxiliary Depth Loss} \\ \text{Auxiliary Depth Loss} &= \text{Minus Penalty Loss} + \text{Edge-aware Smoothness Loss} \end{aligned}$$

Figure 8: Auxiliary Depth Loss

Minus Penalty Loss는 Depth가 음수를 갖게 되면 그 절댓값에 비례하게 Penalty 값을 주도록 설계되었다. Edge-aware Smoothness Loss는 우선 Ground Truth Normal으로부터 Edge를 추출한 후, 해당 Edge가 아닌 영역에 대해 x방향의 depth 변화와 y방향의 depth 변화가 각각 부드럽게 형성될 수 있도록 Loss를 주어 설계되었다. 자세한 구현 방법은 코드에서, 자세한 결과는 Section 3의 Result에서 다룬다.

2.3 DataBoost: Data-driven Boost

Transfer Learning

기본적으로 Baseline은 주어진 Example(Train) Sample 한 쌍에 대해서만 학습이 이루어진다. 그렇기 때문에 같은 환경(같은 Camera Intrinsic 및 비슷한 장면)이더라도, 다른 장면일 때에 예측에 오차가 생길 확률이 높다. 따라서 Robust 한 모델을 만들기 위해, 우리는 Sample 쌍들에 대해 학습을 확장시킬 필요가 있다. 외부 데이터셋으로 NYUv2 Depth Dataset을 활용하는 방안을 고려하였지만, 우리는 과제의 취지에 맞게 주어진 하나의 Sample로부터 Data Augmentation 을 수행하여 학습을 확장하는 방법을 사용한다. 자세한 내용은 바로 다음 Subsection에서 다룬다.

Data Augmentation from single sample

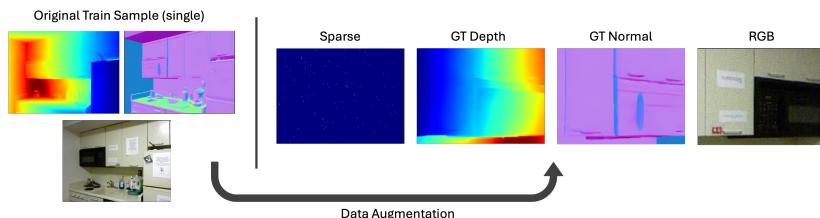


Figure 9: Data Augmentation

Fig 9와 같이 우리는 GT Depth가 주어진 하나의 Example(Train) Sample에 대해서 Crop과 Random Sparse Sampling 을 수행함으로써 Data Augmentation을 수행하였다. 이때 Sparse Sampling은 원본 Sparse 비율과 비슷한 0.002값으로

진행하였으며, Cropping은 원본 이미지의 0.4배만큼의 크기로 진행하였다. 결과적으로 총 1000쌍의 Data Sample을 생성하여 사용할 수 있었다.

3 Result and Discussion

3.1 Baseline Analysis

Hole Filling

아래 Fig 10은 Baseline 모델에 대해 Hole Filling의 Hyperparameter들을 조정하면서 얻은 결과이다. 가로축은 Hole Filling Kernel Size, 세로축은 Hole Filling Iteration의 변경을 나타낸다.

