



길다영, 김현우

HorizonNet

2021-1 Computer Vision Project



논문 해석

2. Related Work

3. Approach

The goal of our approach is to estimate Manhattan room layout from a panoramic image that covers 360° H-FOV. Unlike conventional dense prediction (target output size = $\mathcal{O}(HW)$) for layout estimation using deep learning [4, 9, 7, 15, 19, 23, 33], we formulate the problem as regressing the boundaries and classifying the corner for each column of image (target output size = $\mathcal{O}(W)$). The proposed HorizonNet trained for predicting the $\mathcal{O}(W)$ target is presented in Sec. 3.1. In Sec. 3.2, we introduce a simple yet fast and effective post-processing procedure to derive the layout from output of HorizonNet. Finally in Sec. 3.3, we introduce *Pano Stretch Data Augmentation* which effectively augments the training data on-the-fly by stretching the image and ground-truth layout along x or z axis (Fig. 5).

All training and test images are pre-processed by the panoramic image alignment algorithm mentioned in [36]. Our approach exploits the properties of the aligned panoramas that the wall-wall boundaries are vertical lines under equirectangular projection. Therefore, we can use only one value to indicate the column position of wall-wall boundary instead of two (each for a boundary endpoint).

$\mathcal{O}(W)$ 대상을 예측하기 위해 훈련된 제안된 HorizonNet은 3.1절에 제시되어 있다.

3.2절에서는 HorizonNet의 출력에서 레이아웃을 도출하는 간단하면서도 빠르고 효과적인 후처리 절차를 소개한다.

마지막으로, 3.3절에서는 x 또는 z 축을 따라 영상과 지상 실측 레이아웃을 확장하여 훈련 데이터를 효과적으로 증강하는 Pano Stretch Data Augmentation을 소개한다(그림 5).

모든 교육 및 테스트 영상은 [36]에서 언급한 파노라마 이미지 정렬 알고리즘에 의해 사전 처리됩니다.

우리의 접근 방식은 벽-벽 경계선이 등각 투영 하에서 수직선이라는 정렬된 파노라마의 특성을 이용한다.

따라서 두 개(경계 끝점 각각) 대신 하나의 값만 사용하여 벽-벽 경계 열 위치를 표시할 수 있습니다.

[36] : 경계와 모서리의 2D 전체 이미지 표시는 평활화 후에도 95% 영점 값이 됩니다

Manhattan World 가정

- 실내, 실외의 영상은 대부분 직육면체를 이룸
- 영상에 나타나는 평면들이 3차원상에서 서로 직교하는 평면들로만 이루어져 있다는 가정

<http://library.kaist.ac.kr/search/detail/view.do?bibCtrlNo=569331&flag=t>

Manhattan world 가정

각 끝점이 아닌 경계선이기 때문에 하나의 값

3.1 Horizon Net

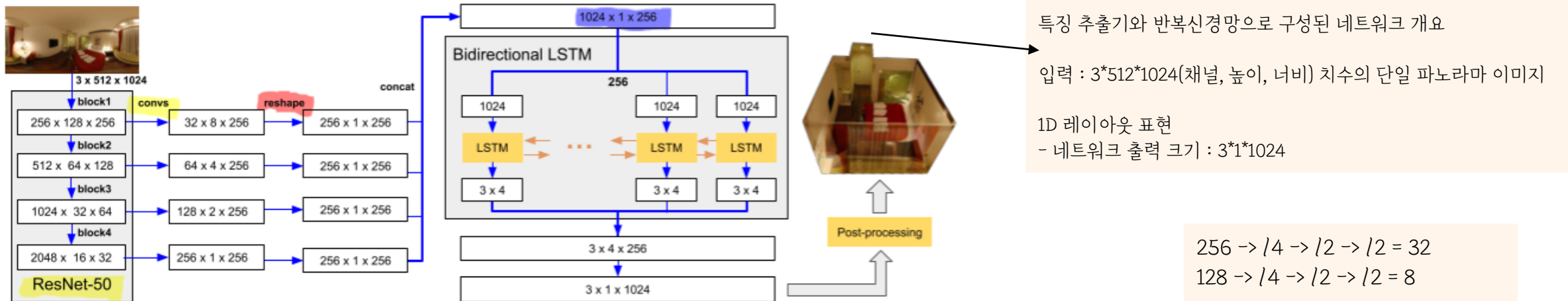


Figure 2: An illustration of the HorizonNet architecture.

Feature Extractor: We adopt ResNet-50 [11] as our feature extractor. The output of each block of ResNet-50 has half spatial resolution compared to that of the previous block. To capture both low-level and high-level features, each block of the ResNet-50 contains a sequence of convolution layers in which the number of channels and the height is reduced by a factor of 8 ($= 2 \times 2 \times 2$) and 16 ($= 4 \times 2 \times 2$), respectively. More specifically, each block contains three convolution layers with 4×1 , 2×1 , 2×1 kernel size and stride, and the number of channels after each Conv is reduced by a factor of 2. All the extracted features from each layer are upsampled to the same width 256 (a quarter of input image width) and reshaped to the same height. The final concatenated feature map is of size $1024 \times 1 \times 256$. The activation function after each Conv is ReLU except the final layer in which we use Sigmoid for y_w and an identity function for y_c, y_f . We have tried various settings for the feature extractor, including deeper ResNet-101, different designs of the convolution layers after each ResNet block, and upsampling to the image width 1024, and find that the results are similar. Therefore, we stick to the simpler and computationally efficient setting.

각 블록은 $4 \times 1, 2 \times 1, 2 \times 1$ 커널 크기와 보폭을 가진 세 개의 컨볼루션 레이어를 포함하고 있으며, 각 Conv 이후의 채널 수는 2의 계수만큼 감소한다.

각 계층에서 추출된 모든 형상은 동일한 너비 256(입력 이미지 폭의 1/4)으로 업샘플링되고 동일한 높이로 재구성됩니다.

마지막으로 연결된 특징 맵의 크기는 $1024 \times 1 \times 256$ 이다.

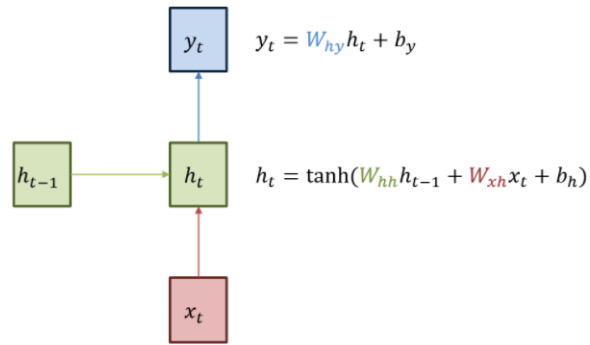
우리는 더 깊은 ResNet-101, 각 ResNet 블록 이후 컨볼루션 계층의 다른 설계, 업샘플링을 포함한 기능 추출기에 대한 다양한 설정을 시도했다. 이미지 폭 1024에 대한 정보를 확인하고 결과가 비슷하다는 것을 확인합니다.

RNN & LSTM

RNN

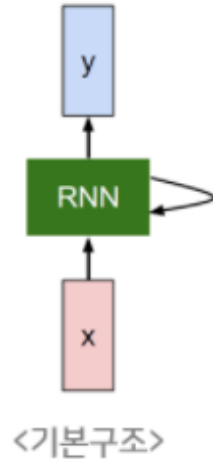
RNN(Recurrent Neural Networks)

: 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망의 한 종류



RNN의 기본 구조는 위 그림과 같습니다. 녹색 박스는 히든 state를 의미합니다. 빨간 박스는 인풋 x , 파란 박스는 아웃풋 y 입니다. 현재 상태의 히든 state h_t 는 직전 시점의 히든 state h_{t-1} 를 받아 갱신됩니다.

현재 상태의 아웃풋 y_t 는 h_t 를 전달받아 갱신되는 구조입니다. 수식에서도 알 수 있듯 히든 state의 활성화함수(activation function)은 비선형 함수인 하이퍼볼릭탄젠트(tanh)입니다.



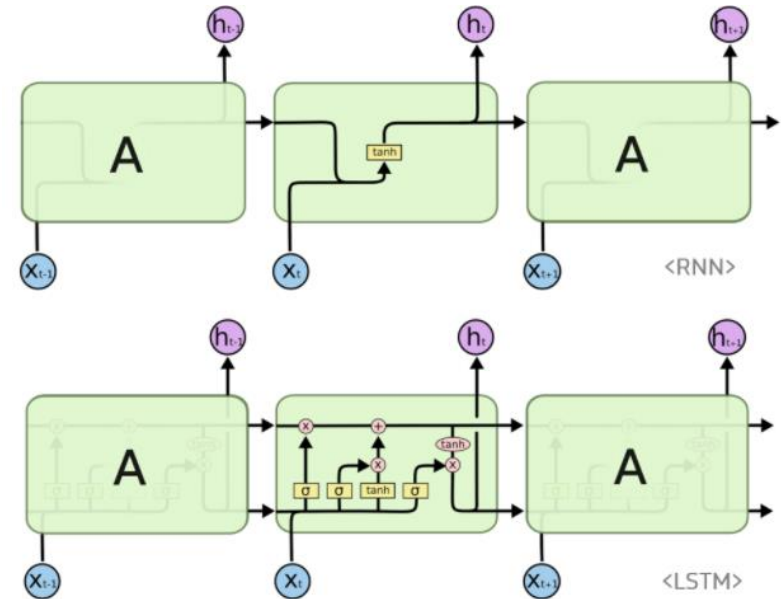
<기본구조>

LSTM

LSTM(Long Short-Term Memory models)

: RNN의 일종

: RNN은 관련 정보과 그 정보를 사용하는 지점 사이의 거리가 멀 경우 역전파시 그래디언트가 점차 줄어 학습능력이 크게 저하된다 (vanishing gradient problem) → 이를 극복하기 위해 고안된 것이 LSTM



3.1 Horizon Net



Yc : 천장-벽 경계
Yf : 바닥-벽 경계
Yw : 벽-벽 경계

Figure 3: Visualization of our 1D ground truth representations. y_w denotes the existence probability of wall-wall boundary. y_c, y_f (plotted in green and blue) denote the positions of the ceiling-wall boundary and floor-wall boundary respectively. For better visualization, we plot y_w, y_c, y_f with line width greater than one pixel.

