Room Layout Estimation using Convolutional Neural Networking

Derek Chung
Stanford University
Department of Computer Science: CS 231A Computer Vision Final Project
dchung22@stanford.edu

Abstract

Room estimation layout involves estimating the internal structure of a room, given an RGB picture of the room. The picture of a room is from a single viewpoint, taken from the interior, and may contain objects within or other than the room's natural boundaries. The estimation our model aims to provide takes the form of a 3d cube, where we estimate the corner locations of a given room (see http://deeplayout.stanford.edu/ for examples)[1]. I will use the project of a public GitHub repository[3] to load the Large-scale Scene Understanding Challenge (LSUN) dataset[2] and the model. I will modify the relevant files to create the Fully Convolutional Neural Network Model and the optimization algorithms described in the research paper titled DeLay: Robust Spatial Layout Estimation for Cluttered Indoor Scenes[8]. I test my model on established benchmarks made specifically for the LSUN dataset.

The model includes 2 parts. One is a convolutional neural network that is described in detail later on. The other part is a greedy optimization algorithm that tries to find the relevant corner points to maximize a scoring algorithm. We output the label map of the entire room where each pixel maps to a predicted surface and the locations of the relevant corners.

1. Introduction

The Princeton 2016 Large-scale Scene Understanding Challenge asked challengers to create a model where given a single image of the interior of a room, to predict the geometric layout of the room by estimating the locations of the corners and the walls. The task is made difficult by the fact that we are given "clutter" within each image; there may be objects or furniture within each room that obscure the geometric layout in the image. In other words, we, as humans, can tell that a table is not part of the floor, wall, or ceiling, but our goal is to model a program that can intelligently overlook the clutter and accurately estimate the room layout.

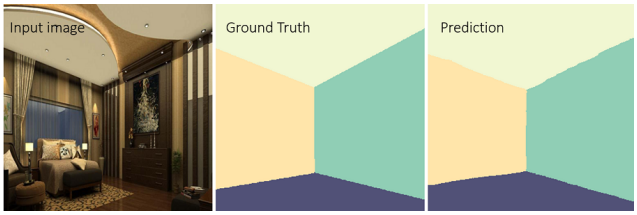

Figure 1. Basic representations of the input and output of my intended model.

There are many works that involve scene recognition, multiple camera views, and RGBD images. For my project, I will use a single RGB image in the form of a 3x320x320 matrix. I will use the image to produce a layout in the form of a 320x320 image. The layout will contain integers that correspond to a certain section of the room, namely the front wall, left wall, right wall, floor, and ceiling. See this figure for a visual representation; the different colors correspond to different surfaces of a room, and will correspond to different values in my output matrix.

The ideas that I will attempt to implement are described in a research publication[8] called DeLay: Robust Spatial Layout Estimation for Cluttered Indoor Scenes. The first stage of my solution involves recreating the model implemented in the DeLay paper. The model involves taking an RGB image as input, feeding it into a convolutional neural network, pruning the output to get initial estimates for key points, and optimizing those key points to maximize a scoring algorithm. The Stanford DeepLayout website[1] provides a nice overview of the inner workings of the algorithm. See Figure (2).

2. Background/Related Works

The DeLay model[8], the design which this paper attempts to emulate, includes a fully convolutional neural network that produced an initial room layout. Fully convolutional networks take in images of arbitrary size and produce a corresponding output. This allows my model to take in RGB images of varying size, as long as it is an RGB

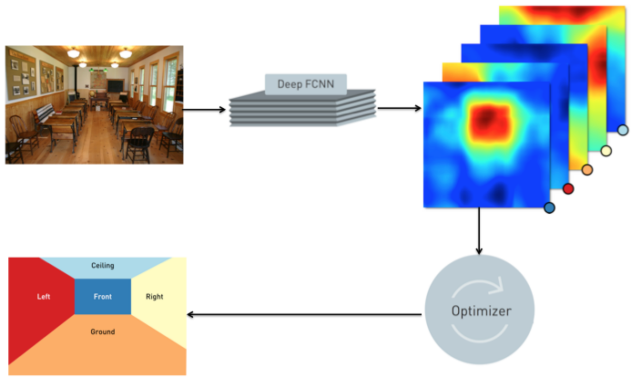

Figure 2. A more detailed diagram of the inner workings of the model. Note that the neural network is only part of the model. Picture taken from deeplayout.stanford.edu

image. For example, one could experiment with how the model's prediction changes given high and low resolutions of the same RGB image. DeLay also extracts line segments $l1, l2, l3, l4$ and vanishing point v shown in Figure (3) and runs an optimization algorithm to find the lines minimizing a scoring function. My model has the same structure, but points $p1, p2, p3, p4$ are maximized using a slightly different algorithm.

