The Pennsylvania State University
The Graduate School

AUTOMATIC GENERATION OF FLOORPLANS USING GENERATIVE ADVERSARIAL NETWORKS

A Thesis in
Computer Science and Engineering
by
Sneha Suhitha Galiveeti

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2021

The thesis of Sneha Suhitha Galiveeti was reviewed and approved by the following:

Daniel Kifer
Professor of Computer Science and Engineering
Thesis Co-Advisor

Sharon Xiaolei Huang
Professor of Information Sciences and Technology
Thesis Co-Advisor

C. Lee Giles
David Reese Professor of Information Sciences and Technology

Chitaranjan Das
Distinguished Professor of Computer Science and Engineering
Department Head of Computer Science and Engineering

Abstract

In the present day, demand for construction of houses is increasing rapidly. But creating and designing a floorplan requires a lot of creativity, technical knowledge and mathematical skills. The number of architects with these skills are not adequate to meet the requirements of the rapidly growing demand. We can use Machine Learning to solve this problem of floorplan generation. This project explores the idea of generation of multiple floorplans using deep learning models especially Generative Adversarial Networks(GANs).

This work concentrates on the generation of rasterized floorplans. The main approach is to let GAN treat floorplans as raster images and learn their distribution to produce new floorplans. This work explored multiple models of GANs like simple DCGAN, LSGAN, WGAN, StyleGAN etc. and studied the pros and cons of each model over two major datasets Structure3D and Graph2Plan. This work also explored the conditional generation of floorplans i.e., controlling the layout of generated floorplans by giving input condition to the models in terms of types of rooms present.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisors Dr. Sharon Huang and Dr. Zihan Zhou for guiding and understanding me throughout the project. I would also like to thank Yuan Xue for clearing my doubts and for continued support throughout. I would like to thank Professor Daniel Kifer for accepting my request and advising me on my thesis. I would also like to thank Dr. C. Lee Giles for being a part of my thesis committee.

I am forever grateful and thankful to my parents Dr. Galiveeti Sreenivasulu Reddy and Galiveeti Sreevani for believing in me and supporting me throughout my graduate studies. I would also like to thank all my family and friends who inspired me and made my graduate journey memorable.

Chapter 1
Introduction

1.1 Motivation

Every adult dreams of having a house designed for their specific needs and liking. But designing a house is a highly skilled process. The basic design of a house is a floorplan. Designing a floorplan with specific needs and requirements in the available plot of land needs creativity along with mathematical skills. Architects who are trained in these fields can create a floorplan of customer's liking only after an iterative process of multiple discussions and re-designing the floorplan. We can now get an idea of the time it takes for a floorplan to develop. But, do we have enough architects to design all the houses in the world? The answer is a big No!, there are far too less architects than the ever growing demand of house construction. According to an architect Duo Dickinson, *"Only 2% of all houses are designed by architects."* [**?**] This can be of multiple reasons apart from less number architects such as, not all can afford to have an architect design their homes and sometimes we only need basic floorplan and nothing fancy. With all these discussed, it is very much useful to have a computer or an algorithm which can generate the floorplans which can be used by anyone for any purpose. Also, this problem of creating floorplans is a fun problem because we are trying to use algorithms to solve a problem which needs creativity!!

1.2 Goal

The main goal of this thesis is to generate multiple floorplans for a given input using Generative Adversarial Networks and help the architects with design process. This thesis also compares the GAN models used to generate the floorplans.

Chapter 2
Related work

2.1 Graph2Plan

Ruizhen Hu et al. [**?**] designed a framework called Graph2Plan to automate the generation of floorplans. This uses deep neural networks and user given constraints to develop the floorplans. The user can give the outer boundary and the constraints such as number of bedrooms, kitchen etc., as input to the model. This model then fetches multiple layout graphs from the database which is closer to the constraints and tries to fit these layout graphs into the outer boundary and generates multiple floorplans. The drawback to this work is that the constraints given can only match with the predefined stored layout graphs and therefore new constraints when given, it can only fetch layout graphs which are closer to the given the constraints.

2.2 HouseGAN

HouseGAN is a graph-constrained house layout generator developed by N. Nauata et al. [**?**]. They have built a relative generator and discriminator architecture to enforce graph constraints. The input to the HouseGAN architecture is a constraint graph which specifies the number of room types, number of rooms of each type and the spatial adjacency. The model then outputs a set of axis aligned bounding boxes for each node in the graph.

Chapter 3
Generative Adversarial Networks

3.1 What are GANs?

Generative Adversarial Networks(GANs) are a class of Machiche Learning architectures designed by Ian Goodfellow et al. [**?**]. GANs are made up of two networks called as Generator(G) and Discriminator(D). The generator network learns the data distribution and creates new/fake data while the discriminator classifies the data as real or fake. The goal of GANs is to generate new/fake data as close to the real data by minimizing the difference between data distribution and model distribution.

3.2 How does GAN work?

The GAN works in an adversarial process, Figure3.1 shows the total process of training a GAN. Both Generator(G) and Discriminator(D) are trained simultaneously in a competitive environment. G learns to produce new data to fool the discriminator into believing that the generated data is real while D learns to classify the generated data as fake. In other words, G and D play a minimax game where D tries to maximize the probability and G tries to minimize the probability.

In one iteration of GAN training, Generator gets noise(z) as input and produces fake data. This fake data along with real data is fed to Discriminator. After the classification, the loss is backpropogated to discriminator to better classify real and fake data. Generator then gets updated on how well or not the discriminator was fooled. Generator basically gets trained on how to fool the discriminator which results in learning the real data distribution.