1D Layout Representation: The size of network output is $3 \times 1 \times 1024$. As illustrated in Fig. 3, two of the three output channels represent the ceiling-wall (y_c) and the floor-wall (y_f) boundary position of each image column, and the other one (y_w) represents the existence of wall-wall boundary (i.e. corner). The values of y_c and y_f are normalized to $[-\pi/2, \pi/2]$. Since defining y_w as a binary-valued vector with 0/1 labels would make it too sparse to detect (only 4 out of 1024 non-zero values for simple cuboid layout), we set $y_w(i) = c^{dx}$ where i indicates the i th column, dx is the distance from the i th column to the nearest column where wall-wall boundary exists, and c is a constant. To check the robustness of our method against the choice of c , we have tried 0.6, 0.8, 0.9, 0.96, 0.99 and get similar results. Therefore, we stick to $c = 0.96$ for all the experiments. One benefit of using 1D representation is that it is less affected by zero dominant backgrounds. 2D whole-image representations of boundaries and corners would result in 95% zero values even after smoothing [36]. Our 1D boundaries representation introduces no zero backgrounds because the prediction for each component of y_c or y_f is simply a real-valued regression to the ground truth. The 1D wall-wall (corners) representation also changes the peak-background ratio of ground truth from $\frac{2N}{512 \times 1024}$ to $\frac{N}{1024}$ where N is the number of wall-wall corners. Therefore, the 1D wall-wall representation is also less affected by zero-dominated background. In addition, computation of 1D compact output is more efficient compared to 2D whole-image output. As depicted in Sec. 3.2, recovering the layout from our three 1D representations is simple, fast, and effective.

1D 벽면(코너) 표현은 또한 지면 진실의 피크-배경 비율을 $2N \times 512 \times 1024$ 에서 $N \times 1024$ 로 변경한다. 여기서 N 은 벽면 모서리의 수이다

3.2절에 설명된 바와 같이, 3개의 1D 표현에서 레이아웃을 복구하는 것은 간단하고 빠르게 효과적입니다.

Recurrent Neural Network for Capturing Global Information: Recurrent neural networks (RNNs) are capable of learning patterns and long-term dependencies from sequential data. Geometrically speaking, any corner of a room can be roughly inferred from the positions of other corners; therefore, we use the capability of RNN to capture global information and long-term dependencies. Intuitively, because LSTM [13], a type of RNN architecture, stores information about its prediction for other regions in the cell state, it has the ability to predict for occluded area accurately based on the geometric patterns of the entire room. In our model, RNN is used to predict y'_c, y'_f, y'_w column by column. That is, the sequence length of RNN is proportional to the image width. In our experiment, RNN predicts for four columns instead of one column per time step, which requires less computational time without loss of accuracy. As the y_c, y_f, y_w of a column is related to both its left and right neighbors, we adopt the bidirectional RNN [25] to capture the information from both sides. Fig. 7 and Table 1 demonstrate

방의 구석구석

다른 모서리의 위치로부터 대략적으로 추론할 수 있다.

우리의 실험에서, RNN은 한 단계당 하나의 열이 아닌 네 개의 열에 대해 예측한다.

열의 y_c, y_f, y_w 는 왼쪽과 오른쪽 이웃 모두와 관련이 있으므로 양방향 RNN[25]을 채택하여 양쪽에서 정보를 캡처한다.

그림 7과 표 1은 RNN이 있는 모델과 없는 모델 간의 차이를 보여준다.

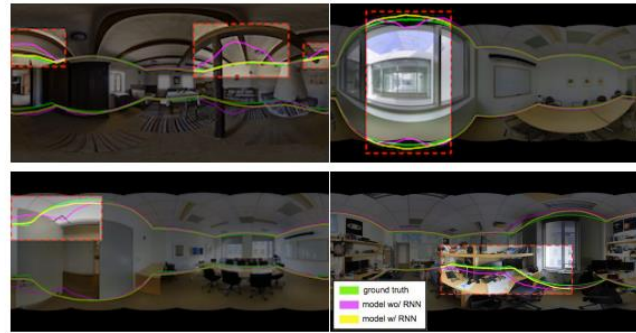


Figure 7: Visualization of model outputs with and without RNN. We plot the ground truth (green), outputs of the model with RNN (yellow), and outputs of the model without RNN (magenta). Both predictions are raw network outputs without post-processing. The model with RNN performs better than the model without RNN in images contain ceiling beam, black missing polar region caused by smaller camera V-FOV, and occluded area.

녹색 : ground truth

노란색 : RNN 사용한 모델의 출력

자홍색 : RNN 없는 모델의 출력

이미지에서 RNN이 있는 모델은 천장 빔, 더 작은 카메라 V-FOV로 인한 검은색 누락 극 영역 및 가려진 영역을 포함하는 RNN이 없는 모델보다 성능이 더 좋다.

3.2 Post-processing

그림 4a :
전처리 알고리즘이 파노라마의 수평 회전을 올바르게 정렬하지 못하는 예입니다.

3.2. Post-processing

We recover general room layouts that are not limited to cuboid under following assumptions: *i*) intersecting walls are perpendicular to each other (Manhattan world assumption); *ii*) all rooms have the one-floor-one-ceiling layout where floor and ceiling are parallel to each other; *iii*) camera height is 1.6 meters following [32]; *iv*) the pre-processing step correctly align the floor orthogonal to y-axis.

As described in Sec. 3.1, raw outputs of our deep model $y'_f, y'_c, y'_w \in \mathcal{R}^{1024}$ contain the layout information for each image column. Each value in y'_f and y'_c is the position of floor-wall boundary and ceiling-wall boundary at the corresponding image column. y'_w represents the probability of wall-wall existence of each image column.

Recovering the Floor and Ceiling Planes: For each column of the image, we can use the corresponding values in y'_f, y'_c to vote for the ceiling-floor distance. Based on the assumed camera height, we can project the floor-wall boundary y'_f from image to 3D XYZ position (they all shared the same Y). The ceiling-wall boundary y'_c shares the same 3D X, Z position with the y'_f on the same image column, and therefore the distance between floor and ceiling can be calculated. We take the average of results calculated from all image columns as the final floor-ceiling distance.

Recovering Wall Planes: We first find the prominent peaks on the estimated wall-wall probability y'_w with two criteria: *i*) the signal should be larger than any other signal within 5°H-FOV, and *ii*) the signal should be larger than 0.05.

Fig. 4a shows the projected y'_c (red points) on ceiling plane. The green lines are the detected prominent peaks which split the ceiling-wall boundary (red points) into multiple parts. To handle possibly failed horizontal alignment in the pre-processing step, we calculate the first principal component of each part, then rotate the scene by the aver-

우리는 다음과 같은 가정 하에서 cuboid로 제한되지 않는 일반적인 룸 레이아웃을 복구한다.

- I. 교차 벽은 서로 수직이다(Manhattan world가정).
- II. 바닥과 천장이 서로 평행한 모든 객실의 1층 1층 배치
- III. 카메라 높이는 1.6미터
- IV. 전처리 단계에서 바닥을 y축에 직교하도록 올바르게 정렬합니다.

바닥과 천장평면 복구 :

천장-바닥 거리 투표하기 위해 y_f (바닥-벽), y_c (천장-벽)에서 상응하는 값을 사용할 수 있다.
가정된 카메라 높이를 기반으로 이미지에서 3D XYZ 위치로 바닥- 벽 경계인 y_f 를 투영할 수 있습니다(모두 동일한 Y를 공유함).

천장-벽 경계 y_c 는 동일한 3D X, Z 위치를 동일한 영상 열에 있는 y_f 와 공유합니다.
그러므로 바닥과 천장 사이의 거리는 계산될 수 있다.

그림 4a는 천장에 투영된 y_c (빨간색)를 보여줍니다.

녹색 선은 천장-벽 경계(빨간색)를 여러 부분으로 분할하는 감지된 눈에 띄는 피크입니다.
그 다음 모든 첫 번째 주 구성 요소의 평균 4세 각도로 장면을 회전합니다(그림 4a의 오른쪽 상단 그림).
이제 우리는 두 종류의 벽을 가지고 있습니다: i) X축 직교 벽과 ii) Z축 직교 벽입니다.

인접 벽은 서로 직교해야 하므로 인접한 두 벽이 아직 구성되지 않은 벽만 직교 유형을 결정할 수 있습니다.
가장 많이 투표한 평면이 선택됩니다.

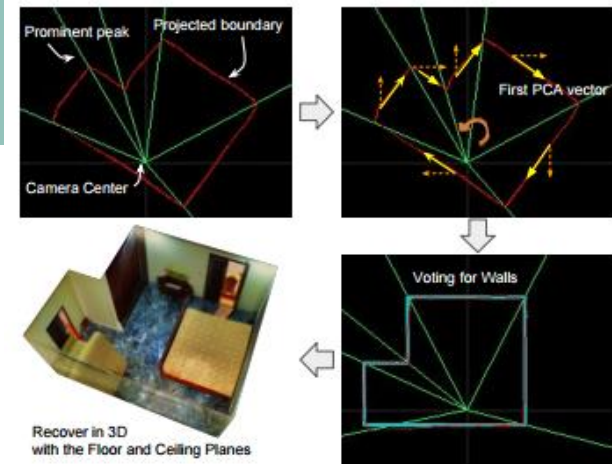
그림 4b에는 두 개의 인접한 벽이 이미 구성되어 있고 서로 직교할 때 발생하는 두 가지 특별한 사례가 나와 있습니다.