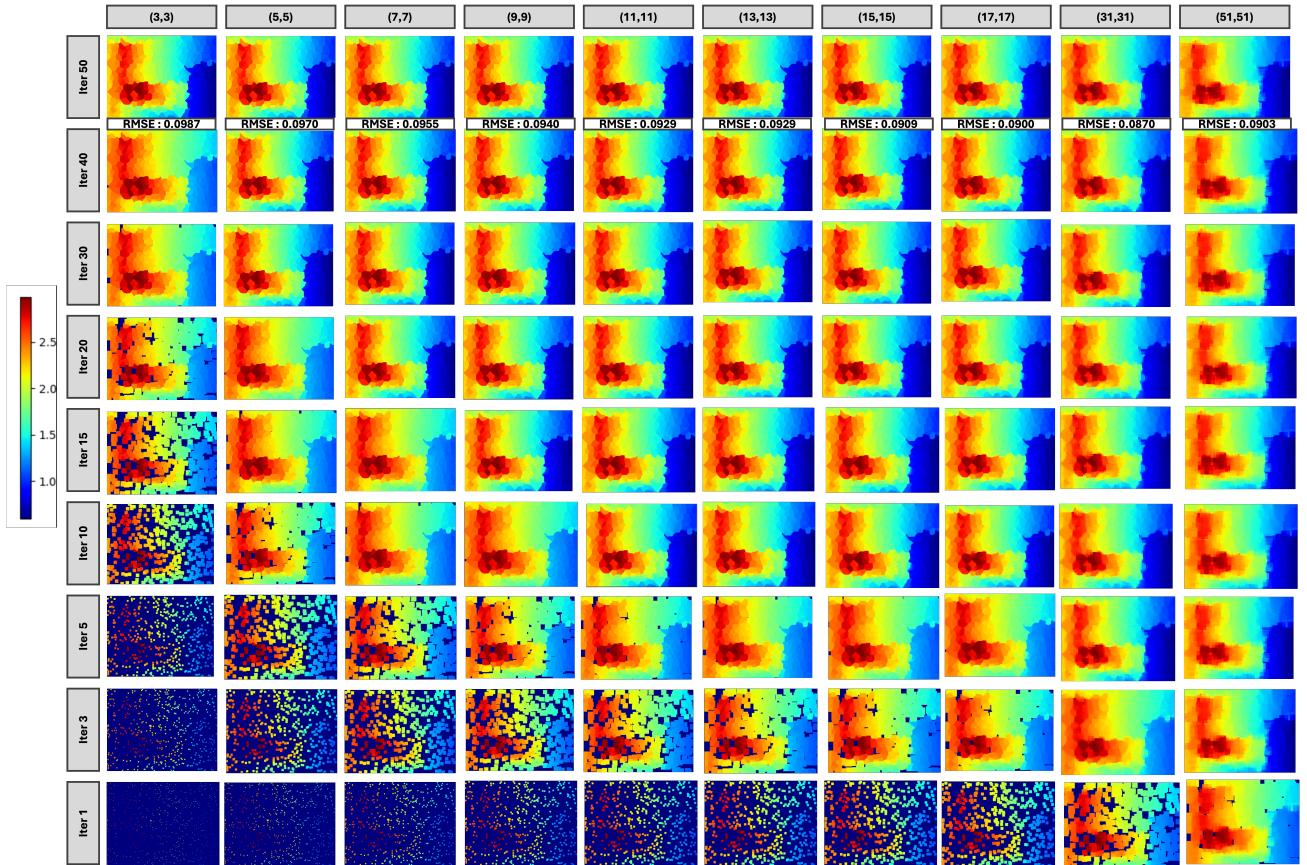


Figure 10: Baseline Analysis: Hole Filling

대개 Iteration 30이상을 되었을 때 전체 영역이 채워지는 것을 확인할 수 있었고, GT Depth와의 RMSE를 Iteration 50의 결과에 대해 평가하였다. Kernel Size가 증가할수록 점진적으로 RMSE가 줄어드는 것을 확인할 수 있었다. (31,31) Kernel에서 최소 RMSE를 얻을 수 있었고, 그 이후로는 다시 RMSE가 증가하였다. 정성적으로는 Kernel이 커질수록 모자이크 모양으로 블러링되듯이 영역이 채워지는 것을 확인할 수 있다. 그러나 Kernel Size에 따른 RMSE 증가가 유의미한 정도로 크게 나타나지 않았기에, 적당한 Kernel Size를 선택하여 사용하기로 결정하였다. 결과적으로 우리는 Kernel Size는 (7,7), Iteration은 30을 사용하였다. 따라서 Baseline의 Initial RMSE는 0.0955이다.

Train Hyperparameter

다음으로 우리는 Training에 사용하는 Hyperparameter를 조절해가며 결과를 확인하였다. 먼저 Learning Rate와 Epoch을 동시에 변경해가며 결과를 비교하였고, Fig 11과 같이 결과가 나타났다. Learning Rate는 1e-3에서 1e-4 사이의 값이 적절한 것으로 나타났고, Epoch은 1000정도가 결과가 가장 좋았다. 결과적으로 Learning Rate는 5e-3을 사용하였으며, 이때 가장 결과가 좋었던 Epoch 10000을 함께 사용하였다.