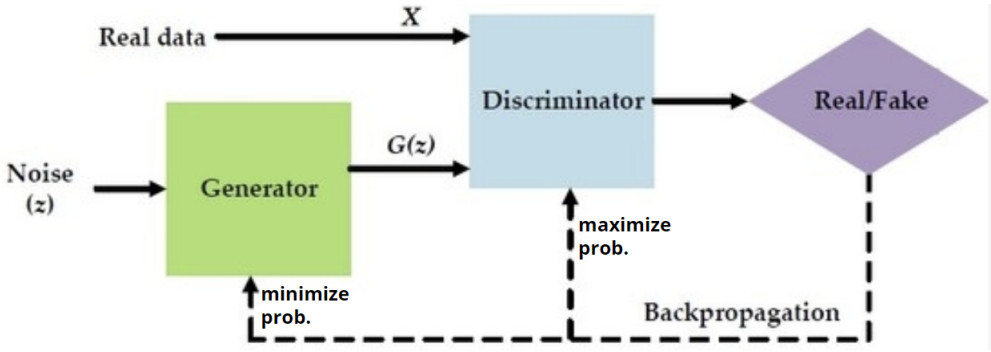

Figure 3.1. General GAN model [**?**]

Looking at the training process we can say that GANs are unsupervised learning models which use supervised loss in the training. The discriminator in the GAN accounts for supervised component since it classifies real/fake data. This knowledge is in turn used by the generator to do unsupervised learning of real data distribution.

Chapter 4
Datasets

This chapter talks briefly about the two original datasets used to train the GAN models specified in this thesis. Data reprocessing performed on the two datasets will be discussed in the following chapters which is specific to the GAN model in discussion.

4.1 Structured3D

Structured3D is a large-scale photo-realistic dataset developed by Jia Zheng et al. [**?**]. This dataset provides rich ground truth 3D annotations to 3,500 scenes. This dataset has been chosen since it produced better results [**?**] in room layout estimation compared to other datsets such as SUNGG [**?**] and SceneNet RGB-D [**?**].

4.1.1 Representation

All the 3D scenes in this dataset are represented using Primitives and Relationships. Each scene is modelled using it's geometric primitives such as Points, Lines and Planes. Relationships such as Plane-Line, Line-Point, etc., are generated between the primitives to represent a complete 3D scene. A .json file is generated for each 3D scene capturing all the details.

Using the visualization tool provided by Jia Zheng et al. [**?**], I have developed the floorplans for each 3D scene and used these raster images to train the GAN models. There are 19 room types as per the annotations of the Structured3D dataset and each each room type is represented with a specific color across all the floorplans. Figure4.1 shows the room types and their respective colors. Also, Figure 4.2 shows few sample floorplans generated using the visualization tool.

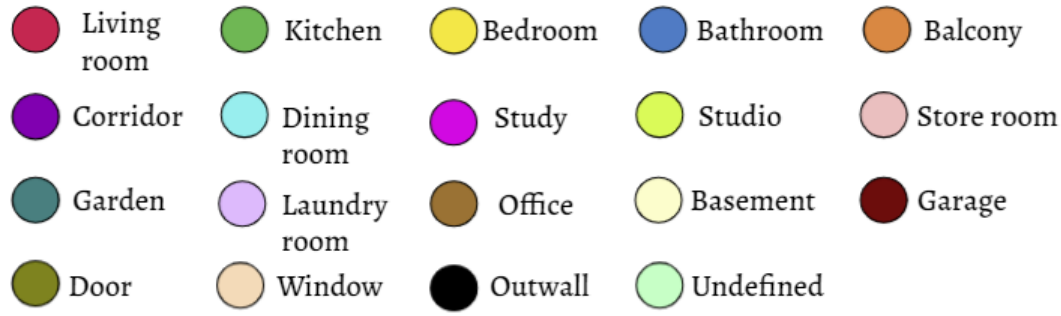

Figure 4.1. Room type and their respective colors [**?**]

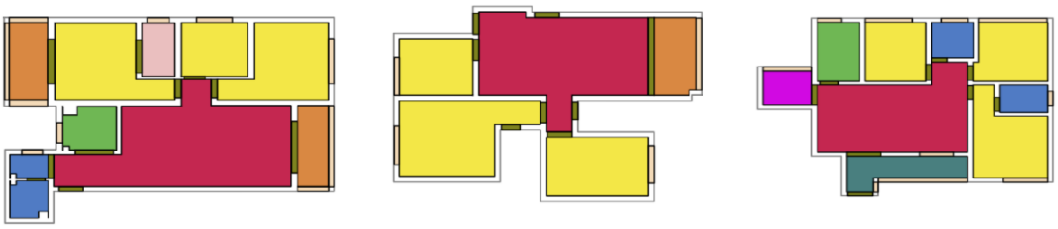

Figure 4.2. Few sample floorplans generated from Structured3D dataset [**?**]

4.2 Graph2Plan

Graph2Plan dataset is generated by Ruizhen Hu et al. [**?**], from their Graph2Plan network which is trained on RPLAN [**?**] dataset. This dataset has 67,453 floorplans represented as raster images which were used to trained few GAN models specified in this thesis.

4.2.1 Representation

Since this dataset is a collection of raster images, they are stored as .png files. There are only six types of rooms present in the floorplans of this dataset. They are Public area/Living room, Bedroom, Bathroom, Balcony, Kitchen and Storage. Each room type has a specific color across the dataset and Figure 4.3 shows the colors. Also, Figure 4.4 shows few sample floorplans taken from Graph2Plan dataset.

Figure 4.3. Room type and their respective colors [**?**]

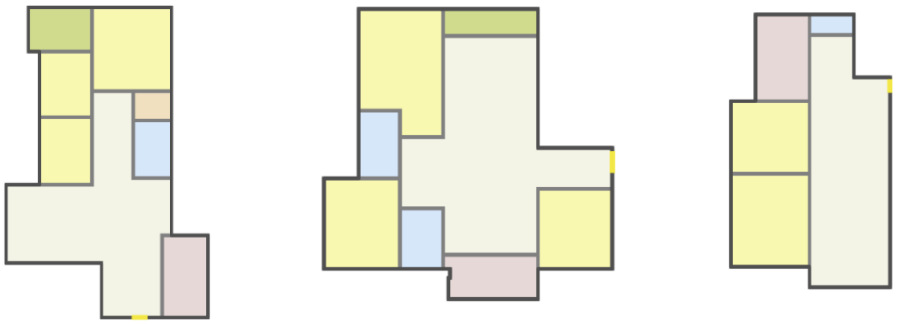

Figure 4.4. Few sample floorplans from Graph2Plan dataset [**?**]

Chapter 5
Unconditional floorplan Generation

This chapter talks about the unconditional generation of floorplans using GANs. In this process of floorplan generation, only the rasterized floorplan images are used to train the GANs without any other conditional inputs. As we have seen in Chapter 3, noise will be given as input to Generator(G) to produce fake floorplans. Then these floorplans along with the real floorplans are given as input to the Discriminator(D) to classify them as real/fake. Then the loss is back- propogated and the GAN model is trained.