The model proposed in DeLay[8] was introduced by Varsha Hedau in the paper "Recovering Spatial Layout of Cluttered Rooms"[6]. Hedau proposes a solution where 3 orthogonal vanishing points are estimated, which are used to guess the layout of the room. Given that the RGB image is taken from the inside of the room, we are guaranteed a layout similar to the interior of a cube, with at least one wall guaranteed to be in the image. Using these constraints,there is a finite number of possible "layouts", with each layout containing geometric constraints that define which walls/edges/corner are visible in the image.

Another paper titled "Indoor Scene Layout Estimation from a Single Image"[7] by Hung Jin Lin, Sheng-Wei Huang, Shang-Hong Lai, and Chen-Kuo Chiang proposes a similar method described in the DeLay paper. One of the main differences is that the neural network is pretrained using the ResNet101 network[5]. The ResNet101 framework is a neural network pretrained on the ImageNet dataset. The framework is ultimately used for image classification, but the residual learning framework may be helpful for some models that rely on scene based algorithms that attempt to classify clutter using image recognition. In contrast, the Indoor Scene Estimation Layout paper takes the output of the neural network and applies optimization algorithms to generate lines, which then produces a final output where all edges are smoothed out.

3. Approach

Some of the formulas below are taken from the DeLay model and the paper associated with it. [8].

3.1. Overview

Given an RGB image $I \in \mathrm{R}^{WxH}$, I produce an output $L \in \mathrm{R}^{WxHx5}$ where

$$L_{ij} \in \{Left, Front, Right, Ceiling, Ground\} \quad (1)$$

and

$$L_{ij} = \hat{I_{ij}} \quad (2)$$

, where $\hat{I_{ij}}$ is our model's prediction for what section of the room pixel I_{ij} resides in. I assume the room's adjacent surfaces are orthogonal, and that opposite surfaces are parallel. In my model, an integer of 0 represents the front wall, 1 is the left wall, 2 is the right wall, 3 is the floor, and 4 is the ceiling.

To calculate L, I feed I into a Fully Convolutional Neural Network with 21 layers that produces an output $T \in \mathrm{R}^{WxHx5}$, where after we normalize,

$$T_{ij}^{k} = P(L_{ij} = k|I) \; k \in \{0, 1, 2, 3, 4\} \quad (3)$$

k also corresponds to the various locations that a pixel could be in.

After we get T, we get an initial estimate given by the equation below:

$$\hat{L}_{ij} = argmax_k T_{ij}^{k} \quad (4)$$

The initial estimate gives us a starting layout map that map each pixel in our image to a label described in equations 1 and 2. This initial estimate, however, performs very poorly, as there are no geometric or labeling constraints. As a result, $L*$ turns out to be, in most cases, an ambiguous cluster where we can't easily draw borders to divide each label. The locations of surfaces are hard to tell as well, due to there being several disjointed components in many cases.

We end up pruning all, but the largest connected components. We use a k-nearest-neighbors algorithm to reassign the rest of the components to a better label. We solve the problem of disjointed components this way.

In addition, I calculated initial estimates of $p1, p2, p3, p4$, and v points delineated in Figure (3). I attempt to optimize the locations of $p1, p2, p3, p4$, and v that maximize the following equation:

$$S(L = f(\tau)|T) = \frac{1}{WH} \sum_{i,j} T_{i,j}^{(L_{ij})} \quad (5)$$

where $\tau = (p1, p2, p3.p4, v)$, and f is a function that maps τ to a wxh matrix L resembling a layout estimation. Having this optimization function enforces the smoothness of the output.