마지막으로, 모든 모서리의 XYZ 위치는 인접한 맨해튼 접속면 3개의 교차점에 따라 결정된다.

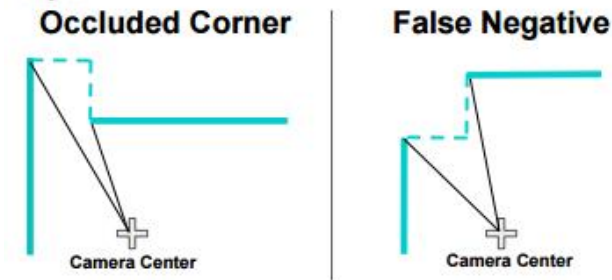
바닥 - 천장 복구 : 가정된 카메라 높이를 기반으로 바닥-벽 경계를 3d로 변경(동일한 y축 공유하여...) → 천장-벽 경계와 바닥-벽 경계 공유 → 이러한 방식으로 바닥과 천장 사이 거리 계산

그림4

빨간색 - 천장에 투영된 천장과 벽 경계 / 초록 선 : 빨간색을 모서리마다 분할 하는 듯 → 이걸 회전시키면서 직교인걸 파악하는 것 같다!(x-z축 직교 / y축 공유) → 그 후 투표를 통해 인접한 평면 결정
→ 직교 시 발생하는 두가지 x특별한 경우 있음(그림 4b)
→ 인접한 맨해튼 접속면 3개의 교차점에 따라 3d 상에서 모서리의 위치 결정



(a) Depicting how we recover the wall planes from our model output.



age angle of all first principal components (top right figure in Fig. 4a). So now we have two types of walls: *i*) X-axis orthogonal walls and *ii*) Z-axis orthogonal walls. We construct the walls from low to high variance suggested by the first principal component. Adjacency walls are forced to be orthogonal to each other, thus only walls whose two adjacent walls are not yet constructed have the freedom to decide the orthogonal type. We use a simple voting strategy: each projected red point votes for all planes within 0.16 meters (bottom right figure in Fig. 4a). The most voted plane is selected. Two special cases are depicted in Fig. 4b which occur when the two adjacency walls are already constructed and they are orthogonal to each other. Finally, the XYZ positions of all corners are decided according to the intersection of three adjacent Manhattan junction planes.

The time complexity of our post-processing procedure is $\mathcal{O}(W)$, where W is the image width. Thus the post-processing can be efficiently done; in average, it takes less than 20ms to finish.

4.1 Datasets & 4.2 Training Details

4.1. Datasets

We train and evaluate our model using the same dataset as LayoutNet [36]. The dataset consists of PanoContext dataset [32] and the extended Stanford 2D-3D dataset [1] annotated by [36]. To train our model, we generate $3 \times 1 \times 1024$ ground truth from the annotation. We follow the same training/validation/test split of LayoutNet.

4.2. Training Details

The Adam optimizer [16] is employed to train the network for 300 epochs with batch size 24 and learning rate 0.0003. The L1 Loss is used for the ceiling-wall boundary (y_c) and floor-wall boundary (y_f). The Binary Cross-Entropy Loss is used for the wall-wall corner (y_w). The network is implemented in PyTorch [20]. It takes four hours to finish the training on three NVIDIA GTX 1080 Ti GPUs.

The data augmentation techniques we adopt include standard left-right flipping, panoramic horizontal rotation, and luminance change. Moreover, we exploit the proposed Pano Stretch Data Augmentation (Sec. 3.3) during training. The stretching factors k_x, k_z are sampled from uniform distribution $U[1, 2]$, and then take the reciprocals of sampled values with probability 0.5. The process time of Pano Stretch Data Augmentation is roughly 130ms per 512×1024 RGB image. Therefore, it is feasible to be applied on-the-fly during training.

우리는 LayoutNet[36]과 동일한 데이터 세트를 사용하여 모델을 교육하고 평가한다. 데이터 세트는 [32] PanoContext 데이터 세트와 [36] 주석이 달린 확장된 Stanford 2D-3D 데이터 세트[1]로 구성된다. 우리의 모델을 훈련시키기 위해, 우리는 주석으로부터 $3 * 1 * 1024$ 개의 ground truth를 생성한다. LayoutNet의 동일한 교육/검증/테스트 분할을 수행합니다.

Adam Optimizer [16]은 배치 크기 24와 학습 속도 0.0003을 가진 300 epoch에 대한 네트워크를 훈련시키는 데 사용된다.

L1 손실은 천장-벽 경계(y_c) 및 바닥-벽 경계(y_f)에 사용됩니다. Binary CrossEntropy Loss는 벽면 코너(y_w)에 사용됩니다. 네트워크는 PyTorch [20]에서 구현된다.

NVIDIA GTX 1080Ti GPU 3개에 대한 교육을 완료하는 데 4시간이 소요됩니다. 우리가 채택하는 데이터 확대 기법에는 표준 좌측-우측 플립, 파노라마 수평 회전 및 휘도 변화가 포함된다. 또한, 우리는 훈련 중에 제안된 Pano Stretch 데이터 확대(3.3절)를 활용한다.

스트레칭 인자 k_x, k_z 는 균일 분포 $U[1, 2]$ 에서 샘플링된 다음 0.5 확률로 샘플링된 값의 역수를 구한다.

Pano Stretch Data Augment의 프로세스 시간은 512×1024 RGB 이미지당 약 130ms이다. 따라서 훈련 중에 즉시 적용하는 것이 가능하다.

→ 트레이닝에 4시간 실화..?

Method	3D IoU(%)	Corner error(%)	Pixel error(%)
Train on PanoContext dataset			
PanoContext [32]	67.23	1.60	4.55
LayoutNet [36]	74.48	1.06	3.34
DuLa-Net [30]	77.42	-	-
CFL [8]	78.79	0.79	2.49
ours	82.17	0.76	2.20
Train on PanoContext + Stnfd.2D3D datasets			
LayoutNet [36]	75.12	1.02	3.18
ours	84.23	0.69	1.90

Table 1. Quantitative results of cuboid layout estimation evaluated on the PanoContext [32] dataset. Our method outperforms all existing methods under all settings.

4.3 Cuboid Room Results

4.3. Cuboid Room Results

We generate cuboid room by only selecting the four most prominent peaks in the post-processing step (Sec. 3.2).

Quantitative Results: Our approach is evaluated on three standard metrics: *i)* **3D IoU:** intersection over union between 3D layout constructed from our prediction and the ground truth; *ii)* **Corner Error:** average Euclidean distance between predicted corners and ground-truth corners (normalized by image diagonal length); *iii)* **Pixel Error:** pixel-wise error between predicted surface classes and ground-truth surface classes.

The quantitative results of different training and testing settings are summarized in Table 1 and Table 2. To clarify the difference, the input resolution of DuLa-Net [30] and CFL [8] are 256×512 while LayoutNet [36] and ours are 512×1024 . Other than conventional augmentation technique, CFL [8] is trained with Random Erasing while ours is trained with the proposed Pano Stretch. DuLa-Net [30] did not report corner errors and pixel errors. Our approach achieves state-of-the-art performance and outperforms existing methods under all settings.

Qualitative Results: The qualitative results are shown in Fig. 6. We present the results from the best to the worst based on their corner errors. Please see more results in the supplemental materials.

Computation time: The 1D layout representation is easy to compute. Forward passing a single 512×1024 RGB image takes 8ms and 50ms for our HorizonNet with and without RNN respectively. The post-processing step for extracting layout from our 1D representation takes only 12ms. We evaluate the result on a single NVIDIA Titan X GPU and an Intel i7-5820K 3.30GHz CPU. The reported execution time is averaged across all the testing data.

우리는 후 처리 단계에서 가장 두드러진 네 개의 피크를 선택하여 큐보이드 룸을 생성한다(3.2절).

정량적 결과:

우리의 접근 방식은 다음 세 가지 표준 메트릭스에서 평가됩니다.

- i) 3D IoU: 우리의 예측에서 생성된 3D 배치와 실측 진실 사이의 결합에 대한 교차점
- ii) 코너 오류: 예측 코너와 지면 실측 코너 사이의 평균 유클리드 거리(이미지 대각선 길이로 정규화)
- iii) 픽셀 오류: 예측 표면 클래스와 지면 진실 표면 클래스 사이의 픽셀 단위 오류.

다른 훈련 및 시험 설정의 정량적 결과는 표 1과 표 2에 요약되어 있다.

차이를 명확히 하기 위해, DuLa-Net [30]과 CFL [8]의 입력 해상도는 256×512 이고, LayoutNet [36]과 우리의 입력 해상도는 512×1024 입니다.

기존의 증강 기법 외에 CFL[8]은 무작위 소거로 훈련되고, 반면 우리는 제안된 Pano Stretch로 훈련한다.

DuLa-Net [30]은 코너 오류와 픽셀 오류를 보고하지 않았습니다.

우리의 접근 방식은 최첨단 성능을 달성하고 모든 설정에서 기존 방법을 능가한다.

정성적 결과는 그림 6에 나타나 있다. 코너 오류를 기준으로 최고에서 최악으로 결과를 제시한다.

계산 시간:

1D 레이아웃 표현은 계산하기 쉽습니다.

단일 512×1024 RGB 이미지를 전달하려면 RNN 유무와 비사용 HorizonNet의 경우 각각 8ms와 50ms가 걸립니다.

1D 표현에서 레이아웃을 추출하기 위한 후 처리 단계는 12ms에 불과합니다.

단일 NVIDIA Titan X GPU와 Intel i7-5820K 3.30GHz CPU에서 결과를 평가한다. 보고된 실행 시간은 모든 테스트 데이터에 걸쳐 평균화됩니다.