When talking about a GAN model, few important parameters to be discussed are the architecture of Generator and Discriminator, inputs to Generator and Discriminator, Loss function used and the hyper parameters which can be tuned. When talking about training a GAN, we need to discuss about the dataset used, the pre-processing of the dataset, results generated and the post processing. I will now discuss in detail about each GAN model in terms of all the parameters mentioned above.

5.1 DCGAN

DCGAN stands for Deep Convolutional Generative Adversarial Network. DCGAN is a very simple GAN model which has deep convolutional neural networks in the architectures of both Generator and Discriminator. Therefore the model of DCGAN is very similar to the model of GAN discussed in Chapter 3.

5.1.1 Raster Images

In the very first attempt to train the GAN to generate floorplans, raster images were directly used.

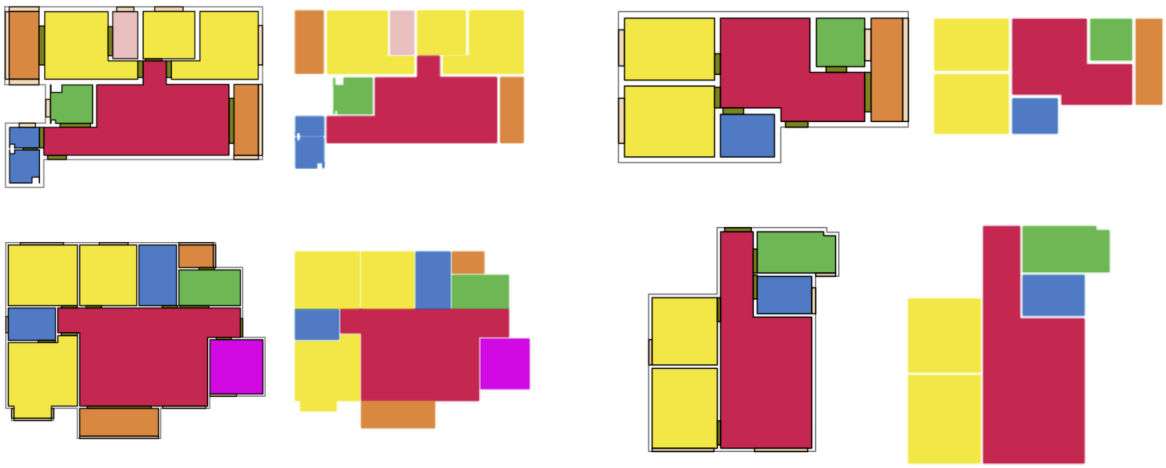

Figure 5.1. Structured3D - Removing doors, windows and outwall

5.1.1.1 Data Preprocessing

Sturctured3D dataset was used to train this model. Following are the preprocessing steps performed on the dataset:

1. Room types such as doors, windows and outer walls were removed from all floorplans. Figure 5.1 shows few examples of this conversion.
2. Resized the floorplans to a resolution of 256 x 256 x 3
3. Augmented each floorplan by rotating 90 degrees thereby creating a dataset of 7,000 images

5.1.1.2 Architecture

Generator: The input layer of the generator network takes in a uniform random noise of dimension 100. This layer is followed by four upsample and convolutional layers with batch normalization and Relu activation function. Finally, a convolutional layer with depth 3 is added along with a tanh activation function to generate images of size 256 x 256 x 3.

Discriminator: The input layer of the discriminator network takes in images of size 256 x 256 x 3. This input layer is followed by 4 convolutional layers, each with a LeakyRelu activation function and a dropout. Finally a sigmoid activation is used to classify the images as real(1) or fake(0).

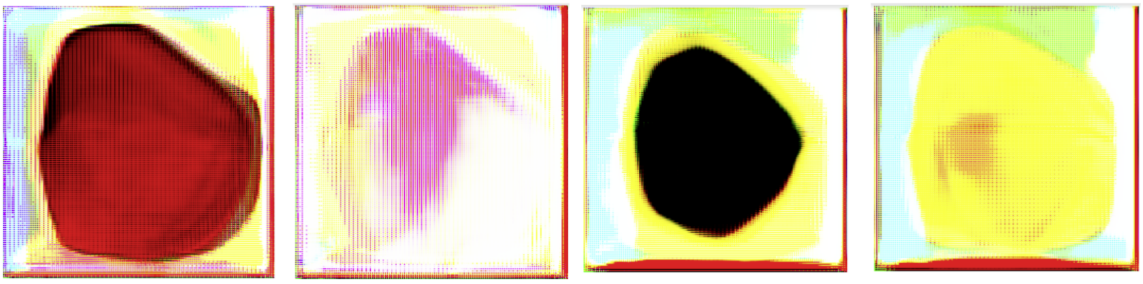

Figure 5.2. Generated floorplans for DCGAN - raster images

Loss Function: Binary cross-entropy was used as the loss function in DCGAN. This model uses Adam optimizer with a learning rate of 0.0001.

5.1.1.3 Results

This model was trained with 7k images for 200 epochs with a batch size of 32. Figure 5.2 shows few generated floorplans after training the model for 200 epochs.

5.1.1.4 Observations

Looking at the generated results, we can see that there is a color mixing which resulted in new colors other than the ones that were in the dataset. To eliminate this, a segmentation map was used, which is discussed in the next section.

5.1.2 Segmentation Map

In this attempt, segmentation maps were used to train the DCGAN instead of directly using the RGB images.