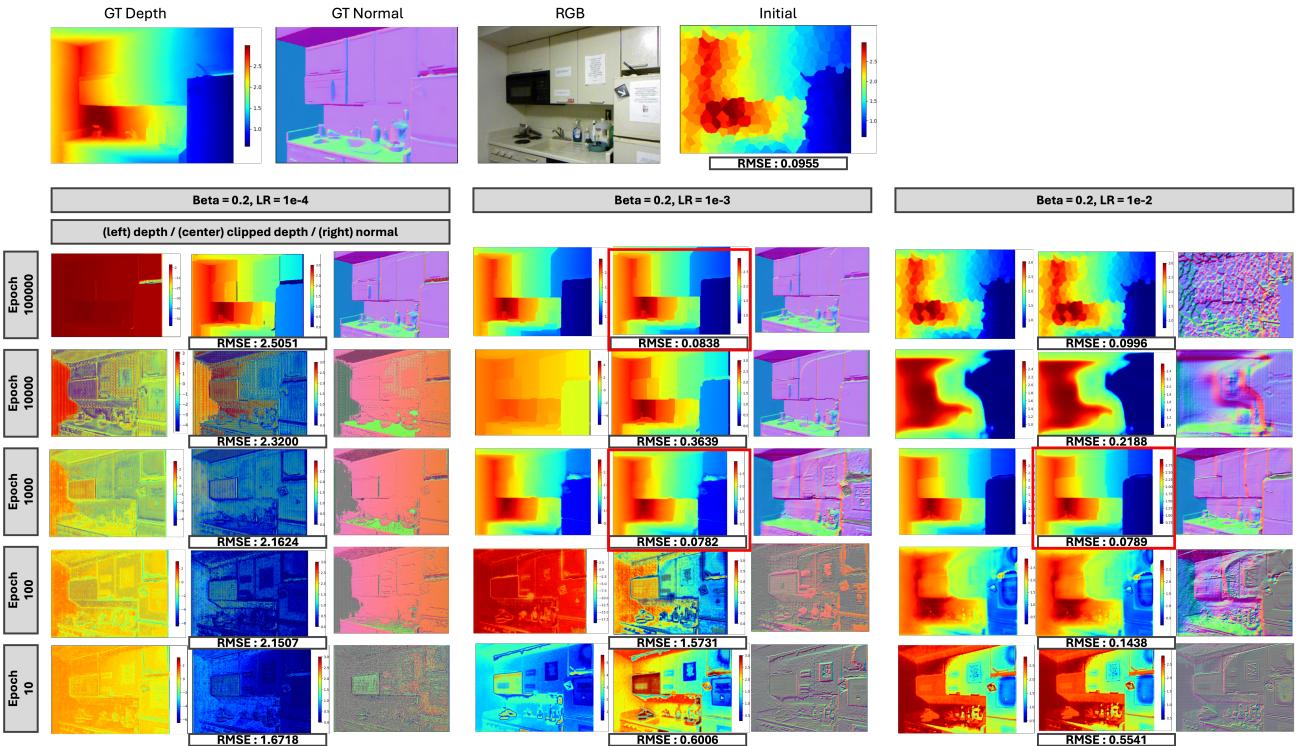


Figure 11: Baseline Analysis: Learning Rate and Epoch

여기서 주목할 점은 Baseline 학습에 있어서의 한계점이다. Fig 12는 Predicted Depth의 Outlier를 시각화 한 것이다. 특히 음수의 값을 가지도록 Depth가 예측되는 경우가 많았고, 이것이 학습을 불안정하게 만든다고 추정할 수 있다. 이것은 Auxiliary Depth Loss의 필요성을 보여준다. 뿐만 아니라, Fig 11의 Learning Rate가 1e-2일 때 Overfitting되면서 Depth의 블록 경계에 영향을 많이 받게 된다는 것을 알 수 있다. 따라서 이 또한 학습을 불안전하게 만드는 요소라고 추정할 수 있으며, Smooth Hole Filling의 필요성을 보여준다.

