UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Attribute Transfer in Image Generation using Conditional GANs

Degree Thesis

Jorge Pueyo Morillo

In partial fulfillment of the requirements for the degree in
Telecommunications Technologies and Services Engineering

Director: Javier Ruiz Hidalgo
Barcelona, Date 19/06/2021

Contents

List of Figures

Abbreviations

ADGAN Attribute-Decomposed GAN

CNN Convolutional Neural Network

CSAIL Computer Science and Artificial Intelligence Laboratory

DNN Deep Neural Network

GAN Generative Adversarial Network

SPADE Spatially-Adaptive Normalization

Abstract

Deep Learning and Computer Vision have had a rapid evolution over the past few years, and new techniques have risen to give response to multiple tasks. One of those tasks is Attribute Transferring, or the ability to transfer visual components of an image (from textures or shapes to more concrete ones such as facial features) to another.

The goal of this thesis is to research the state of the art methods in this field and develop our own model to perform Attribute Transferring over bedrooms. This model, deployed as part of an app, would allow the user to take a picture of its bedroom and swap its appearance with the one of a desired bedroom, conserving the original elements and their disposition.

Revision history and approval record

Revision	Date	Purpose
0	26/04/2021	Document creation
1	16/06/2021	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Jorge Pueyo Morillo	jorge.pueyo@estudiantat.upc.edu
Aleix Clemens Suriol	aleix.clemens@estudiantat.upc.edu
Javier Ruiz Hidalgo	j.ruiz@upc.edu

Written by:		Reviewed and approved by:	
Date	26/04/2021	Date	16/06/2021
Name	Jorge Pueyo Morillo	Name	Javier Ruiz Hidalgo
Position	Project Author	Position	Project Supervisor

1 Introduction

The aim of this thesis is to examine the state of the art in Deep Learning applied to images, specially the Generative Adversarial Networks, and the techniques used to perform attribute transfer. Attribute transferring involves any task with the objective of transferring a visual attribute from one image to another. These attributes include facial features (hair, mouth, glasses) or clothes, but also more generic ones such as shape or color.

Originally we planned to apply our research to perform attribute transfer over people's clothes, with the final goal being an application capable of switching the clothes from the image of one person over to another one. However, we found out that there were already many solutions for this problem, so we decided to apply our research to perform attribute transfer over rooms, more precisely bedrooms.

We begin by examining the Attribute-Decomposed GAN (ADGAN) model [12], which can perform both pose and clothes transfer over people. Using ADGAN as our foundation, we have developed a new model called RoomGAN, which performs attribute transfer over bedrooms (Figure 1). Our method can transfer the shape, pose and texture from the different components of a bedroom (bed, lamp, window, ...) to another one.

(a) Desired appearance (b) Original bedroom (c) Result

Figure 1: Attribute transfer over bedrooms.

The motivation behind this research was to end up offering an application that would allow the user to visualize how their bedrooms would look like in a different style. The user would be able to take a picture of their own bedroom and the picture of another bedroom whose appearance would like to have, and the app would output the result with the smallest delay possible.

1.1 Objectives

The objectives of this thesis can be summarized as:

- Research the state of the art in attribute transfer and select the most appropiate model to use as foundation.
- Achieve a method capable of performing shape, pose and texture transfer over two bedroom images, on a set of defined components.
- Achieve photo-realistic outputs.
- Compare RoomGAN to other state of the art methods for the specific task of attribute transfer over bedrooms.
- Adapt the model to be ready for deployment as part of an app.
- Keep the total processing time relatively low, so the model can be practical in real-life scenarios.

1.2 Planification

This research was performed in collaboration with Aleix Clemens, a colleague of mine who is also a student of Telecommunications Technologies and Services Engineering. We have shared all the effort of investigating and developing RoomGAN, but while Aleix's focus was to create a mobile app that could be used as a fronted to our method, mine was to deepen the research into some alternative state of the art methods, and compare them to our own.