5.1.2.1 Data Preprocessing

Additional data preprocessing steps were added to the previous steps:

1. Floorplans were filtered. Floorplans which have at least 1 living room, 1 bedroom and 1 bathroom were considered and the others were eliminated. This reduced the dataset size to 4,238 images

2. These floorplans were then converted to segmentation maps. Each room type was represented as 1 label. Since there are 16 room types now and with the background

white color, the dataset totally has 17 colors. When each color is mapped to one label ranging from 0 to 16, each floorplan image with 256 x 256 x 3 dimensions is mapped to 256 x 256 segmentation map with 17 labels

3. After generating the segmentation maps, every image is represented in one-hot encoding thereby changing the image dimension to 256 x 256 x 17

5.1.2.2 Architecture

Generator: The starting architecture remains the same as the generator network in DCGAN with raster images. The only difference is in the final convolutional layer and activation function. Thconvolutional layer's depth is changed to 17 from 3 and the tanh activation function is changed to sofmax function.

Discriminator: The input layer to the discriminator network is of size 256 x 256 x 17. The rest of the architecture remains same as the network in DCGAN with raster images.

Loss Function: Binary cross-entropy was used as the loss function in DCGAN.
This model also uses Adam optimizer with a learning rate of 0.0001.

5.1.2.3 Data Postprocessing

Since the generated images from the generator network are not RGB images, post processing needs to be performed. Following are the steps to convert the fake images of size 256 x 256 x 17 to RGB images:

1. Convert the image from 256 x 256 x 17 to 256 x 256 by using an argmax function

2. Reverse map the label at each position of 256 x 256 matrix to the original pixel value. This step increases the dimension and adds a depth of 3

3. Save these images as raster images

5.1.2.4 Results

This model was trained with 4,238 images for 200 epochs with a batch size of 32. Figure 5.3 shows few generated floorplans after training the model for 200 epochs.

Figure 5.3. Generated floorplans for DCGAN - Segmentation Map

5.1.2.5 Observations

Clearly, the generated results show that the problem of color mixing is eliminated with segmentation maps and the original colors are retained. We can also see that the room shapes are better than the previous results. To further improve the results, a change of loss function is implemented in the next section.

5.2 LSGAN

LSGAN stands for Least Squares Generative Adversarial Network. This LSGAN is very similar to DCGAN except in the loss function.

5.2.1 Data Preprocessing

Structured3D dataset was used to train this model and the pre processing is the same as the one for DCAGN with segmentation map.

5.2.2 Architecture

The architecture for LSGAN is same as the DCGAN with segmentation map expect for Discriminator network which has the final activation layer as the linear activation instead of sigmoid activation.

Loss Function: Mean squared error or the L2 loss is the loss function for LSGAN model. This model also uses Adam optimizer with a learning rate of 0.0001.

5.2.3 Data Postprocessing

Postprocessing for this model is same as that of DCGAN with segmentation map. Since, the dataset is also same reverse mapping of labels to pixel values is also same.

Figure 5.4. Generated floorplans for LSGAN

5.2.4 Results

This model was also trained with 4,238 images for 200 epochs with a batch size of 32. Figure 5.4 shows few generated floorplans after training the model for 200 epochs.

5.2.5 Observations

The generated floorplans show no better improvement than the previous results. Also, we can see that only two colors corresponding to two room types are always dominant. This can be due to faster convergence of discriminator. To avoid this, a change of discriminator's architecture and training is required. Next section talks about the changes.

5.3 WGAN

WGAN stands for Wasserstein Generative Adversarial Network and was developed by M.Arjovsky et al. [**?**]. It differs from the other two GANs in Discriminator and it's loss function. The Discriminator called as Critic in WGAN computes the Earth Mover(EM) distance/Wasserstein Metric and thereby scores the realness/fakeness of the input image instead of just classifying it. According to the M.Arjovsky et al. [**?**], critic can be trained till optimality and a loss function called Wasserstein loss is designed to minimize the EM distance leading to better image generation.

This model was trained with both Structred3D and Graph2Plan datasets. Both these attempts are discussed in detail as follows:

5.3.1 Structured3D

5.3.1.1 Data Preprocessing

The pre processing is the same as the one for DCGAN with segmentation map, thus producing segmaps of size 256 x 256 with 17 labels.

5.3.1.2 Architecture

Generator: The generator architecture is same as that of DCGAN with segmentation map.

Critic: The input layer to the critic takes data of size 256 x 256 x 17. This is followed by 4 convolutional layers, each with a LeakyRelu activation and dropout similar of DCGAN discrminator. Finally, linear activation is used to produce the scores.

Loss function: Wasserstein Loss as discussed in the work by M.Arjovsky et al. [**?**] is used. Weights are clipped in WGAN to enforce constraints in calculating EM [**?**]. So, a clipping constraint is introduced with value 0.01.
This model also uses Adam optimizer with a learning rate of 0.0001.

5.3.1.3 Data Postprocessing

The post processing is also same as the one for DCGAN with segmentation map.

5.3.1.4 Results

This model was also trained with 4,238 images for 200 epochs with a batch size of 32. In this model, the critic was trained 5 times for every 1 training of the generator. Figure 5.5 shows few generated floorplans after training the model for 200 epochs.

5.3.1.5 Observations

The floorplans generated show a lot of improvement over the previous results in terms of straight lines. This model has produced rooms with better geometric shapes, however, the results also tell that the model has run into a model collapse issue and the generator is generating very few types of floorplans without variations. To avoid this issue, Graph2Plan dataset with larger number of data points is used.

