Digital Tools for
Generative Building Design

MASTER IN ADVANCED COMPUTATION
FOR ARCHITECTURE & DESIGN
2020/2021

Deep Convolutional GAN
FLOOR PLAN GENERATOR

FACULTY: Stanislas Chaillou
FACULTY ASSISTANT: Oana Taut

STUDENTS:
Jaime Cordero Cerrillo
Jonathan Hernández López

ABSTRACT

00 INTRODUCTION

The use of Artificial Intelligence is expanding over the architecture field. It is inevitable to think about it as a tool for designing. Following this line, the goal of this study is to generate schematic floor plan configurations based on the relationship between the spaces. This is the background to explore and study Deep Convolutional Generative Adversarial Networks (DCGAN), which is used to generate the floor plans.

As a reference to generate the initial input dataset, it was taken the Preescolar System designed by Giancarlo Mazzanti. In this system, some different modules are aggregated on a systematic way, adapting to some specific conditions to design schools.

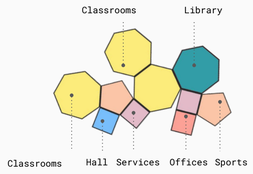
In the same way, in this project different modules representing each of the spaces of a building, are aggregated using WASP (a plugin for Grasshopper). These spaces could follow some predefined rules of aggregation.

Once the dataset was created, it was used as the excuse for exploring DCGANs and its performance. Given a preconfigured model architecture, the hyperparameters of these model have been changed to find the best performance for the created dataset through several iterations. Here they are shown three of the most representative trials, which have helped to develop the exposed conclusions.

01 GOAL

BUILDING TYPOLOGY GENERATOR

Generate schematic floor plan configurations based on the relationship between spaces given to the initial dataset.

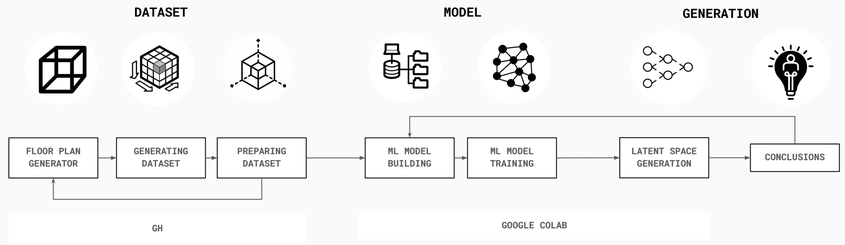


02 REFERENCE

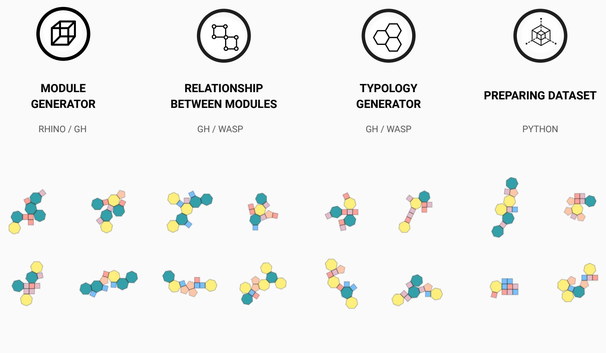
G. MAZZANTI - PREESCOLAR SYSTEM

https://www.archdaily.mx/mx/941638/sistema-preescolar-cdi-del-equipo-mazzanti-para-el-departamento-del-atlantico-en-colombia

03 METHODOLOGY



04 DATASET



05 MODEL ARCHITECTURE

DISCRIMINATOR

```
model = Sequential()
model.add(Conv2D(128,(5,5), padding='same', input_shape=in_shape))
model.add(LeakyReLU(alpha=0.2))
model.add(Conv2D(128, (5,5), strides=(2,2), padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(Conv2D(128, (5,5), strides=(2,2), padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(Conv2D(128, (5,5), strides=(2,2), padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(Conv2D(128, (5,5), strides=(2,2), padding='same'))
model.add(LeakyReLU(alpha=0.2))

# OUTPUT LAYERS
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(1,activation='sigmoid'))
opt = Adam(learning_rate=LR_disc,beta_1=0.5)
model.compile(loss='binary_crossentropy', optimizer=opt,metrics='accuracy')
```