Method	3D IoU(%)	Corner error(%)	Pixel error(%)
Train on PanoContext dataset			
PanoContext [32]	67.23	1.60	4.55
LayoutNet [36]	74.48	1.06	3.34
DuLa-Net [30]	77.42	-	-
CFL [8]	78.79	0.79	2.49
ours	82.17	0.76	2.20
Train on PanoContext + Stnfd.2D3D datasets			
LayoutNet [36]	75.12	1.02	3.18
ours	84.23	0.69	1.90

Table 1. Quantitative results of cuboid layout estimation evaluated on the PanoContext [32] dataset. Our method outperforms all existing methods under all settings.

왼쪽이 최고, 오른쪽이 최악
녹색선 : 실제 배치 / 주황선 : 추정



Figure 6: Qualitative results of cuboid layout estimation. The results are separately sampled from four groups that comprise results with the best 0-25%, 25-50%, 50-75% and 75-100% corner errors (displayed from the first to the fourth columns). The green lines are ground truth layout while the orange lines are estimated. The images in the first row are from PanoContext dataset [32] while second row are from Stanford 2D-3D dataset [1].

그림 6: 입체 배치 추정의 정성적 결과 결과는 최적의 0-25%, 25-50%, 50-75%, 75-100%의 코너 오차를 가진 결과를 구성하는 네 개의 그룹(첫 번째 열부터 네 번째 열까지 표시)에서 별도로 샘플링됩니다. 녹색 선은 실제 배치이고 주황색 선은 추정됩니다. 첫 번째 행의 이미지는 PanoContext 데이터 세트[32]에서, 두 번째 행은 Stanford 2D-3D 데이터 세트[1]에서 가져온 것입니다.

4.4 Ablation Study

too many computing resources. We can see that learning on our 1D $\mathcal{O}(W)$ layout representation is better than conventional dense $\mathcal{O}(HW)$ layout representation.

We observe that training with the proposed Pano Stretch Data Augmentation can always boost the performance. Note that the proposed data augmentation method can also be adopted in other tasks on panoramas and has the potential to increase their accuracy as well. See supplemental material for the experiment using Pano Stretch Data Augmentation on semantic segmentation task.

For the rows where RNN columns are unchecked, the RNN components shown in Fig 7 are replaced by fully connected layers. Our experiments show that using RNN in network architecture also improves performance. Fig. 7 shows some representative results with and without RNN. The raw output of the model with RNN is highly consistent with the Manhattan world even without post-processing, which

우리는 제안된 Pano Stretch Data Augment를 통한 훈련이 항상 성능을 향상시킬 수 있다는 것을 관찰한다.

우리의 실험은 네트워크 아키텍처에서 RNN을 사용하는 것도 성능을 향상시킨다는 것을 보여준다.

그림 7은 RNN을 포함하거나 포함하지 않은 몇 가지 대표적인 결과를 보여준다.

RNN을 사용한 모델의 원시 출력은 사후 처리가 없어도 맨해튼 세계와 매우 일치하며, 이는 RNN이 전체 룸의 기하학적 패턴을 캡처할 수 있는 능력을 보여준다.

RNN이 있는 모델이 없는 모델보다 성능이 좋다

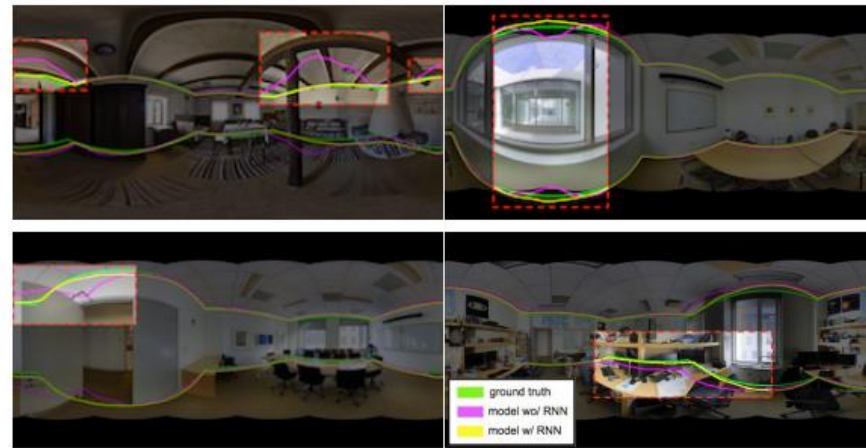


Figure 7: Visualization of model outputs with and without RNN. We plot the ground truth (green), outputs of the model with RNN (yellow), and outputs of the model without RNN (magenta). Both predictions are raw network outputs without post-processing. The model with RNN performs better than the model without RNN in images contain ceiling beam, black missing polar region caused by smaller camera V-FOV, and occluded area.

그림 7:

RNN 포함 및 제외 모델 출력 시각화

우리는 접지 진실(녹색), RNN(노란색)을 사용한 모델의 출력 및 RNN(자홍색)을 제외한 모델의 출력을 플롯한다.

두 예측 모두 후처리가 없는 원시 네트워크 출력입니다.

이미지에서 RNN이 있는 모델은 천장 빔, 더 작은 카메라 V-FOV로 인한 검은색 누락 극 영역 및 가려진 영역을 포함하는 RNN이 없는 모델보다 성능이 더 좋다.

4.5 Non-cuboid Room Results

4.5. Non-cuboid Room Results

Since the non-cuboid rooms in PanoContext and Stanford 2D-3D dataset are labeled as cuboids, our model is never trained to recognize non-cuboid layouts and concave corners. This bias makes our model tend to predict complex-shaped rooms as cuboids. To estimate general room layouts, we re-label 65 rooms from the training split to fine-tune our trained model. We fine-tune our model for 300 epochs with learning rate $5e-5$ and batch size 2.

To quantitatively evaluate the fine-tuning result on general-shaped rooms, we use 13-fold cross validation on the 65 re-annotated non-cuboid data. The results are summarized in Table 4. We depict some examples of reconstructed non-cuboid layouts from the testing and validation splits in Fig. 1 and Fig. 8. See supplemental material for more reconstructed layouts. The results show that our approach can work well on general room layout even with corners occluded by other walls.

Cuboid 모델에 대한 미세 조정, 교차 검증을 통해 non-cuboid 모델을 훈련시킨다. 결론적으로 잘 작동한다.

가려진 벽이 검게 보이는 듯!
파란색 선 : 추정

그림 8:

비큐보이드 레이아웃 추정의 정성적 결과
가려진 벽은 검은 색으로 가득 차 있다.

등각 이미지의 파란색 선은 추정된 룸 레이아웃 경계입니다.

PanoContext 및 Stanford 2D-3D 데이터 세트의 비큐보이드 룸은 큐보이드로 레이블이 지정되므로, 우리의 모델은 비큐보이드 레이아웃을 인식하고 모서리를 오목하도록 결코 훈련되지 않는다.

이 편향은 우리의 모델이 복잡한 모양의 방을 큐보이드로 예측하는 경향이 있게 만든다.

일반적인 룸 레이아웃을 추정하기 위해, 우리는 훈련 분할에서 65개의 룸에 레이블을 다시 지정하여 훈련된 모델을 미세 조정한다.

일반 모양의 룸에 대한 미세 조정 결과를 정량적으로 평가하기 위해 65개의 다시 주석 처리된 비 큐보이드 데이터에 대해 13배 교차 검증을 사용한다.

결과는 우리의 접근 방식이 다른 벽으로 가려진 모서리를 사용하더라도 일반적인 방 배치에서 잘 작동할 수 있다는 것을 보여준다.



Figure 8: Qualitative results of non-cuboid layout estimation. The occluded walls are filled with black. The blue lines in the equirectangular images are the estimated room layout boundary.

5. Conclusion

5. Conclusion

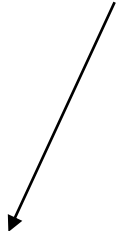
We have presented a new 1D representation for the task of estimating room layout from a panorama. The proposed HorizonNet trained with such 1D representation outperforms previous state-of-the-art methods and requires fewer computation resources. Our post-processing method which recovers 3D layout from the model output is fast and effective, and it also works for complex room layouts even with occluded corners. The proposed Pano Stretch Data Augmentation further improves our results, and can also be applied to the training procedure of other panorama tasks for potential improvement.

우리는 파노라마에서 룸 레이아웃을 추정하는 작업을 위한 새로운 1D 표현을 제시했다.

그러한 1D 표현을 사용하여 훈련된 제안된 HorizonNet은 이전의 최첨단 방법을 능가하고 더 적은 계산 리소스를 필요로 한다.

모델 출력에서 3D 레이아웃을 복구하는 우리의 후처리 방법은 빠르고 효과적이며, 막힌 모서리가 있더라도 복잡한 룸 레이아웃에 효과적이다.

제안된 Pano Stretch 데이터 확대는 우리의 결과를 더욱 향상시키며, 잠재적 개선을 위한 다른 파노라마 작업의 훈련 절차에도 적용할 수 있다.



1D 표현 : 3.1과 3.2가 적용된 빨간, 파란, 녹색 선이 있는 것들을 말하는 것 같다.
꽤든 효과적임.



코드 해석

Preprocess.py → inference.py → post_proc.py → layout_viewer.py

```
# Process each input
for i_path in paths:
    print('Processing', i_path, flush=True)

    # Load and cat input images
    img_ori = np.array(Image.open(i_path).resize((1024, 512), Image.BICUBIC))[...,:3]

    # VP detection and line segment extraction
    _, vp, _, _, panoEdge, _, _ = panoEdgeDetection(img_ori,
                                                    qError=args.q_error,
                                                    refineIter=args.refine_iter)

    # panoEdge : image for visualize line segments
    panoEdge = (panoEdge > 0)

    # Align images with VP
    i_img = rotatePanorama(img_ori / 255.0, vp[2::-1])
    l_img = rotatePanorama(panoEdge.astype(np.float32), vp[2::-1])

    # Dump results
    basename = os.path.splitext(os.path.basename(i_path))[0]
    if args.rgbonly:
        path = os.path.join(args.output_dir, '%s.png' % basename)
        Image.fromarray((i_img * 255).astype(np.uint8)).save(path)
    else:
        path_VP = os.path.join(args.output_dir, '%s_VP.txt' % basename)
        path_i_img = os.path.join(args.output_dir, '%s_aligned_rgb.png' % basename)
        path_l_img = os.path.join(args.output_dir, '%s_aligned_line.png' % basename)

        Image.fromarray((i_img * 255).astype(np.uint8)).save(path_i_img)
        Image.fromarray((l_img * 255).astype(np.uint8)).save(path_l_img)
```

파노라마 사진에서
edge와 소실점을 찾음

소실점과 line_Segment를 찾아서 정렬을 하고, 이미지를 저장.