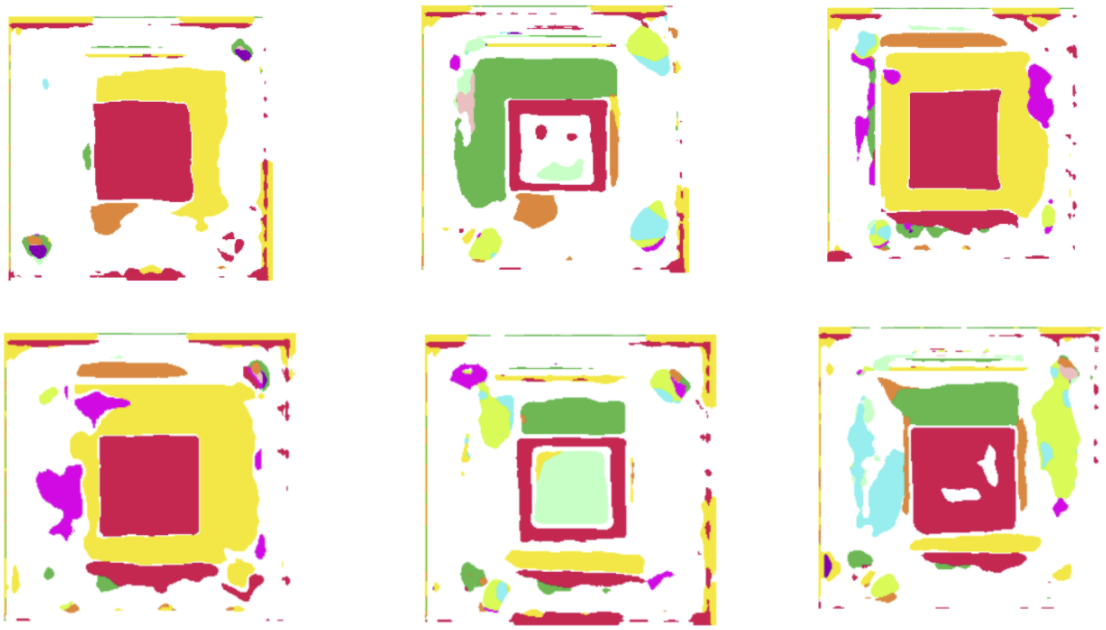

Figure 5.5. Generated floorplans for WGAN for Structured3D

5.3.2 Graph2Plan

5.3.2.1 Data Preprocessing

As discussed in Chapter 4, this dataset has 67,453 floorplans and all these floorplans were used to train the WGAN with the following pre processing steps:

1. Linear interpolation algorithm was used and the image dimensions were reduced from 1167 x 875 x 3 to 128 x 128 x 3

2. Similar to Structured3D dataset, the boundaries of each room, the outer wall and the door(little yellow line) on the outer wall were removed from each floorplan

3. Now, each raster image is converted into a segmentaion map mapping each unique pixel value to one label. Since there are 6 room types and a white background, a total of 7 labels are possible ranging from 0 to 6. This process creates a segmap of size 128 x 128 with 7 labels.

Figure 5.6 shows the steps of data preprocessing from Graph2Plan dataset.

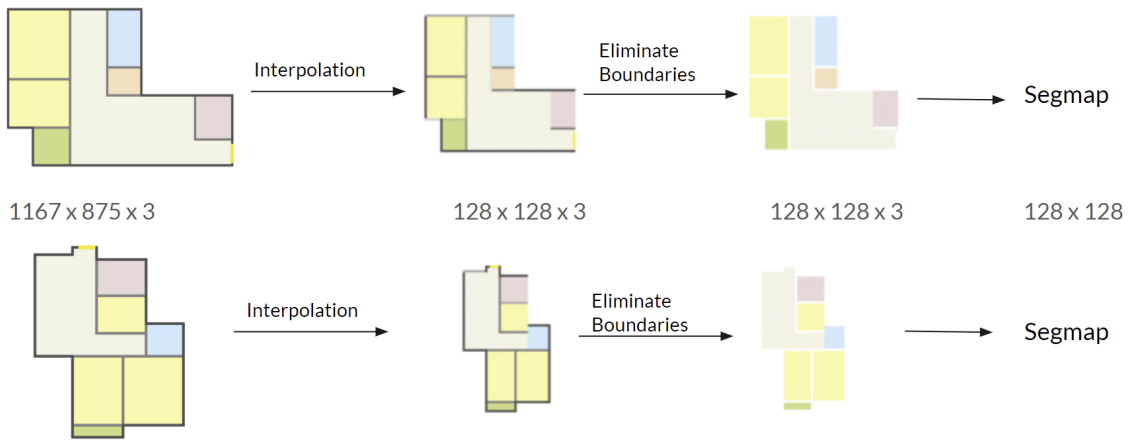

Figure 5.6. Graph2Plan - Data Preprocessing

5.3.2.2 Architecture

Generator: The input layer to this generator takes in uniform random noise with dimension 100. This layer is followed by 3 convolutional layers and upsample layers each with a batch normalization and Relu activation function. Finally, a convolutional layer with depth 7 is added along with a softmax activation layer. This ensures the generated data is of size 128 x 128 x 7

Critic: This critic network gets data of size 128 x 128 x 7 for the input layer. This input layer is followed by 3 convolutional layers each with a LeakyRelu activation function. Finally, a linear activation layer is added to produce the scores.

Loss function: Wasserstein loss is used similar to the previous WGAN model. This model also uses the same clip constraint.
This WGAN model uses RMSprop optimizer with a learning rate of 0.00005.

5.3.2.3 Data Postprocessing

The generator network generates data of dimensions 128 x 128 x 7. This data needs to be converted into RGB images to visualize the floorplans. Following are the steps followed for the conversion:

1. Convert the data of dimensions from 128 x 128 x 7 to 128 x 128 by using an argmax function

Figure 5.7. Generated floorplans for WGAN for Graph2Plan

2. Reverse map the label at each position of 128 x 128 matrix to the original pixel value. This steps ensures the creation of raster images

5.3.2.4 Results

This model was trained with 67,453 images for 200 epochs with a batch size of 32. Figure 5.7 shows few generated floorplans after training the model for 200 epochs.

5.3.2.5 Observations

This WGAN model with Graph2Plan dataset has by far produced the best results. The generated images have straight lines, maintained a good geometric shape and all the major room types are included in each floorplan. We can see that one room type - storage is missing from all the floorplans. This can be because of the under representation of this room type in the original dataset. From this model we can conclude that, data plays a major role and for a GAN to learn the proper distribution, it has to be trained on a larger dataset.