GENERATOR

```
model = Sequential()

n_nodes = 128*5*5

model.add(Dense(n_nodes, input_dim=latent_dim))
model.add(LeakyReLU(alpha=0.2))
model.add(Reshape((5,5,128)))
model.add(Conv2DTranspose(128,(4,4),strides=(2,2),padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(Conv2DTranspose(128,(4,4),strides=(2,2),padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(Conv2DTranspose(128,(4,4),strides=(2,2),padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(Conv2DTranspose(128,(4,4),strides=(2,2),padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(Conv2D(3,(5,5),activation='tanh',padding='same'))
```

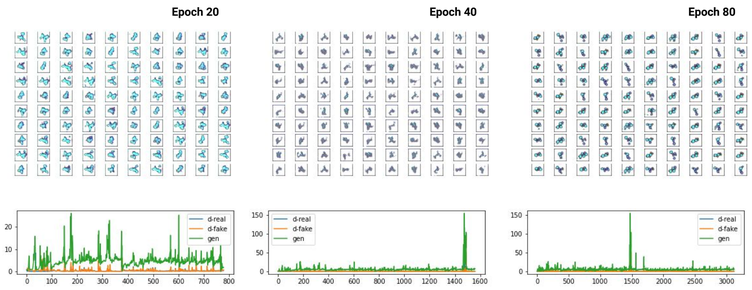
RUN 1

HYPERPARAMETERS
latent_dim = 15
LR_disc = 0.0002
LR_gen = 0.001
np_epochs = 80
n_batch = 128

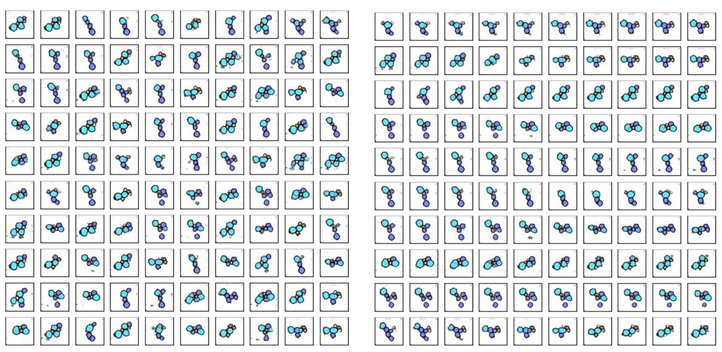
HYPOTHESIS

Exploration on how the model works when using a low latent dimension. The other hyperparameters are kept on a regular range to see the effect that the latent dimension has. It is expected to work well, as the latent dimension is understood as the input shape for the generator, which shouldn't affect much the behavior of the model.

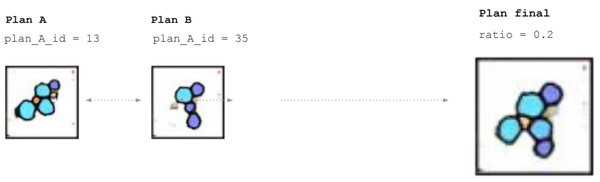
RESULTS



2 interpolated points



IMAGES GENERATOR



RUN 01 CONCLUSIONS

Run 01 shows higher diversity since it was run with the lowest latent dimension.
The learning rate 0.001 seems to be a stable value, but only good results were shown at the end of the 80 Epochs.
Overall it seems that the chosen values were promising it only needed to be run with a higher amount of epochs
-

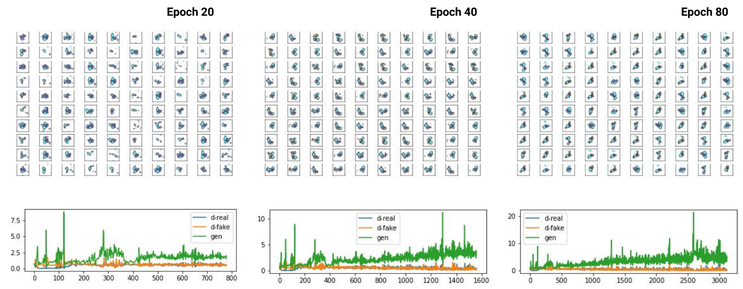
RUN 2

HYPERPARAMETERS
latent_dim = 25
LR_disc = 0.00007
LR_gen = 0.0009
np_epochs = 80
n_batch = 128