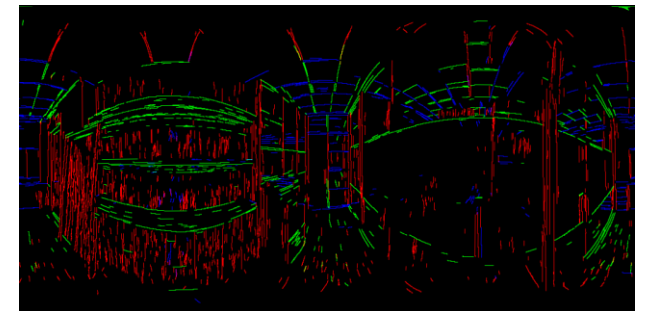
결과물



Demo424_aligned_rgb.raw.png

1	-0.000345	0.002584	0.999997
2	0.000766	0.999993	-0.003688
3	1.000000	-0.000765	0.000347
4			

Demo424_VP.txt



Demo424_aligned_line.png

구부러진 사진을 바로 세움

소실점 가진 이미지를 align
(구부러진 사진이 펴지는 것 의미)

소실점 찾음.

찾은 edge

저장된 이미지를 다른 파일로
넘기는 부분

Preprocess.py → inference.py → post_proc.py → layout_viewer.py

```
if __name__ == '__main__':  
    # Prepare image to processed  
    paths = sorted(glob.glob(args.img_glob))  
    if len(paths) == 0:  
        print('no images found')  
    for path in paths:  
        assert os.path.isfile(path), '%s not found' % path  
  
    # Check target directory  
    if not os.path.isdir(args.output_dir):  
        print('Output directory %s not existed. Create one.' % args.output_dir)  
        os.makedirs(args.output_dir)  
    device = torch.device('cpu' if args.no_cuda else 'cuda')  
  
    # Loaded trained model  
    net = utils.load_trained_model(HorizonNet, args.pth).to(device)  
    net.eval()
```

Preprocess된
이미지 불러옴.

Trained model(pth 파일) 불러옴

Inference()로 넘어감.

```
# Inferencing  
with torch.no_grad():  
    for i_path in tqdm(paths, desc='Inferencing'):  
        k = os.path.split(i_path)[-1][:4]  
  
        # Load image  
        img_pil = Image.open(i_path)  
        if img_pil.size != (1024, 512):  
            img_pil = img_pil.resize((1024, 512), Image.BICUBIC)  
        img_ori = np.array(img_pil)[..., :3].transpose([2, 0, 1]).copy()  
        x = torch.FloatTensor([img_ori / 255])  
  
        # Inferencing corners  
        cor_id, z0, z1, vis_out = inference(net, x, device,  
                                           args.flip, args.rotate,  
                                           args.visualize,  
                                           args.force_cuboid,  
                                           args.min_v, args.r)  
  
        # Output result  
        with open(os.path.join(args.output_dir, k + '.json'), 'w') as f:  
            json.dump({  
                'z0': float(z0),  
                'z1': float(z1),  
                'uv': [[float(u), float(v)] for u, v in cor_id],  
            }, f)  
  
        if vis_out is not None:  
            vis_path = os.path.join(args.output_dir, k + '.raw.png')  
            vh, vw = vis_out.shape[:2]  
            Image.fromarray(vis_out)\  
                .resize((vw//2, vh//2), Image.LANCZOS)\  
                .save(vis_path)
```

Preprocess.py → inference.py → post_proc.py → layout_viewer.py

```
def inference(net, x, device, flip=False, rotate=[], visualize=False,
             force_cuboid=True, min_v=None, r=0.05):
    """
    net      : the trained HorizonNet
    x        : tensor in shape [1, 3, 512, 1024]
    flip     : flipping testing augmentation
    rotate   : horizontal rotation testing augmentation
    """

    H, W = tuple(x.shape[2:])

    # Network feedforward (with testing augmentation)
    x, aug_type = augment(x, flip, rotate)
    y_bon_, y_cor_ = net(x.to(device))
    y_bon_ = augment_undo(y_bon_.cpu(), aug_type).mean(0)
    y_cor_ = augment_undo(torch.sigmoid(y_cor_).cpu(), aug_type).mean(0)

    # Visualize raw model output
    if visualize:
        vis_out = visualize_a_data(x[0],
                                   torch.FloatTensor(y_bon_[0]),
                                   torch.FloatTensor(y_cor_[0]))
    else:
        vis_out = None

    y_bon_ = (y_bon_[0] / np.pi + 0.5) * H - 0.5
    y_cor_ = y_cor_[0, 0]
```

```
def augment(x_img, flip, rotate):
    x_img = x_img.numpy()
    aug_type = []
    x_imgs_augmented = [x_img]
    if flip:
        aug_type.append('flip')
        x_imgs_augmented.append(np.flip(x_img, axis=-1))
    for shift_p in rotate:
        shift = int(round(shift_p * x_img.shape[-1]))
        aug_type.append('rotate %d' % shift)
        x_imgs_augmented.append(np.roll(x_img, shift, axis=-1))
    return torch.FloatTensor(np.concatenate(x_imgs_augmented, 0)), aug_type
```

Supplementary of HorizonNet CVPR'19

Part 1 Pano Stretch Data Augmentation Demo

$k_x = 1.0, k_z = 1.0$ (original)



$k_x = 2.0, k_z = 1.0$



$k_x = 1.0, k_z = 2.0$



$k_x = 2.0, k_z = 2.0$



Augmentation : 양적인 결과 계산
정확도를 높이는 데 사용됨.

Preprocess.py → inference.py → post_proc.py → layout_viewer.py

```
# Detect wall-wall peaks
if min_v is None:
    min_v = 0 if force_cuboid else 0.05
r = int(round(W * r / 2))
N = 4 if force_cuboid else None
xs_ = find_N_peaks(y_cor_, r=r, min_v=min_v, N=N)[0]
```

Find_N_peaks()
: prominent peak를 찾는 과정

```
# Generate wall-walls
cor, xy_cor = post_proc.gen_ww(xs_, y_bon[0], z0, tol=abs(0.16 * z1 / 1.6), force_cuboid=force_cuboid)
if not force_cuboid:
```

```
# Check valid (for fear self-intersection)
xy2d = np.zeros((len(xy_cor), 2), np.float32)
for i in range(len(xy_cor)):
    xy2d[i, xy_cor[i]['type']] = xy_cor[i]['val']
    xy2d[i, xy_cor[i-1]['type']] = xy_cor[i-1]['val']
```

```
if not Polygon(xy2d).is_valid:
    print(
        'Fail to generate valid general layout!! '
        'Generate cuboid as fallback.',
        file=sys.stderr)
xs_ = find_N_peaks(y_cor_, r=r, min_v=0, N=4)[0]
cor, xy_cor = post_proc.gen_ww(xs_, y_bon[0], z0, tol=abs(0.16 * z1 / 1.6), force_cuboid=True)
```

Gen_ww()
: cuboid, non-cuboid 구분
: gen_ww_cuboid() -> vote 함수
: gen_ww_general() -> vote 함수

```
# Expand with btn coory
cor = np.hstack([cor, post_proc.infer_coory(cor[:, 1], z1 - z0, z0)[:, None]])
```

```
# Collect corner position in equirectangular
cor_id = np.zeros((len(cor)*2, 2), np.float32)
for j in range(len(cor)):
    cor_id[j*2] = cor[j, 0], cor[j, 1]
    cor_id[j*2 + 1] = cor[j, 0], cor[j, 2]
```

```
# Normalized to [0, 1]
cor_id[:, 0] /= W
cor_id[:, 1] /= H
```

```
return cor_id, z0, z1, vis_out
```

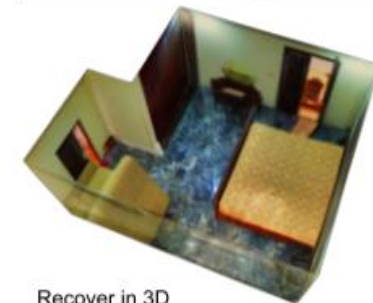
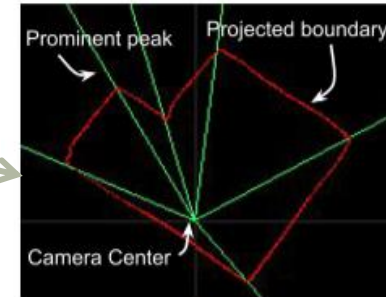
```
def gen_ww_general(init_coory, xy, gpid, tol):
    xy_cor = []
    assert len(init_coory) == len(np.unique(gpid))

    # Candidate for each part separated by wall-wall boundary
    for j in range(len(init_coory)):
        now_x = xy[gpid == j, 0]
        now_y = xy[gpid == j, 1]
        new_x, x_score, x_l1 = vote(now_x, tol)
        new_y, y_score, y_l1 = vote(now_y, tol)
        u0 = np.coorx2u(init_coory[(j - 1 + len(init_coory)) % len(init_coory)])
        u1 = np.coorx2u(init_coory[j])
        if (x_score, -x_l1) > (y_score, -y_l1):
            xy_cor.append({'type': 0, 'val': new_x, 'score': x_score, 'action': 'ori', 'gpid': j, 'u0': u0, 'u1': u1, 'tbd': True})
        else:
            xy_cor.append({'type': 1, 'val': new_y, 'score': y_score, 'action': 'ori', 'gpid': j, 'u0': u0, 'u1': u1, 'tbd': True})
```