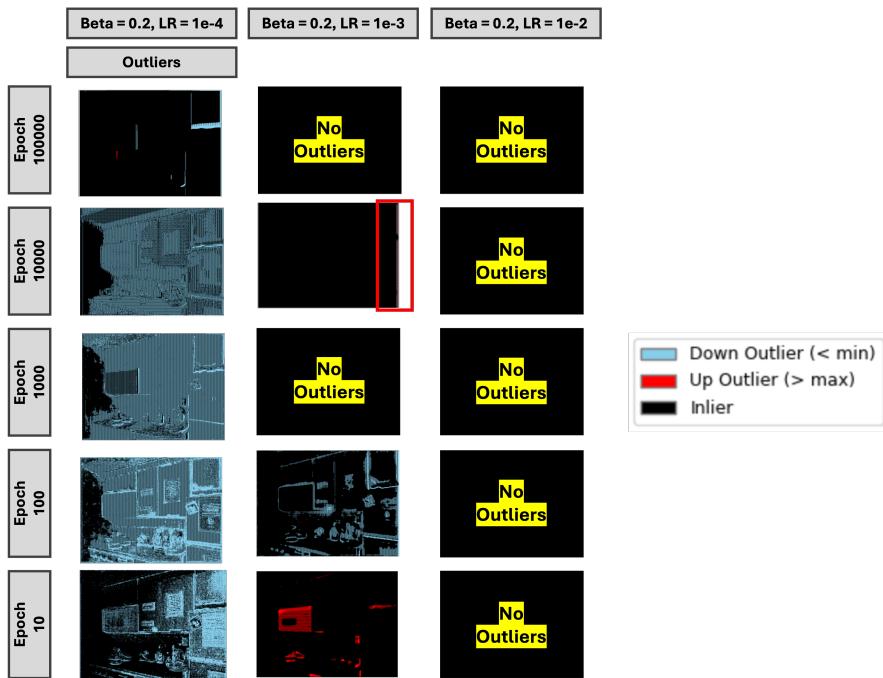


Figure 12: Baseline Analysis: Learning Rate and Epoch, Visualize Depth Outliers

다음으로 Loss에 대한 Hyperparameter인 Beta값을 조정해가며 학습시킨 결과가 Fig 13에 나와있다. 이때 주의할 점은 우리는 Alpha는 1.0으로 고정한 채 Beta만을 조정해가며 비교하였다.