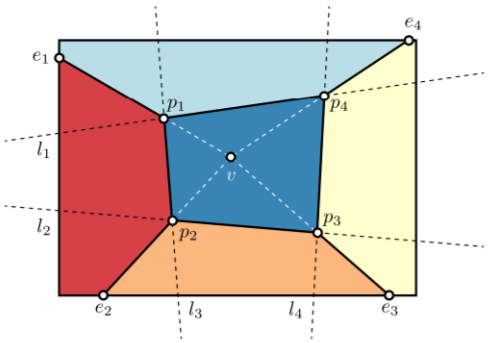

Figure 3. Layout with lines l_i and a vanishing point v. We try to optimize values of l_i and v using S.

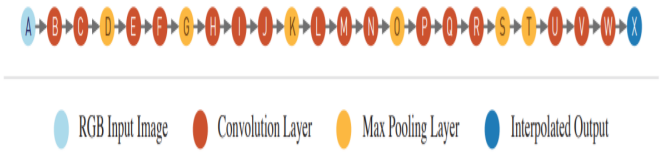

Figure 4. The fully convolutional neural network used in the first part of my model. Each Convolutional layer is followed by a rectified linear unit (ReLU), and I established the Interpolated output as a linear layer. The 2nd, 3rd, and 4th maxpool layers have a kernel space of 2 to make up for smaller features in the convolutional networks.

3.2. Manhattan Assumption

I assume the Manhattan World Assumption described in more detail in a paper[4] titled "The Manhattan world assumption: regularities in scene statistics which enable Bayesian inference". In the context of my project, I assume that all surfaces within the input rooms are orthogonal. The assumption allows me to model the room as shown in Figure (3).

3.3. Fully Convolutional Neural Network

The first part of my model involves feeding an RGB image into a convolutional neural network further described in the DeLay paper[8] and Figure (4).

The model outputs T, a $WxHx5$ matrix, which is the same output produced in equation (3).

3.4. Finetuning and Constraints

Once I have T, the output from the neural network, we extract $\hat{L}$ in equation (4). The first 2 dimensions of T represent pixels in the RGB image, and the last dimension of T represents a probability distribution of a pixel being mapped to a certain label. A label is in $\{0, 1, 2, 3, 4\}$, where each integer represents a particular surface that can exist within a room. I assume a given image contains at most a front wall (the wall directly opposite to the camera), a left wall, a right wall, the floor, and the ceiling. Some of these surfaces may be missing in T. My model accounts for this when estimating points $p1, p2, p3, p4$, and v in Figure (3).

Then, I prune all components in $\hat{L}$ except for the largest connected component for each label. I use a breath-first search algorithm to get all the connected components for each label. After selecting the largest component for each label, I use a k-nearest-neighbors algorithm to independently reassign the other components to a new label. I now have a matrix

$$L* = g(\hat{L}) \tag{6}$$

where g is a function that prunes and reassigns components in the way described above.

We are left with $L*$, a matrix that has at most one connected component per label, but doesn't necessarily have the smooth edges in Figure (3) that I would like. I enforce smoothness by generating a variable

$$\tau = (p1, p2, p3, p4, v) \tag{7}$$

corresponding to the labels in Figure (3). I provide initial estimates for $(p1, p2, p3, p4)$ by starting from the center of $\hat{L}$ and traversing diagonally in different directions until I hit a point not labeled as being part of the front wall. I employ the method in part because I enforce a constraint that $\hat{L}$ must have a component representing the front wall, or the dark blue section in Figure (3). I initialize v as the midpoint of $(p1, p2, p3, p4)$.

With this information, I can easily create a function f for equation (5) such that

$$f : A \longrightarrow B, A \in (p1, p2, p3, p4), B \in \mathrm{R}^{WxH} \tag{8}$$

f maps my list of points to a matrix resembling a layout estimation.

3.5. Optimization

I propose the modified cordinent ascent algorithm inspired by a similar one in DeLay[8]

```
Initialize (p1, p2, p3, p4, v)

For 3 iterations or until
score didn't improve:
    for all points:
```

```
shift the point
(3 - iter_num) * 4
units left/down/up/right

#the first iteration
is number 0

evaluate
Score(p1, p2, p3, p4, v | T)

if the score improved:
    set the corresponding
    point equal to
    the shifted one
```

I continuously shift the points in the layout until the score doesn't improve, or after 3 iterations. The (3 - iteration number) * 4 factor shifts the points more in early iterations, and less so in later iterations, further refining the algorithm the closer the coordinates approach their optimal values. T is the $WxHx5$ matrix outputted in Equation (3).