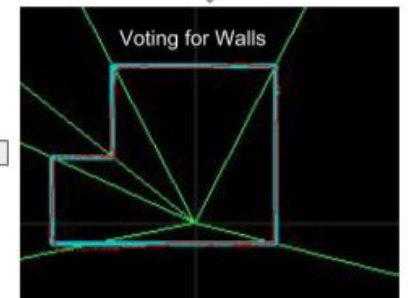
```
def gen_ww_cuboid(xy, gpid, tol):
    xy_cor = []
    assert len(np.unique(gpid)) == 4

    # For each part separated by wall-wall peak, voting for a wall
    for j in range(4):
        now_x = xy[gpid == j, 0]
        now_y = xy[gpid == j, 1]
        new_x, x_score, x_l1 = vote(now_x, tol)
        new_y, y_score, y_l1 = vote(now_y, tol)
        if (x_score, -x_l1) > (y_score, -y_l1):
            xy_cor.append({'type': 0, 'val': new_x, 'score': x_score})
        else:
            xy_cor.append({'type': 1, 'val': new_y, 'score': y_score})
```

For문의 개수 차이
Cuboid : 4개
Non-cuboid : 모서리 개수



Recover in 3D with the Floor and Ceiling Planes



```
def gen_ww(init_coory, coory, z=50, coorW=1024, coorH=512, floorW=1024, floorH=512, tol=3, force_cuboid=True):
    gpid = get_gpid(init_coory, coorW)
    coor = np.hstack([np.arange(coorW)[:, None], coory[:, None]])
    xy = np.coor2xy(coor, z, coorW, coorH, floorW, floorH)
```

```
# Generate wall-wall
if force_cuboid:
    xy_cor = gen_ww_cuboid(xy, gpid, tol)
else:
    xy_cor = gen_ww_general(init_coory, xy, gpid, tol)
```

```
# Ceiling view to normal view
cor = []
for j in range(len(xy_cor)):
    next_j = (j + 1) % len(xy_cor)
    if xy_cor[j]['type'] == 1:
        cor.append((xy_cor[next_j]['val'], xy_cor[j]['val']))
    else:
        cor.append((xy_cor[j]['val'], xy_cor[next_j]['val']))
cor = np_xy2coor(np.array(cor), z, coorW, coorH, floorW, floorH)
cor = np.roll(cor, -2 * cor[:, 2, 0].argmin(), axis=0)
```

```
return cor, xy_cor
```

Post_proc.py



코드 해석

주식 보정 전처리 → 1D Layout 표현 → 특징 추출기 (Feature Extractor) → 순환 신경망 (Recurrent Neural Network for Capturing Global Information) → Post-processing → Pano Stretch 데이터 확장 (Data Augmentation)

```
# Process each input
for i_path in paths:
    print('Processing', i_path, flush=True)

    # Load and cat input images
    img_ori = np.array(Image.open(i_path).resize((1024, 512), Image.BICUBIC))[:, :, :3]

    # VP detection and line segment extraction
    _, vp, _, _, panoEdge, _, _ = panoEdgeDetection(img_ori,
                                                    qError=args.q_error,
                                                    refineIter=args.refine_iter)

    # panoEdge : image for visualize line segments
    panoEdge = (panoEdge > 0)

    # Align images with VP
    i_img = rotatePanorama(img_ori / 255.0, vp[2::-1])
    l_img = rotatePanorama(panoEdge.astype(np.float32), vp[2::-1])

    # Dump results
    basename = os.path.splitext(os.path.basename(i_path))[0]
    if args.rgonly:
        path = os.path.join(args.output_dir, '%s.png' % basename)
        Image.fromarray((i_img * 255).astype(np.uint8)).save(path)
    else:
        path_VP = os.path.join(args.output_dir, '%s_VP.txt' % basename)
        path_i_img = os.path.join(args.output_dir, '%s_aligned_rgb.png' % basename)
        path_l_img = os.path.join(args.output_dir, '%s_aligned_line.png' % basename)

        Image.fromarray((i_img * 255).astype(np.uint8)).save(path_i_img)
        Image.fromarray((l_img * 255).astype(np.uint8)).save(path_l_img)
```

파노라마 사진에서
edge와 소실점을 찾음

소실점 가진 이미지를 align
(구부러진 사진이 펴지는 것 의미)

저장된 이미지를 다른 파일로
넘기는 부분

Preprocess.py

소실점과 line_Segment를 찾아서 정렬을 하고, 이미지를 저장.

구부러진 사진을 바로 세움

소실점 찾음.

찾은 edge

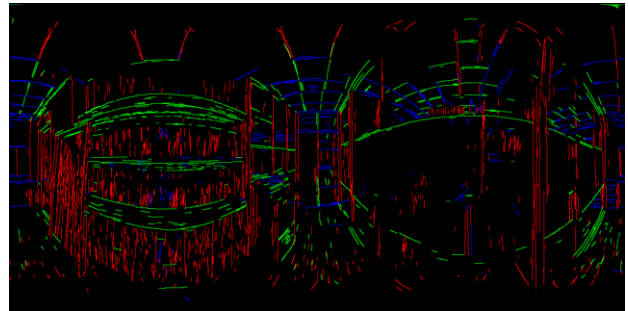
결과물



Demo424_aligned_rgb.raw.png

1	-0.000345	0.002584	0.999997
2	0.000766	0.999993	-0.003688
3	1.000000	-0.000765	0.000347
4			

Demo424_VP.txt



Demo424_aligned_line.png

수직 보정 전처리 → 1D Layout 표현 → 특징 추출기 (Feature Extractor) → 순환 신경망 (Recurrent Neural Network for Capturing Global Information) → Post-processing → Pano Stretch 데이터 확장 (Data Augmentation)



수직 보정 전처리

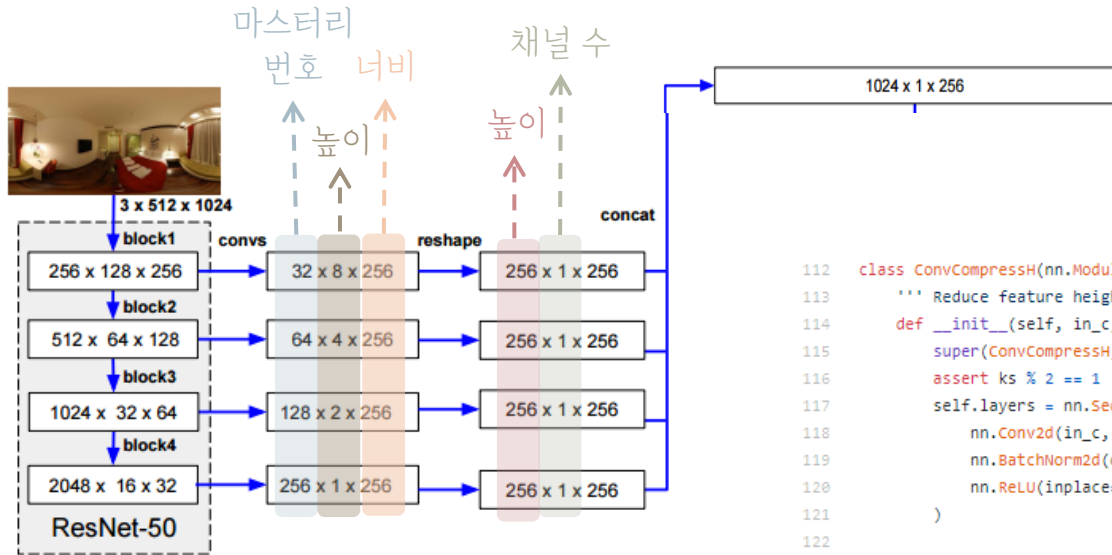


y_c : 천장-벽 경계

y_f : 바닥-벽 경계

y_w : 벽-벽 경계

주식 보정 전처리 → 1D Layout 표현 → 특징 추출기(Feature Extractor) → 순환 신경망(Recurrent Neural Network for Capturing Global Information) → Post-processing → Pano Stretch 데이터 확장(Data Augmentation)



```

112 class ConvCompressH(nn.Module):
113     ''' Reduce feature height by factor of two '''
114     def __init__(self, in_c, out_c, ks=3):
115         super(ConvCompressH, self).__init__()
116         assert ks % 2 == 1
117         self.layers = nn.Sequential(
118             nn.Conv2d(in_c, out_c, kernel_size=ks, stride=(2, 1), padding=ks//2),
119             nn.BatchNorm2d(out_c),
120             nn.ReLU(inplace=True),
121         )
122
123     def forward(self, x):
124         return self.layers(x)

```

model.py

```

126
127 class GlobalHeightConv(nn.Module):
128     def __init__(self, in_c, out_c):
129         super(GlobalHeightConv, self).__init__()
130         self.layer = nn.Sequential(
131             ConvCompressH(in_c, in_c//2),
132             ConvCompressH(in_c//2, in_c//2),
133             ConvCompressH(in_c//2, in_c//4),
134             ConvCompressH(in_c//4, out_c),
135         )
136
137     def forward(self, x, out_w):
138         x = self.layer(x)
139
140         assert out_w % x.shape[3] == 0
141         factor = out_w // x.shape[3]
142         x = torch.cat([x[...], -1:], x, x[...], :1], 3)
143         x = F.interpolate(x, size=(x.shape[2], out_w + 2 * factor), mode='bilinear', align_corners=False)
144         x = x[...], factor:-factor]
145         return x
146
147
148 class GlobalHeightStage(nn.Module):
149     def __init__(self, c1, c2, c3, c4, out_scale=8):
150         ''' Process 4 blocks from encoder to single multiscale features '''
151         super(GlobalHeightStage, self).__init__()
152         self.cs = c1, c2, c3, c4
153         self.out_scale = out_scale
154         self.ghc_lst = nn.ModuleList([
155             GlobalHeightConv(c1, c1//out_scale),
156             GlobalHeightConv(c2, c2//out_scale),
157             GlobalHeightConv(c3, c3//out_scale),
158             GlobalHeightConv(c4, c4//out_scale),
159         ])
160
161     def forward(self, conv_list, out_w):
162         assert len(conv_list) == 4
163         bs = conv_list[0].shape[0]
164         feature = torch.cat([
165             f(x, out_w).reshape(bs, -1, out_w)
166             for f, x, out_c in zip(self.ghc_lst, conv_list, self.cs)
167         ], dim=1)
168         return feature
169

```

ResNet-50의 각 block

: 3개의 convolution Layer을 포함하고 특정 피라미드를 출력 함.