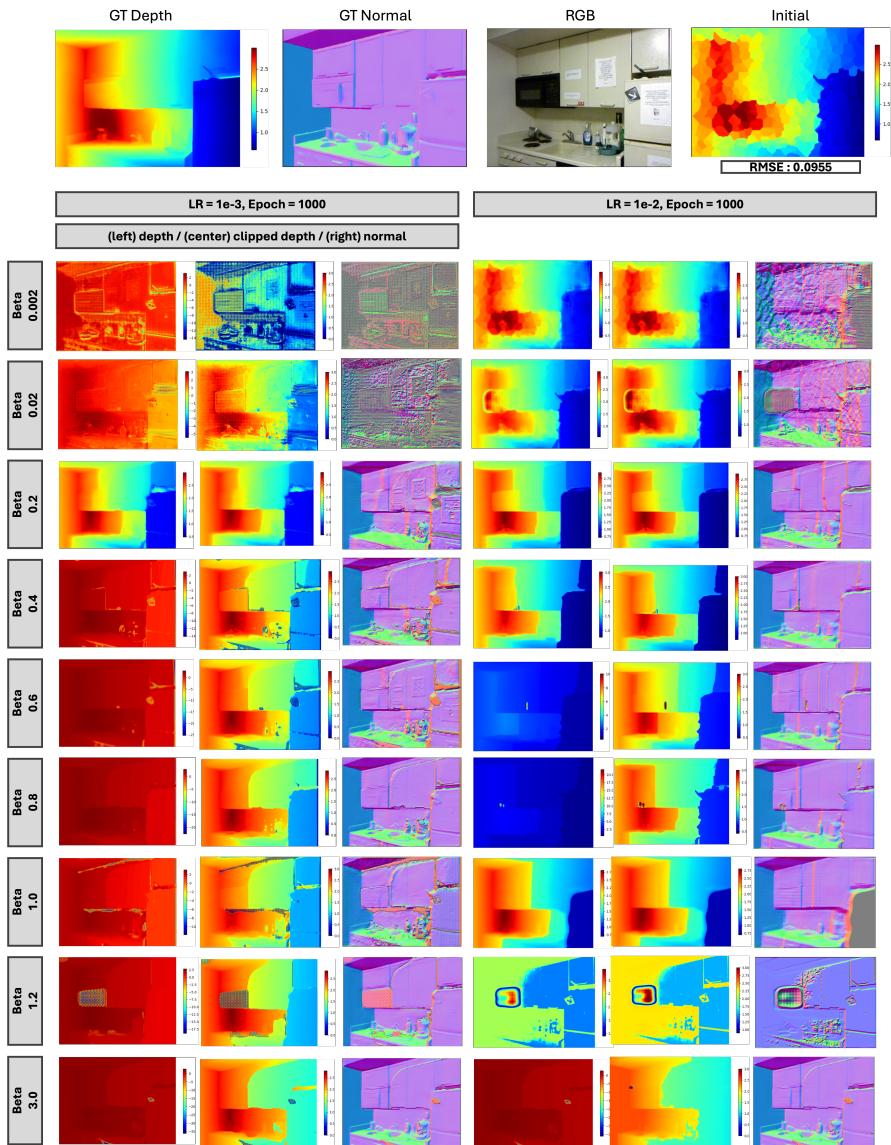


Figure 13: Baseline Analysis: Beta

Beta 값에 따라 학습 결과가 극단적으로 달라지는 것을 확인할 수 있었다. 결과적으로 우리는 1e-3에서 1e-2사이의 Learning Rate에 대해 안정적으로 학습이되는 Beta=0.2를 사용하였다.

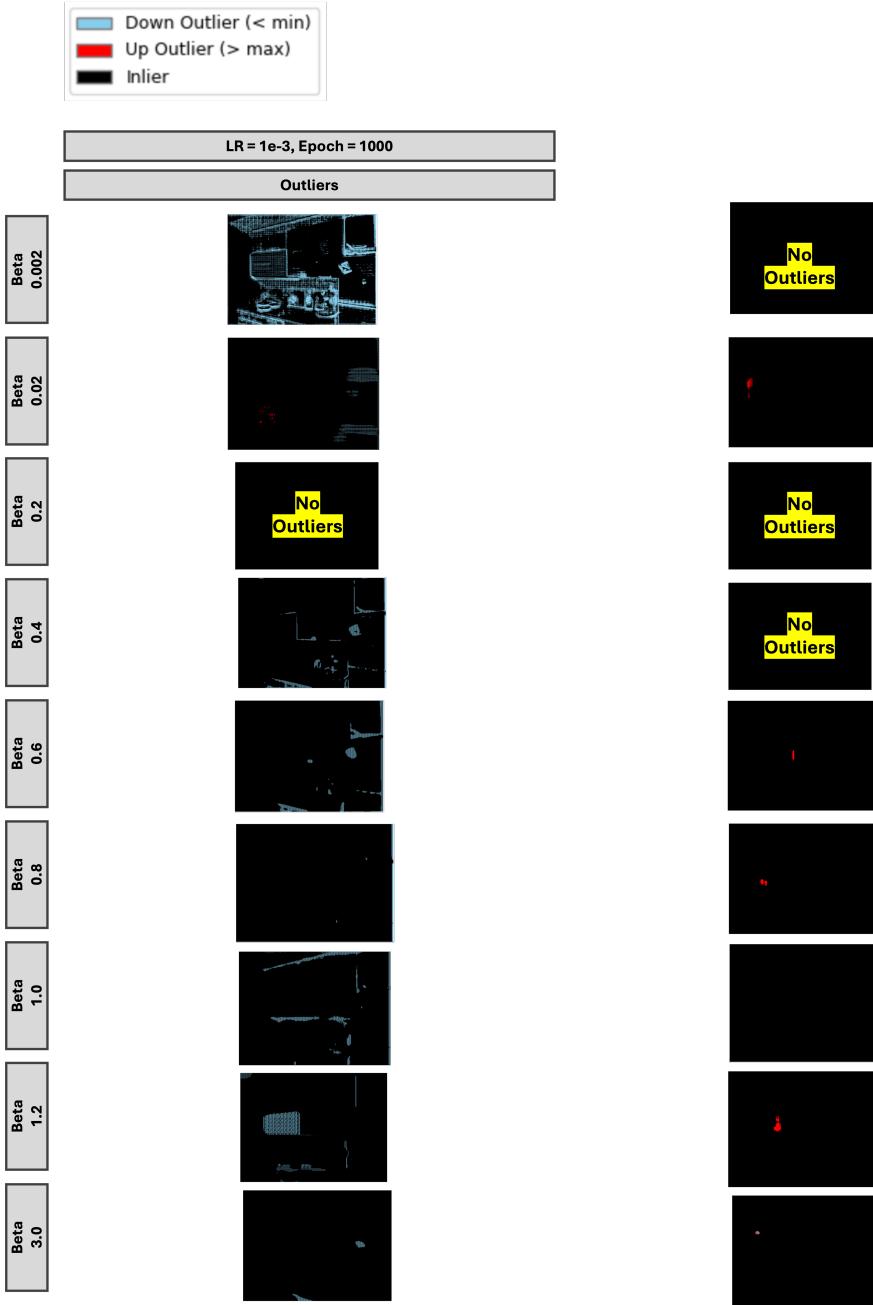


Figure 14: Baseline Analysis: Beta, Visualize Depth Outliers

최종적으로 Baseline에 대해 RMSE 0.0808을 얻을 수 있었다.