5.4 StyleGAN

StyleGAN called as Style Adversarial Networks was designed to generate high quality images by NIVIDA researches, Karras et al. [**?**]. They have used this GAN to produce

a very high quality human faces dataset. They have proposed many changes to the generator network which led to better generator of images.

5.4.1 Data Preprocessing

This model is trained with Graph2Plan dataset only and the preprocessing steps include only the first two steps of WGAN - Graph2Plan model i.e., this model is trained with only raster images and not the segmentaion map.

5.4.2 Architecture

Generator: The StyleGAN includes an other network called as Mapping network in the Generator to map latent space to an intermediate latent space which controls the style at each point. Figure 5.8 shows the generator architecture. Looking at the figure, we can see that the output of mapping network is used to control the style. The synthesis network is the network which generates fake images. This network is progressive, meaning, the network first generates small size images like 4 x 4, trains until both generator and discriminator are stable and then the size of the output image is doubled. This continues till the desired resolution is reached. Looking at the architecture in Figure 5.8 we can see that different noise sample gets added to the network at each point to introduce variation at each level.

The StyleGAN trained for this thesis used random samples from normal distribution as noise and the Synthesis network grows till it reaches a resolution of 128 x 128.
Discriminator: The discriminator is similar to WGAN Graph2Plan dataset except the input layer which takes data in 128 x 128 x 3 dimensions.
Loss function: Wasserstein loss is the loss function used by StyleGAN. This model also uses Lrelu activation functions, and Adam optimizer.

5.4.3 Results

This model was trained with 67,453 images for 20,000 iterations on 4 GPUs with a batch size of 16. Figure 5.9 shows few generated floorplans after training the model for 20,000 iterations.

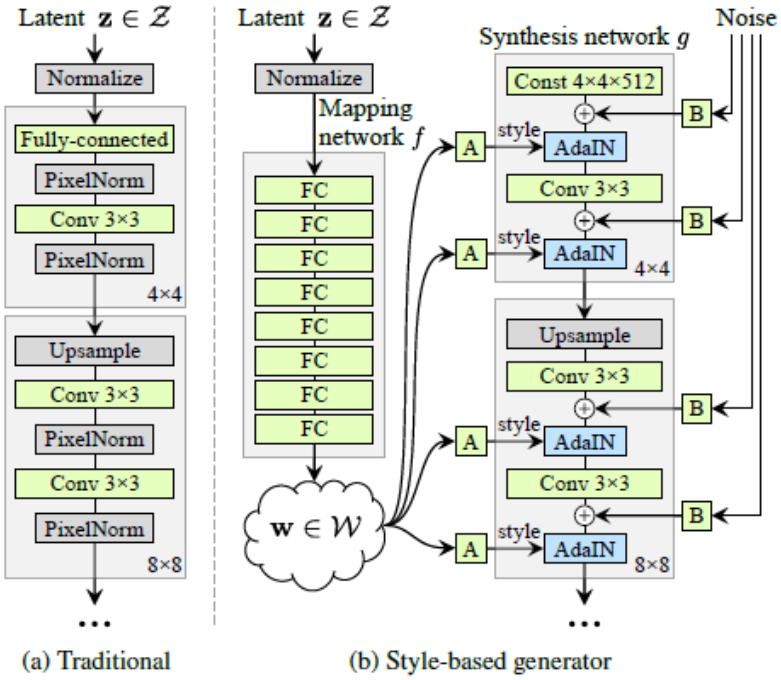

Figure 5.8. Generator architectures of traditional GAN and StyleGAN [**?**]

Figure 5.9. Generated floorplans for StyleGAN

5.4.4 Observations

StyleGAN has produced better results without color mixing, even without a segmentation map. The generated floorplans have maintained the geometric shapes with straight lines along with maintaining the relative sizes of rooms. The color of kitchen room type has changed from the original dataset, but this color remained constant throughout the generated floorplans.

Chapter 6
Conditional floorplan Generation

This chapter talks about the attempts made to introduce input constraints/conditions to GAN models and control the layout of generated floorplans. Due to the very small size of Structured3D dataset, only Graph2Plan dataset was used to train the following models.

6.1 WGAN

6.1.1 Data Preprocessing

Data preprocessing steps for input data are the same as the ones used in WGAN - Graph2Plan model.

Labels: The input labels were generated from the data set. Following are the steps followed:

1. Since the datset has 7 labels corresponding to 7 different colors corresponding to 6 room types and a background, the corresponding y label to each floorplan was designed as a linear vector of size 1 x 7.

2. Each position represents the presence or absence of the corresponding room in the floorplan. 1 in the position represents presence of that room type and 0 in the positions represents the absence of that room type.

Table 6.1 shows the label positions and corresponding room type. Also, Figure 6.2 shows few examples of input data to the model i.e., floorplans as raster images and the corresponding label as the constraint.

Position	Room type
0	Living room
1	Bedroom
2	Bathroom
3	Kitchen
4	Balcony
5	storage
6	Background

Table 6.1. label positions and corresponding room type.

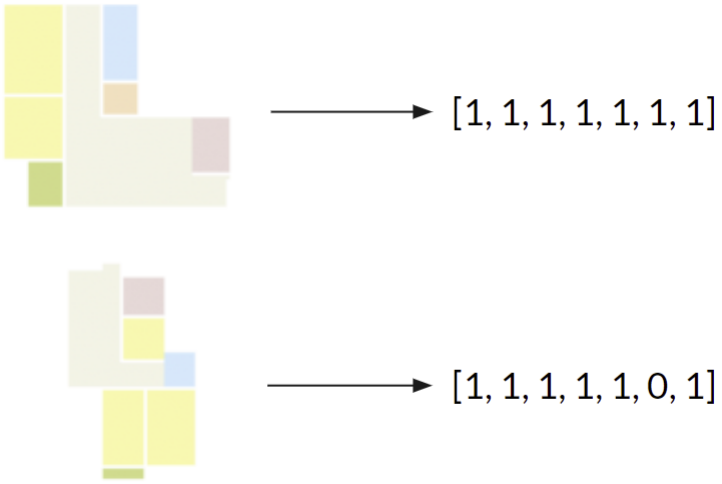

Figure 6.1. Label generation for conditional WGAN

6.1.2 Architecture

The architecture of conditional WGAN is very similar to that unconditional WGAN except that it now includes labels. Generator takes noise and label in the input layer and generates the floorplans accordingly. Discriminator takes labels and floorplans in the input layer and calculates the Wasserstein distance.