: Convolution 커널의 높이는 4, 2, 2 / 너비는 1 / 채널 수는 원래의 절반 (%2)

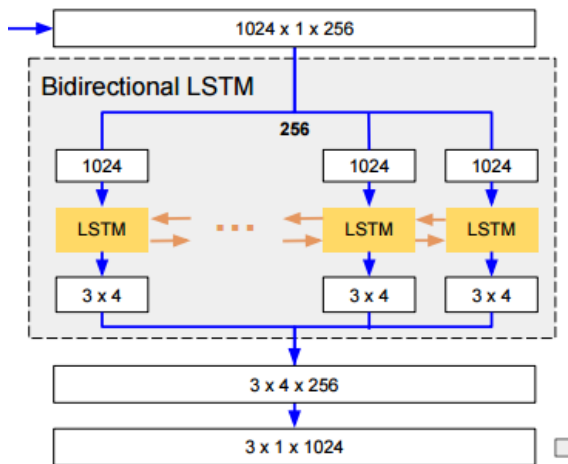
: 3개의 Convolution 후 마스터리 번호는 $1/8(2^2 \cdot 2)$, 높이는 $1/16(4^2 \cdot 2)$, 너비 = 256이 되도록 업 샘플링 함.

Reshape : 높이가 256이 되도록 모양을 변경하면 채널 수가 1이 된다.

Concat : 4ro의 기능 맵을 연결하여 크기를 만든다. $(2014 \cdot 1 \cdot 256)$

수직 보정 전처리 → 1D Layout 표현 → 특징 추출기(Feature Extractor) → 순환 신경망(Recurrent Neural Network for Capturing Global Information) → Post-processing → Pano Stretch 데이터 확장(Data Augmentation)

model.py

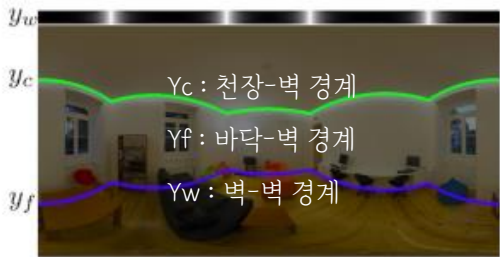


```
237 def forward(self, x):
238     if x.shape[2] != 512 or x.shape[3] != 1024:
239         raise NotImplementedError()
240     x = self._prepare_x(x)
241     conv_list = self.feature_extractor(x)
242     feature = self.reduce_height_module(conv_list, x.shape[3]//self.step_cols)
243
244     # rnn
245     if self.use_rnn:
246         feature = feature.permute(2, 0, 1) # [w, b, c*h]
247         output, hidden = self.bi_rnn(feature) # [seq_len, b, num_directions * hidden_size]
248         output = self.drop_out(output)
249         output = self.linear(output) # [seq_len, b, 3 * step_cols]
250         output = output.view(output.shape[0], output.shape[1], 3, self.step_cols) # [seq_len, b, 3, step_cols]
251         output = output.permute(1, 2, 0, 3) # [b, 3, seq_len, step_cols]
252         output = output.contiguous().view(output.shape[0], 3, -1) # [b, 3, seq_len*step_cols]
253     else:
254         feature = feature.permute(0, 2, 1) # [b, w, c*h]
255         output = self.linear(feature) # [b, w, 3 * step_cols]
256         output = output.view(output.shape[0], output.shape[1], 3, self.step_cols) # [b, w, 3, step_cols]
257         output = output.permute(0, 2, 1, 3) # [b, 3, w, step_cols]
258         output = output.contiguous().view(output.shape[0], 3, -1) # [b, 3, w*step_cols]
259
260     # output.shape => B x 3 x W
261     cor = output[:, :1] # B x 1 x W
262     bon = output[:, 1:] # B x 2 x W
263
264     return bon, cor
```

기하학적으로 말하면 방의 모든 구석은 다른 구석의 위치에서 대략적으로 유추 할 수 있으므로 RNN은 전역 정보 및 장기 종속성을 캡처하는 데 사용됩니다. LSTM의 셀 상태에있는 다른 영역의 정보를 저장합니다. RNN은 열별로 열을 예측합니다. $와이_{시}$, $와이_{오프}$, $와이_w$ 즉, RNN의 시퀀스 길이는 이미 지 너비에 비례합니다. 이 논문의 방법에서 RNN은 시간 단계 당 1 개의 열이 아닌 4 개의 열을 예측하며 정확도에 영향을주지 않고 계산 시간이 더 적게 걸립니다. 파노라마의 왼쪽과 오른쪽이 연속적이기 때문에 양방향 LSTM이 사용됩니다.

수직 보정 전처리 → 1D Layout 표현 → 특징 추출기(Feature Extractor) → 순환 신경망(Recurrent Neural Network for Capturing Global Information) → Post-processing → Pano Stretch 데이터 확장(Data Augmentation)

- 가정
- I. 교차 벽은 서로 수직이다(Manhattan world가정).
 - II. 바닥과 천장이 서로 평행한 모든 객실의 1층 1층 배치
 - III. 카메라 높이는 1.6미터
 - IV. 전처리 단계에서 바닥을 y축에 직교하도록 올바르게 정렬합니다.



misc/post_proc.py

1. 바닥 및 천장 평면 복구
- 천장-바닥 거리 투표하기 위해 yf(바닥-벽), yc(천장-벽)에서 상응하는 값을 사용할 수 있다. 가정된 카메라 높이를 기반으로 이미지에서 3D XYZ 위치로 바닥- 벽 경계인 yf를 투영할 수 있습니다(모두 동일한 Y를 공유함).
- 천장-벽 경계 yc는 동일한 3D X, Z 위치를 동일한 영상 열에 있는 yf와 공유합니다. 그러므로 바닥과 천장 사이의 거리는 계산될 수 있다.

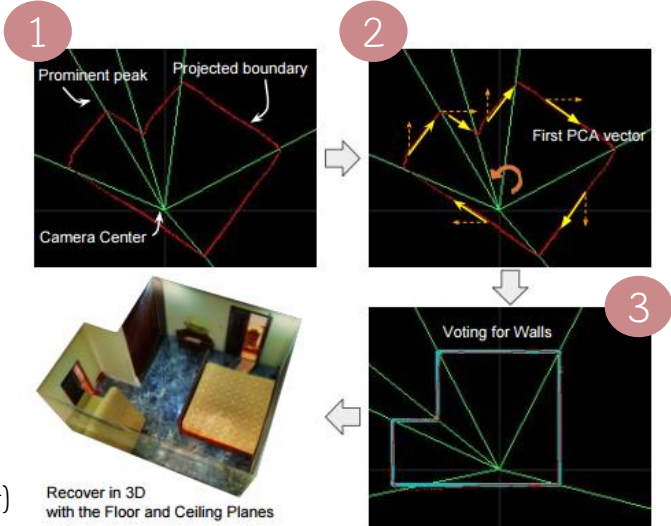
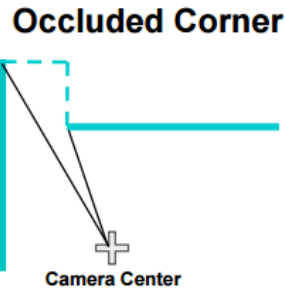
$$s_i = \frac{-1.6}{\tan(\phi_i^f)}$$
$$H_i^c = s_i \times \tan(\phi_i^c)$$
$$H = H_f + \text{mean}_i(H_i^c)$$

2. 벽면 복구
- 빨간 선 : Yc(천장-벽)
- 1 녹색 선 : 눈에 띄는 봉우리는 여러 부분으로 나눔
 - 2 각 부품에서 PCA를 수행하여 부품의 주 방향 벡터를 얻음. 각 부품의 방향 각도를 평균화하고 전체 평면을 회전함.
 - 3 즉, 벽은 XZ 축과 정렬되어야 하고 인접한 벽은 수직이며 마지막으로 벽이 선택됨

Gen_ww_Cuboid 자체에는 문제가 있음. 두 경우 모두 벽 대신 모서리를 추가함. (벽에 투표하는 대신, 두개의 두드러진 봉우리와 두 개의 벽의 위치에 따라 코너를 추가함)

```
def gen_ww_cuboid(xy, gpid, tol):
    xy_cor = []
    assert len(np.unique(gpid)) == 4

    # For each part separated by wall-wall peak, voting for a wall
    for j in range(4):
        now_x = xy[gpid == j, 0]
        now_y = xy[gpid == j, 1]
        new_x, x_score, x_l1 = vote(now_x, tol)
        new_y, y_score, y_l1 = vote(now_y, tol)
        if (x_score, -x_l1) > (y_score, -y_l1):
            xy_cor.append({'type': 0, 'val': new_x, 'score': x_score})
        else:
            xy_cor.append({'type': 1, 'val': new_y, 'score': y_score})
```



(a) Depicting how we recover the wall planes from our model output.