3.2 ArchBoost Analysis

Smooth Hole Filling

ArchBoost의 Smooth Hole Filling의 Hyperparameter를 변경해가며 확인한 결과는 Fig 15와 같다. Gaussian Kernel에 따른 Smoothing 효과를 정량적, 정성적으로 확인할 수 있었다. Baseline Hole Filling보다 더 낮은 RMSE를 얻을 수 있었고, 정성적으로도 부드러운 결과를 얻는 것을 확인하였다. 특히 (13,13) size와 30 iter에 대해 안정적인 결과를 보여 해당 파라미터를 사용하였다.

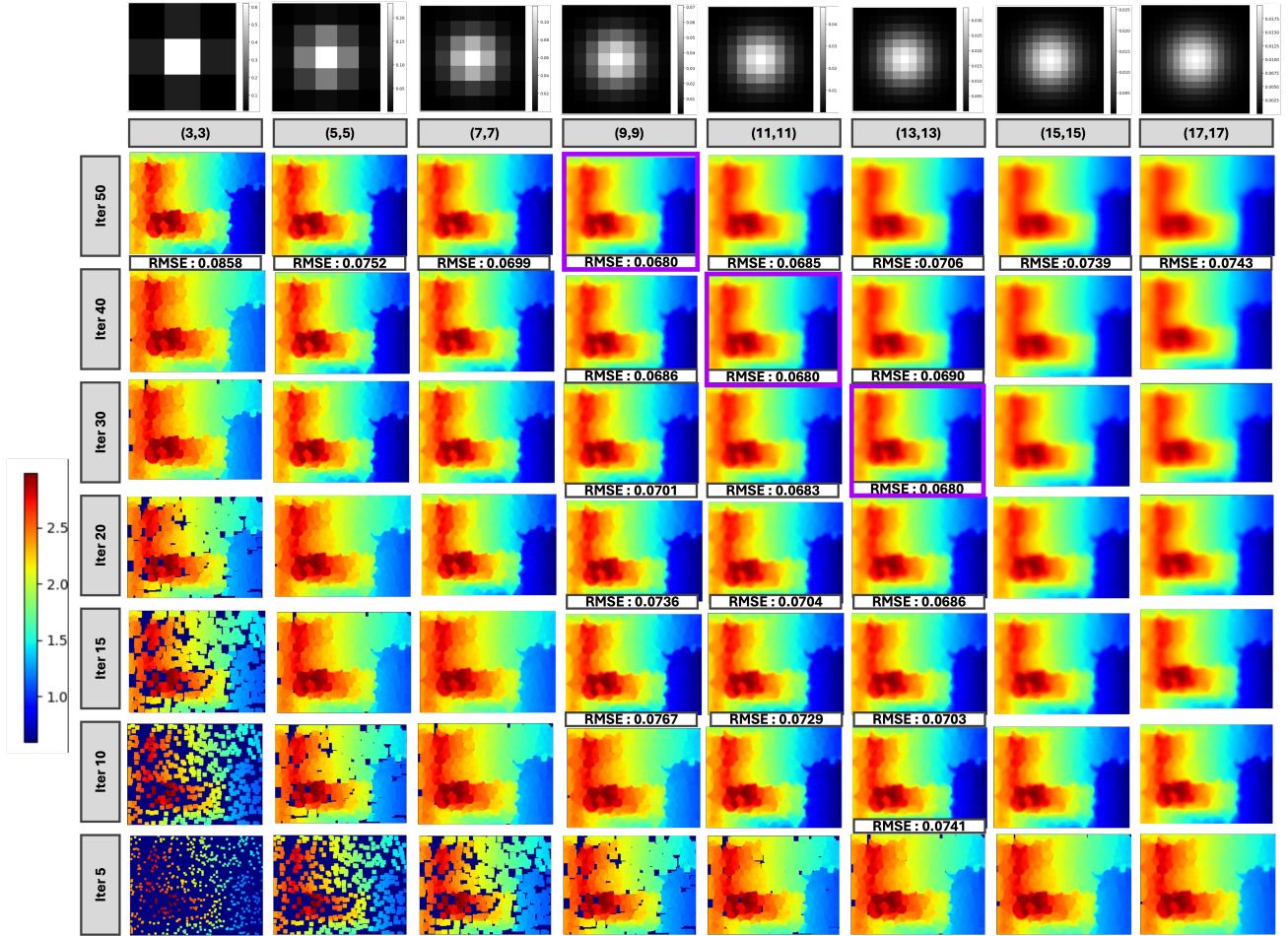


Figure 15: ArchBoost Analysis: Smooth Hole Filling

Result and Analysis

다음 Fig 16은 최종적으로 ArchBoost를 학습하여 얻은 결과이다. Initial RMSE는 0.0680, Predicted RMSE는 0.0530을 얻을 수 있었다. Baseline의 RMSE 0.0808보다 확연히 개선된 결과를 얻을 수 있었다. Fig 16의 정성적인 결과를 보면, (i) Smooth Hole Filling을 사용함으로써 부드러운 Initial을 얻을 수 있고, (ii) Auxiliary Depth Loss와 Average Pooling D2N을 함께 사용함으로써 예측된 Depth와 Normal이 부드러우면서도 경계를 잘 보존하도록 나왔다. 정확도도 올라간 것을 확인할 수 있다.