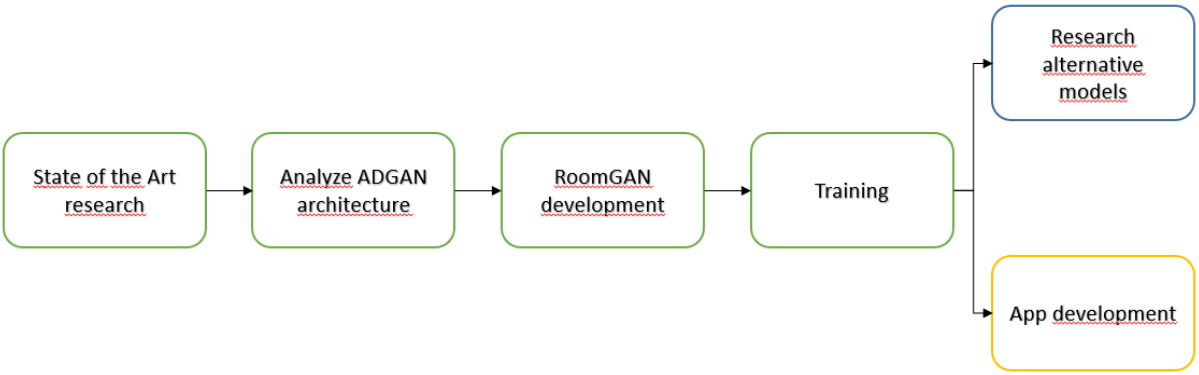


Figure 2: Separation of tasks performed. Green indicates tasks performed by both of us, yellow denotes tasks performed by Aleix and blue for the ones I carried out.

This document has also been a common effort between Aleix and I. We share some of the subsections, including the following:

- Section 1
- Section 2.1, 2.2.1, 2.2.2 and 2.2.3
- Section 3.1 and 3.2
- Section 4.4

Finally, Figure 3 shows the original Gantt diagram of the project, which we managed to carry as planned. Green denotes the common work packages, while blue denotes the ones exclusively performed by me.

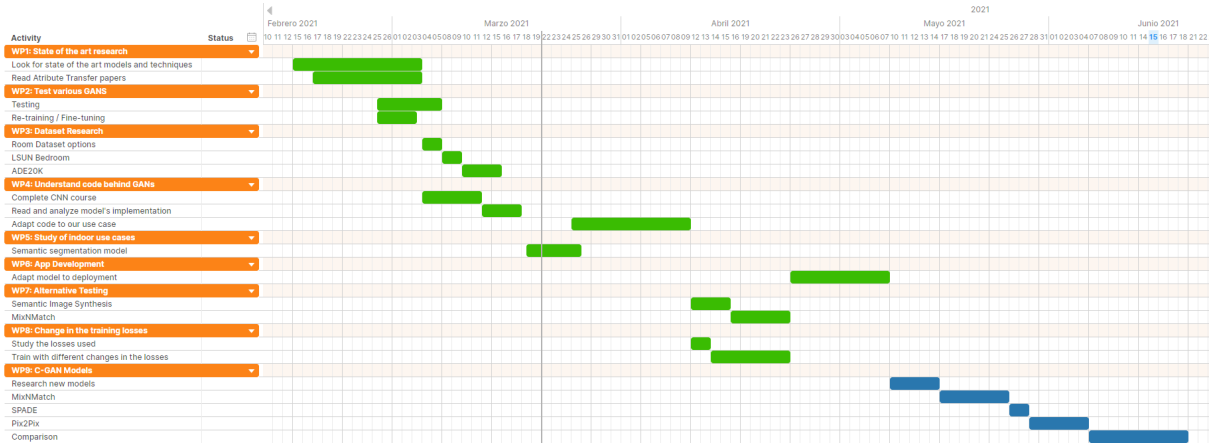


Figure 3: Gantt diagram of the project

2 Review of the Literature

2.1 Fundamentals

In the next section we will give a brief overview on two of the techniques that are predominantly used in Computer Vision and in Attribute Transferring tasks: Convolutional Neural Networks (CNN) and Generative Adversarial Networks (GAN).

2.1.1 Convolutional Neural Network

CNNs are a deep learning algorithm, predominantly used in Computer Vision tasks. They present multiple advantages with respect to classical Neural Networks, specially when working with images as inputs. Not only are CNNs able to capture spatial and temporal dependencies in an image, but they also reduce the number of parameters involved in the model, thus saving computation time and storage.