주식 보정 전처리 → 1D Layout 표현 → 특징 추출기(Feature Extractor) → 순환 신경망(Recurrent Neural Network for Capturing Global Information) → Post-processing → Pano Stretch 데이터 확장(Data Augmentation)

```
# Detect wall-wall peaks
if min_v is None:
    min_v = 0 if force_cuboid else 0.05
r = int(round(W * r / 2))
N = 4 if force_cuboid else None
xs_ = find_N_peaks(y_cor_, r=r, min_v=min_v, N=N)[0]

# Generate wall-walls
cor, xy_cor = post_proc.gen_ww(xs_, y_bon[0], z0, tol=abs(0.16 * z1 / 1.6), force_cuboid=force_cuboid)
if not force_cuboid:
    # Check valid (for fear self-intersection)
    xy2d = np.zeros((len(xy_cor), 2), np.float32)
    for i in range(len(xy_cor)):
        xy2d[i, xy_cor[i]['type']] = xy_cor[i]['val']
        xy2d[i, xy_cor[i-1]['type']] = xy_cor[i-1]['val']
    if not Polygon(xy2d).is_valid:
        print(
            'Fail to generate valid general layout!! '
            'Generate cuboid as fallback.',
            file=sys.stderr)
    xs_ = find_N_peaks(y_cor_, r=r, min_v=0, N=4)[0]
    cor, xy_cor = post_proc.gen_ww(xs_, y_bon[0], z0, tol=abs(0.16 * z1 / 1.6), force_cuboid=True)

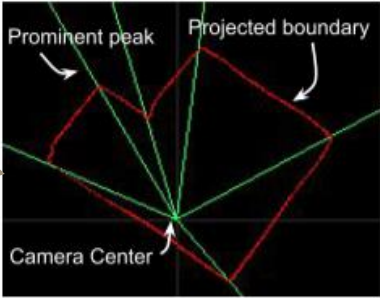
# Expand with btn coory
cor = np.hstack([cor, post_proc.infer_coory(cor[:, 1], z1 - z0, z0)[:, None]])

# Collect corner position in equirectangular
cor_id = np.zeros((len(cor)*2, 2), np.float32)
for j in range(len(cor)):
    cor_id[j*2] = cor[j, 0], cor[j, 1]
    cor_id[j*2 + 1] = cor[j, 0], cor[j, 2]

# Normalized to [0, 1]
cor_id[:, 0] /= W
cor_id[:, 1] /= H

return cor_id, z0, z1, vis_out
```

Find_N_peaks()
: prominent peak를 찾는 과정



Gen_ww()
: cuboid, non-cuboid 구분
: gen_ww_cuboid() -> vote 함수
: gen_ww_general() -> vote 함수

misc/post_proc.py

```
def gen_ww_cuboid(xy, gpid, tol):
    xy_cor = []
    assert len(np.unique(gpid)) == 4

    # For each part separated by wall-wall peak, voting for a wall
    for j in range(4):
        now_x = xy[gpid == j, 0]
        now_y = xy[gpid == j, 1]
        new_x, x_score, x_l1 = vote(now_x, tol)
        new_y, y_score, y_l1 = vote(now_y, tol)
        if (x_score, -x_l1) > (y_score, -y_l1):
            xy_cor.append({'type': 0, 'val': new_x, 'score': x_score})
        else:
            xy_cor.append({'type': 1, 'val': new_y, 'score': y_score})
```

```
def gen_ww_general(init_coory, xy, gpid, tol):
    xy_cor = []
    assert len(init_coory) == len(np.unique(gpid))

    # Candidate for each part separated by wall-wall boundary
    for j in range(len(init_coory)):
        now_x = xy[gpid == j, 0]
        now_y = xy[gpid == j, 1]
        new_x, x_score, x_l1 = vote(now_x, tol)
        new_y, y_score, y_l1 = vote(now_y, tol)
        u0 = np.coorx2u(init_coory[(j - 1 + len(init_coory)) % len(init_coory)])
        u1 = np.coorx2u(init_coory[j])
        if (x_score, -x_l1) > (y_score, -y_l1):
            xy_cor.append({'type': 0, 'val': new_x, 'score': x_score, 'action': 'ori', 'gpid': j, 'u0': u0, 'u1': u1, 'tbd': True})
        else:
            xy_cor.append({'type': 1, 'val': new_y, 'score': y_score, 'action': 'ori', 'gpid': j, 'u0': u0, 'u1': u1, 'tbd': True})
```

For문의 개수 차이
Cuboid : 4개
Non-cuboid : 모서리 개수

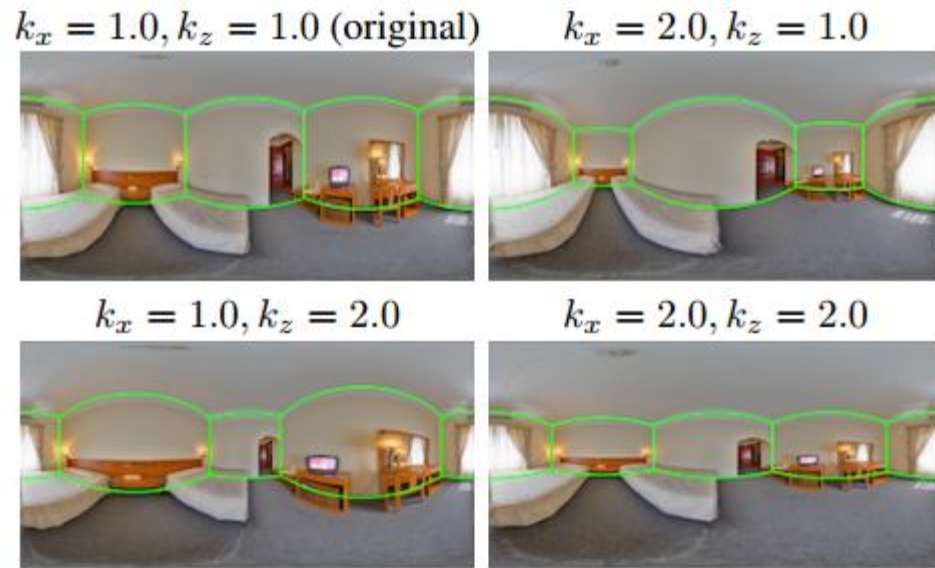
```
def gen_ww(init_coory, coory, z=50, coorW=1024, coorH=512, floorW=1024, floorH=512, tol=3, force_cuboid=True):
    gpid = get_gpid(init_coory, coorW)
    cor = np.hstack([np.arange(coorW)[:, None], coory[:, None]])
    xy = np.coor2xy(cor, z, coorW, coorH, floorW, floorH)

    # Generate wall-wall
    if force_cuboid:
        xy_cor = gen_ww_cuboid(xy, gpid, tol)
    else:
        xy_cor = gen_ww_general(init_coory, xy, gpid, tol)

    # Ceiling view to normal view
    cor = []
    for j in range(len(xy_cor)):
        next_j = (j + 1) % len(xy_cor)
        if xy_cor[j]['type'] == 1:
            cor.append((xy_cor[next_j]['val'], xy_cor[j]['val']))
        else:
            cor.append((xy_cor[j]['val'], xy_cor[next_j]['val']))
    cor = np_xy2coor(np.array(cor), z, coorW, coorH, floorW, floorH)
    cor = np.roll(cor, -2 * cor[:, 2, 0].argmin(), axis=0)

    return cor, xy_cor
```


주식 보정 전처리 → 1D Layout 표현 → 특징 추출기(Feature Extractor) → 순환 신경망(Recurrent Neural Network for Capturing Global Information) → Post-processing → Pano Stretch 데이터 확장(Data Augmentation)



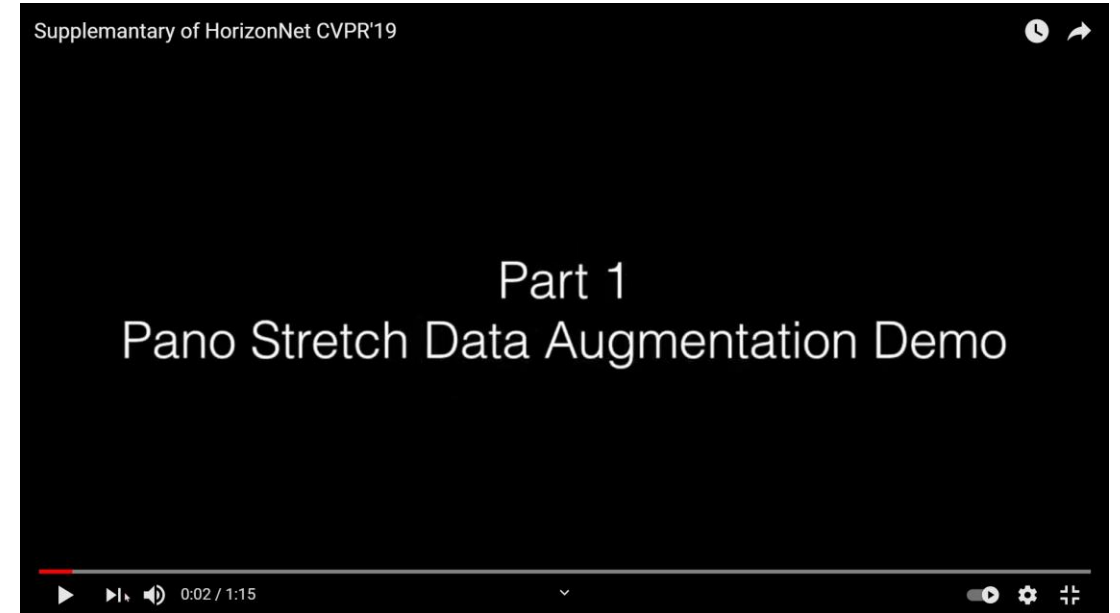
녹색선 : 이미지와 ground truth 배치

녹색 선은 x또는 z축을 따라 늘어난다.

→ 룸의 길이와 너비를 변경하여 데이터를 늘릴 수 있다..

This augmentation strategy improves our quantitative results under all experiment settings .

이 증강 전략은 모든 실험 설정에서 정량적 결과를 개선한다.(정확도 높이는데 사용)





출처

<https://www.freepik.com>

<https://www.flaticon.com/kr/>

www.flaticon.com

2021-1 Computer Vision Project

The End