6.1.3 Results

This model was trained with 67,453 images for 200 epochs with a batch size of 32. Figure 6.2 shows few generated floorplans after training the model for 200 epochs. The corresponding labels used for the generation of that floorplan is placed on top of each image.

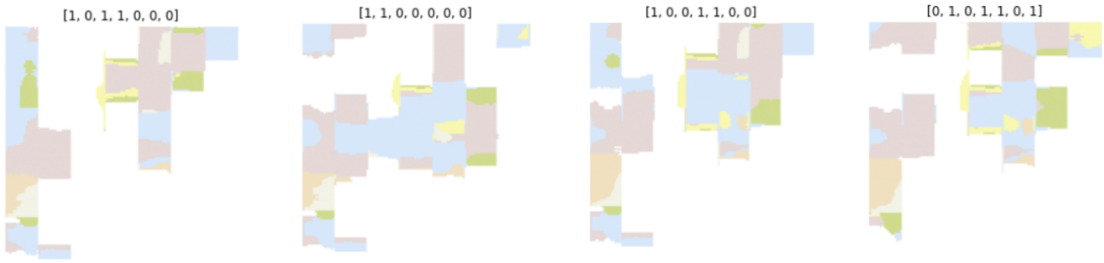

Figure 6.2. Generated floorplans for conditional WGAN

6.1.4 Observations

Conditional WGAN couldn't generate better floorplans. Looking at the results one can say that, even though straight lines were maintained in the boundaries, there is no definite shape for rooms and the floorplan is not continuous.

6.2 StyleGAN

6.2.1 Data Preprocessing

Even for this model, data preprocessing steps for input data are the same as WGAN - Graph2Plan model.

Labels: The input labels were created form the dataset. The y labels for this model are generated as follows:

1. For this model only 6 labels are considered eliminating background color which is present in all the images
2. Firstly, a single dimension vector of size 1 x 6 is created which indicates the presence or absence of a particular room type. The Table 6.1 shows the mappings of positions and room type without the background label. We only have 0 to 5 positions
3. Next, considering this vector as a binary string, a decimal number is generated by converting the binary. This generated decimal scalar is then fed to the network as the y label. Figure 6.3 shows how the labels are created.

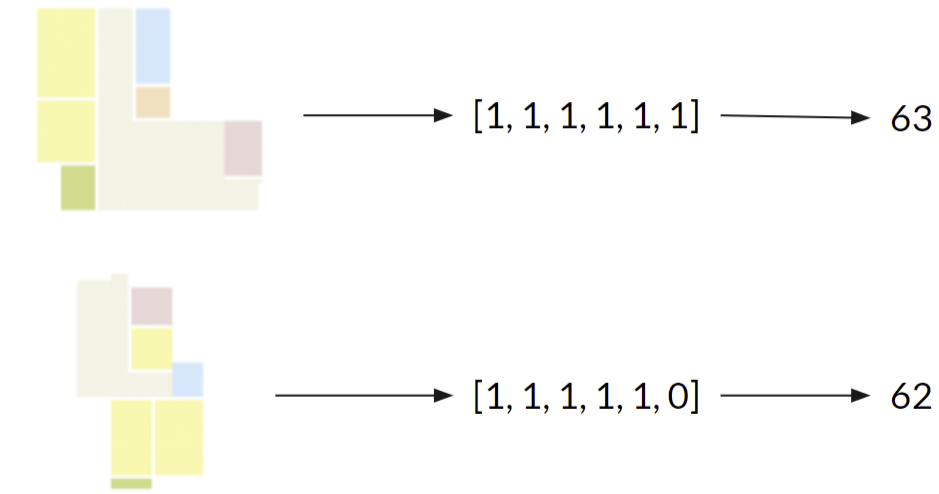

Figure 6.3. Label generation for conditional StyleGAN

6.2.2 Architecture

The architecture is similar to that of the unconditional StyleGAN except for the loss function. The loss function was taken from the work by C. Oeldorf et al. [**?**] to give weightage to the labels in training of StyleGAN. Apart from changing the loss function, few hyper parameters of the model were also modified. Following are the changes made:

1. The pixel normalization which was disabled earlier is now enabled
2. Learning rate of mapping network is changed to 0.001 from 0.01
3. Generator and Discriminator learning rate is changed from 0.0003 to 0.0001
4. The number of times discriminator is trained per generator iteration was changed to 1 from 2

Figure 6.4. Results for conditional StyleGAN for label 58

Figure 6.5. Results for conditional StyleGAN for label 62

6.2.3 Results

This model was trained with 67,453 images for 20,000 iteration on 4 GPUs with a batchsize of 16. Figures 6.4, 6.5 and 6.6 shows the results generated along with their corresponding labels.

Figure 6.6. Results for conditional StyleGAN for label 60

6.2.4 Observations

Conditional StyleGAN has produced the best results overall. Straight lines and geometric shapes were maintained along with relative sizes of the rooms. Also, the floorplans were mostly generated in accordance with the given label, thereby ensuring that the GAN has learnt to generate the floorplans based on the labels/controls given as input. For example, Figure 6.6 shows the floorplans generated for the label 60. All these floorplans are generated according to label 60, which represents the presence of all room types except balcony and storage.