I chose only to optimize for a max of 3 iterations due to performance constraints and/or unoptimal code. In other words, we maximize Equation (5) using our above algorithm, which tries to find optimal values for τ. Once my algorithm gets an optimal value of τ, I return the corresponding labeling matrix $f(\tau)$ for training and evaluation.

4. Experiment

4.1. Overview

I train my neural network on the Large-scale Scene Understanding Challenge (LSUN) Dataset[2] for 30 epochs and a batch size of 8. The dataset contains 4000 training examples and 394 validation examples that I will use throughout training. My optimization algorithm will attempt to find the set of parameters that maximize the score. The scoring function is given by Equation (5).

I then test my data on the dataset published by Hedau[6], containing 105 testing images.

I compare my results to the existing model provided in a public GitHub Repository[3], which is the same model described in "Indoor scene layout estimation from a single image"[7], which I refer to as the "baseline" model in the following sections. The baseline model is a ResNet101 model pretrained on the ImageNet database, and does not include the optimization algorithm outlined in Section 3.5.

4.2. Benchmarks

The built-in framework outputs 4 different scoring mechanisms for testing, but I will focus on the number labeled "score", which is the averaged maximum bipartite labeling between my prediction and the ground truth. Consider a predicted layout matrix P and the ground truth matrix T. Both P and T are dimensions (w, h) and contain elements from the set $k \in \{0, 1, 2, 3, 4\}$, where each integer corresponds to a specific surface.

Consider a matrix $C \in \mathrm{R}^{mxn}$, where m is the number of unique labels in P and n is the number of unique labels in T. I calculate C such that

$$C_{ij} = -SUM(MASK(P, M[i] \ \& \ MASK(T, N[i]) \tag{9}$$

where $SUM(x)$ is the total sum of elements in x, M and N are matrices with dimensions m and n respectively that store the values of the unique labels in P and T respectively, and $MASK(X, y) = Z$ such that $Z_{ij} = 1$ if $X_{ij} = y$, 0 otherwise. The $\&$ is the bitwise operator.

In other words, what Equation (9) does is take every combination of unique labels from P and T in the form of $(M[a], N[b])$. If we denote $P*$ and $T*$ as the set of all points where $P_{ij} = M[a]$ and $T_{ij} = N[b]$ respectively, C_{ab} is the negative sum of the set intersetion between $P*$ and $T*$. Labels that have more common matrix indices will have a more negative value in C.

Next, I find the linear sum problem, or $\min \sum_i \sum_j C_{ij}$ such that each row i corresponds to at most 1 column j, and vice versa. This formula is the same as Minimum Weight Matching in Bipartite Graphs.

I get the cost of the assignment from the linear sum problem. The positive normalized cost represents the percentage of elements in P and T that have the same label if I reassign labels in P or T using the mapping produced by the linear sum problem. Note that a score of 1.0 means that the predicted layout matches the ground truth layout exactly in terms of the geometric layout outlined in Figure (3), but may have different label values for the same area. The higher the score, the better the prediction.

Therefore I can model the score of a particular model as

$$Score(Model) = \frac{1}{n} \sum_n F(Model, a_n) \tag{10}$$

where F produces the positive normalized linear sum of C using a prediction matrix P generated from model Model and testing data a_n, and T as the ground truth layout of a_n.

4.3. Methodology

I use Google's Collab to develop and run the baseline model and my model. A link to the page is included at supplementary material at the bottom of the paper. I use a Jupyter Notebook to download requirements and import necessary modules. I run the models using a GPU runtime on Colab to utilize PyTorch tensor.

4.4. Quantitative results

The baseline model produced a score of 0.54773 testing on the dataset published by Hedau[6]. The output is provided in Figure (5).

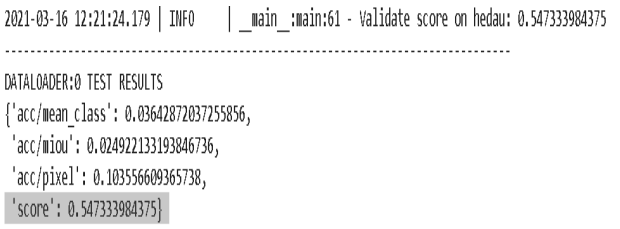

Figure 5. The validation scores after training the baseline model.