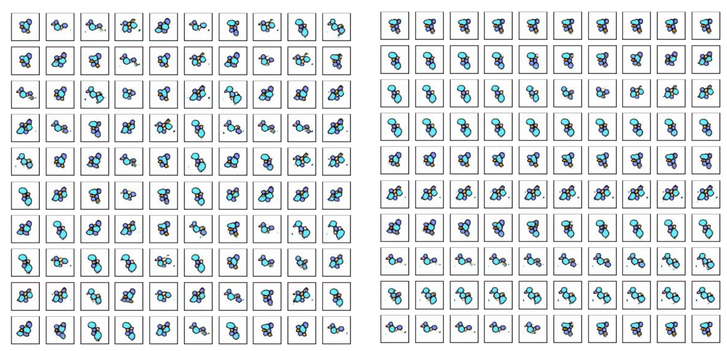
HYPOTHESIS

After seeing the effect of latent dimension it was increased to see the change of it. Both learning rates for the discriminator and the generator were lowered together. As it was seeing that sometimes the discriminator was falling to mode collapse and "winning" the generator, it was lowered a bit more. It is expected to work better.

RESULTS



2 interpolated points



IMAGES GENERATOR

RUN 02 CONCLUSIONS

Run 02 shows a slightly increased later dimension and therefore a slightly less diversity than the first run.
The learning rate was decreased to 0.0007 showings as well good results, but at a slower pace.
It could have been run with more epochs, but it helped to conclude whether to increase or decrease the final values for the 3rd Run.

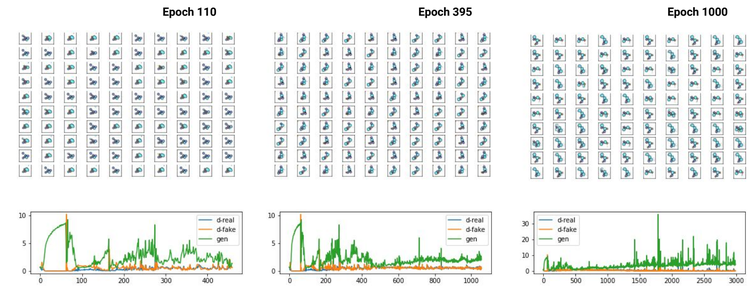
RUN 3

HYPERPARAMETERS
latent_dim = 100
LR_disc = 0.0001
LR_gen = 0.001
np_epochs = 1000
n_batch = 128

HYPOTHESIS

The previous iterations seemed to be stopped in the middle of a good training. To let the model train, the number of epochs was increased. The latent dimension was also increased to 100.

RESULTS



2 interpolated points

IMAGES GENERATOR

RUN 03 CONCLUSIONS

Run 03 shows an increased later dimension which led the results to a higher uniformity of options.
The learning rate was proposed based on the balance between Run 1 and Run 2. Since the first Epochs were showing successful results, it was decided to run till the end of 1000 Epochs. Finally, Epoch 1000 was used as weight fur the final results.

CONCLUSIONS

- The third run seemed to be the best one. Although it was trained for more epochs, on early stages it also looked better. However, it is not as diverse as the first one.
- The **bigger the latent dimension** the **less variability** it was obtained in the generations.
- **Discriminator learning rate** should be **1/10 generator's learning rate** because of the way the model is built, otherwise it comes quickly to mode collapse. Sometimes it could even work better to have a smaller than 1/10 (as in the second run:0.00007-0.0009).
- It needs a lot of time to train and a **high number of epochs**.
- **Batch size did not affected** that much in other iterations that were tested, although it should be balanced with the latent dimension and learning rates in order not to collapse.
- The model had some problems when generating the colors, specially blue, as in the initial dataset there wasn't that much as it was generated and blue was assigned to small squares, although the generator assigned it to big modules.

NEXT STEPS

- Although the last run seemed to work fine,it would be good to run the first two runs with more epochs to see better the differences between them at that high number of epochs, because as it has been concluded, 80 epochs wasn't enough and the model stopped on the middle of training.
- The last run, with learning rates of 0.0001 and 0.001, could be trained with a lower latent dimension, as the relationship between learning rates on the first run didn't seem great.
- It would be possible to design some more detailed interior floor plans for each of the modules. Once that the floor plans have been generated with colors; each color could be assigned to one of those distributions and then train a Pix2Pix model to have the generated schematic floor plan as an input and the actual floor plan generated from the Pix2Pix model. In this way, two models of GAN would be combined, being the DCGAN model the generator of the input of the other.