Chapter 7
Evaluation

This chapter talk about the evaluation metrics used to evaluate the generated floorplans. Only the floorplans generated from models trained with Graph2Plan dataset are considered since the size of Structured3D dataset is relatively very small. Therefore, only the unconditional WGAN, StyleGAN and conditional WGAN, StyleGAN are evaluated. Since the results generated are rster images, two types of evaluation is possible, they are:

1. Qualitative Evaluation - Human Evaluation
2. Quantitative Evaluation - FID scores

7.1 Human Evaluation

Human evaluation is generally done by looking at the visuals, in this case floorplans and decide if they are good images based on some rubric set for the given problem. For this evaluation, following are the rubrics set:

1. The rooms and the layout of the floorplan are in straight lines
2. Floorplan has at least 1 living room, 1 bedroom, 1 bathroom and 1 kitchen
3. The size living room is greater than bedroom which is greater than kitchen and bathroom

This evaluation is performed on 50 images from results of each model and the Table 7.1 shows the values. For each rubric, number of floorplans following that rubic out of 50 images is given and then finally an average score is computed. Therefore, the higher the average the better the results. Looking at the table, one can easily tell that Conditional StyleGAN has generated better results compared to all models.

Model	Straight lines	Four rooms	Relative size	Average Score (↑)
Unconditional WGAN	21	15	37	0.486
Unconditional StyleGAN	33	31	46	0.733
Conditional WGAN	0	2	0	0.013
Conditional StyleGAN	33	31	48	0.833

Table 7.1. Human evaluated scores of models for Graph2Plan dataset on 50 images

7.2 FID scores

FID score called as Frechet inception distance is a quantitative metric used to access the quality of images. FID score is a measure of similarity between two sets of images. FID score can be calculated between the real floorplans and the generated floorplans. The lower the score, the better the quality of generated floorplans. But FID score is not a good metric for this problem statement because, the goal of this problem is to generate new floorplans not to just replicate the original ones. Also, FID score cannot capture the conditional constraints enforced onto the network. However, FID scores are computed to show the relative difference between the models. 500 images from each model were considered in computing the score. Table 7.2 shows the computed scores.

Model	FID score (↓)
Unconditional WGAN	1294.5840
Unconditional StyleGAN	60.3847
Conditional WGAN	2999.3934
Conditional StyleGAN	89.9564

Table 7.2. FID scores of models for Graph2Plan dataset on 500 images

Looking at the table, we should come to a conclusion that unconditional StyleGAN produced better results because as mentioned above, FID cannot take conditions into account. Therefore, for the purposes of this thesis human evaluation is a better metric and according to the results shown in Table 7.1, we can conclude that Conditional StyleGAN produces better results.

Chapter 8
Conclusion and Future Scope

8.1 Conclusion

Following can be concluded based on the results generated and the discussions above:

1. Dataset plays a huge difference in the training of GAN, we have seen that for the same unconditional WGAN model change of dataset from Structured3D to Graph2Plan produce far better results
2. There is a lot of uncertainty to the GANs. Understanding the modal collapse problem and hyper-parameter tuning for better training of the GANs is tricky
3. For evaluation, human evaluation is a better metric than FID score in this thesis
4. Overall, conditional StyleGAN has produced better results for the mentioned labels.

8.2 Future Work

There are many interesting and possible ways in which one can explore and add to this work.

1. Looking at the results of StyleGAN, one can notice that there is a problem of color mixing for the kitchen room type. This can be fixed by introducing segmentation maps in conditional and unconditional StyleGANs.
2. Designing better labels as conditions to the GAN, to control the training of GAN will always be an interesting problem.

3. This problem of designing floorplans for a single level can be extended to a problem of floorplan generation with multiple levels. This is a great idea to work on, since places with heavly population need multilevel floorplans. Factors like staircase and the height of each level should be considered to solve this problem.

4. Introduction of building-codes in the generation of floorplans is another interesting idea to work on. Shape grammars may be used in this context to develop floorpalns according to the building-codes. Learning of shape grammars from the given data is yet an interesting idea.

Bibliography

[1] Dickinson, D. (2016), "Architects Design Just 2% of All Houses–Why?" . URL https://commonedge.org/architects-design-just-2-of-all-houses-why/

[2] Hu, R., Z. Huang, Y. Tang, O. Van Kaick, H. Zhang, and H. Huang (2020) "Graph2Plan," *ACM Transactions on Graphics*, **39**(4). URL http://dx.doi.org/10.1145/3386569.3392391

[3] Nauata, N., K.-H. Chang, C.-Y. Cheng, G. Mori, and Y. Furukawa (2020), "House-GAN: Relational Generative Adversarial Networks for Graph-constrained House Layout Generation," 2003.06988.

[4] Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014), "Generative Adversarial Networks," 1406.2661.

[5] Feng, J., X. Feng, J. Chen, X. Cao, X. Zhang, L. Jiao, and T. Yu (2020) "Generative Adversarial Networks Based on Collaborative Learning and Attention Mechanism for Hyperspectral Image Classification," *Remote Sensing*, **12**, p. 1149.

[6] Zheng, J., J. Zhang, J. Li, R. Tang, S. Gao, and Z. Zhou (2020) "Structured3D: A Large Photo-realistic Dataset for Structured 3D Modeling," in *Proceedings of The European Conference on Computer Vision (ECCV)*.

[7] Song, S., F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser (2016), "Semantic Scene Completion from a Single Depth Image," 1611.08974.

[8] McCormac, J., A. Handa, S. Leutenegger, and A. J. Davison (2017), "SceneNet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth," 1612.05079.

[9] Wu, W., X.-M. Fu, R. Tang, Y. Wang, Y.-H. Qi, and L. Liu (2019) "Data-driven Interior Plan Generation for Residential Buildings," *ACM Transactions on Graphics (SIGGRAPH Asia)*, **38**(6).

[10] Arjovsky, M., S. Chintala, and L. Bottou (2017), "Wasserstein GAN," 1701.07875.

[11] Karras, T., S. Laine, and T. Aila (2019), "A Style-Based Generator Architecture for Generative Adversarial Networks," 1812.04948.

[12] Oeldorf, C. and G. Spanakis (2019), "LoGANv2: Conditional Style-Based Logo Generation with Generative Adversarial Networks," 1909.09974.