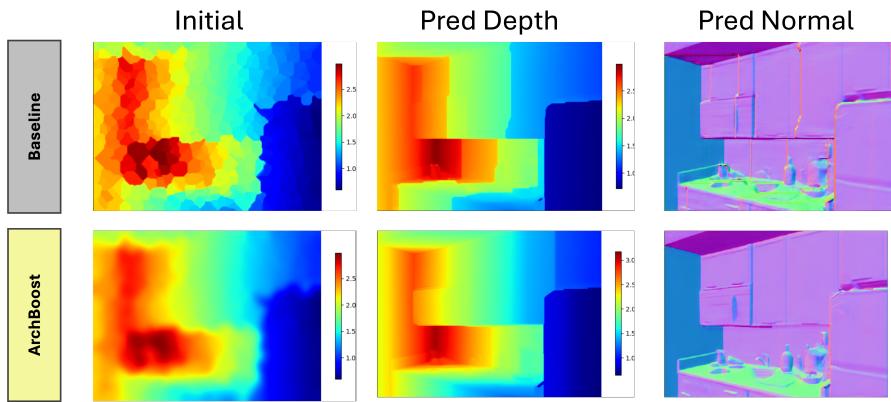


Figure 16: ArchBoost Analysis: Result

3.3 DataBoost and Final Result

우리는 DataBoost 과정으로 ArchBoost Model에 대해 (i) 먼저 주어진 Single Sample에 대해 학습시킨후 (ii) 해당 모델에 이어서 Augmentation한 1000개 Sample에 대해 Transfer Learning을 수행하였다. 결과적으로 Test Sample에 대해 다음 Fig 17과 같은 결과를 얻을 수 있었다. Baseline, ArchBoost, Arch+DataBoost로 갈수록 해당 Test Sample에 대해 더 정확한 결과를 얻을 수 있었다.

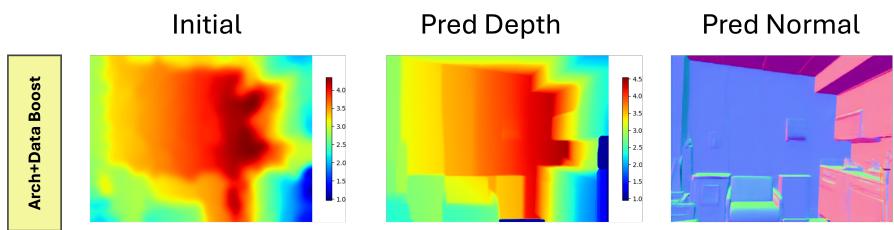


Figure 17: DataBoost and Final Result

References

- [1] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241, 2015.
- [2] X. Liao, *Pytorch-UNet Implementation*, GitHub repository, Available at: <https://github.com/xiaopeng-liao/Pytorch-UNet>, Accessed: 2025-06-11.