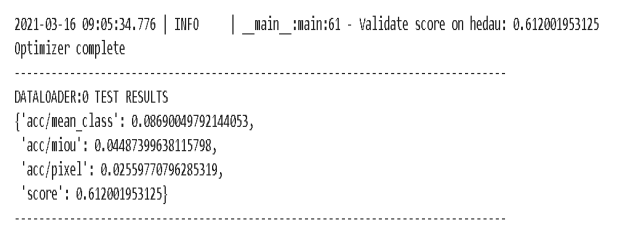

Figure 6. The validation scores after training my customized DeLay model.

My model produces a score of 0.612, which is approximately a 7 percent increase from the baseline model. More details in Figure (6),

4.5. Qualitative results

The runtime of my model was significantly longer that the baseline model, with testing taking an average of 80 seconds per iteration. In contrast, the baseline model took 2-3 seconds on average per iteration. I can attribute the drop in efficiency to the optimization of code described in sections 3.4 and 3.5. Significant improvements should be possible to improve the speed of getting the score of a layout $(p1, p2, p3, p4, v)$.

Even though the optimization algorithm can run a maximum of 3 iterations, the overwhelming majority of testing examples ran 2 iterations, which means there was no score improvement on iteration 2. Given the algorithm shifts points 12 units in the first iteration and 8 units in the second, I make the assumption that the optimal values pf $(p1, p2, p3, p4, v)$ are contained within a 40 by 40 box surrounding the initial estimates of each point. Given that my image size is 320x320, there are 64 40x40 squares contained in the image. In optimizing the layout $(p1, p2, p3, p4, v)$, I can add constraints to evaluate scores only if all points remain within the bounding box.

5. Conclusion

Estimating the layout of a room given an RGB image seems like a reasonable task to the human eye, but when clutter is introduced, there are several approaches that a computer can take to perform the same task. Some methods include using image classification techniques to identify clutter, optimizing parameters from convolutional neural networks to enforce geometric constraints, and using vanishing point estimates to get a predefined layout of a room. I combined developments in Fully Convolutional Neural Networks and principles of coordinate ascent to get a collection of 5 key points defining a room layout. While my optimization wasn't ideal, my benchmarks were better than I expected, given the runtime constraints on Google Collab.

The coordinate ascent optimization algorithm described in Section 3.5 is unlikely to get the exact set of points that maximize the scoring function. If I knew how to optimize the code, then my model's score would increase, as I could increase or remove constraints on the number of iterations presented in the optimization algorithm. I can also decrease the increment that I use to shift each point to get more fine-tuned and accurate results. Finally, I can utilize multiple cores and vectorize the optimization algorithm to simultaneously run the algorithm on all batches. Currently, I optimize one batch at a time, and I structured my code so that another batch cannot start until the previous one finishes.

It is also necessary for me to understand the different types of errors shown and integrate the standard benchmarks into my testing models. For example, pixel accuracy and corner accuracy were included in the output, but I have yet to comprehend the mathematics behind the benchmarks, and didn't include them as a result. Many of the related works on room layout estimation include these two metrics when comparing results to other models, so to increase the relevance of these results, there is a need to take into account corner and pixel accuracy in my analysis.

6. Supplementary Material

My Google Collab project: `https://drive.google.com/drive/folders/17z50KCQS6bFHMjgMhnmTMsMZLOa3lEyY?usp=sharing`

References

[1] Deep layout estimation. `http://deeplayout.stanford.edu/`. Website of project this paper is based off.

[2] Largescale scene understanding challenge: Room layout estimation.

[3] levirve lsun github repository. `https://github.com/leVirve/lsun-room`.

[4] J. M. Coughlan and A. L. Yuille. The manhattan world assumption: Regularities in scene statistics which enable bayesian inference. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 809–815, Cambridge, MA, USA, 2000. MIT Press.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[6] V. Hedau, D. Hoiem, and D. Forsyth. Recovering the spatial layout of cluttered rooms. pages 1849–1856, 2009.

[7] H. J. Lin, S. Huang, S. Lai, and C. Chiang. Indoor scene layout estimation from a single image. pages 842–847, 2018.

[8] K. C. S. S. Saumitro Dasgupta, Kuan Fang. Delay: Robust spatial layout estimation for cluttered indoor scenes. Recommended by course advisor for final project ideas.