The main difference is that CNNs are able to reduce images into simpler forms, which are easier to process, by performing convolutions over multiple layers, called Convolutional Layers, of the network. The element involved in carrying these convolutions (Figure 4) is called the Kernel or Filter. The kernel can vary on dimensions between layers of the same network, and its values are trainable parameters.

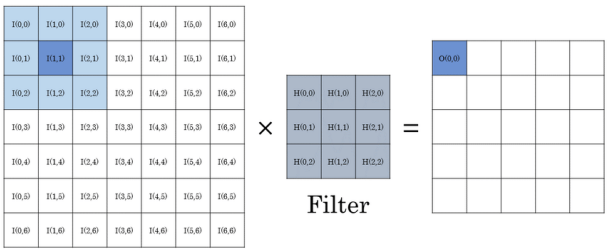


Figure 4: Single step of a 2D convolution operation, using a 3x3 kernel [15]

Usually, the first Convolutional Layer is responsible to extract low-level features from the input image, such as edges or color, while the following layers focus on extracting progressively higher-level features, providing a complete understanding of the image.

Apart from Convolutional Layers, a conventional CNN will also present a series of Pooling Layers, responsible of reducing the dimensions of the outputs from the different Convolutional Layers in order to decrease the computational power required to process all the data, while also extracting some dominant features. There are two ways to perform the pooling, average and maximum. Like shown in Figure 5, they both take a small patch of the data, and return a value, which can be the maximum or the average of said patch.

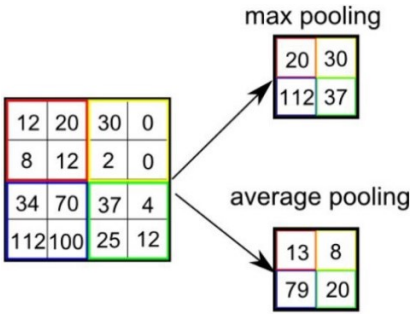


Figure 5: Types of pooling [10].

Once all the relevant features are extracted, and the data has an adequate format, it can be fed to a fully-connected layer. A classic example is to add a Softmax layer after the fully-connected layer to perform classification tasks, and then apply the backpropagation throughout the whole network. A typical example of the structure of a CNN can be seen in Figure 6.

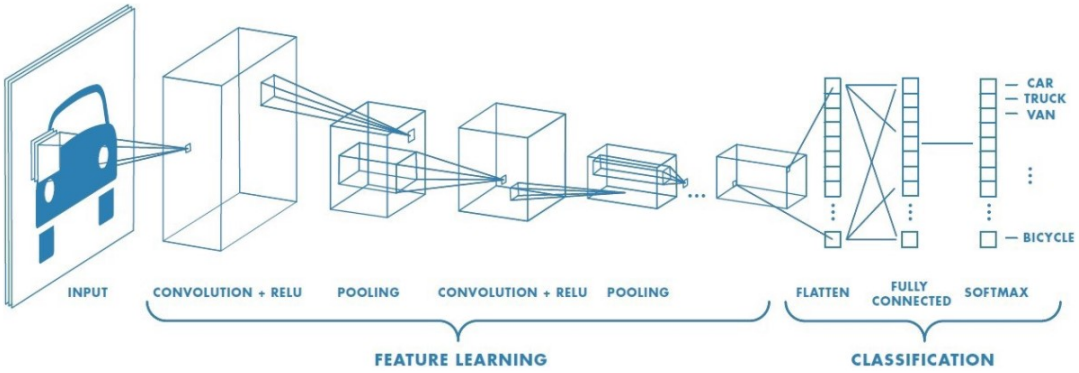


Figure 6: Example of a CNN structure [14].

2.1.2 Generative Adversarial Network

GANs are a deep learning method used in image generation tasks. Generative models are those that automatically discover and learn the regularities or patterns in the input data, in such a way that the model can be used to generate new examples that resemble the original data.

To train these generative models, GANs propose a structure (Figure 7) consisting of two sub-models: Generator and Discriminator. The generator tries to output new examples from the same distribution, while the discriminators tries to classify the examples as real or fake.

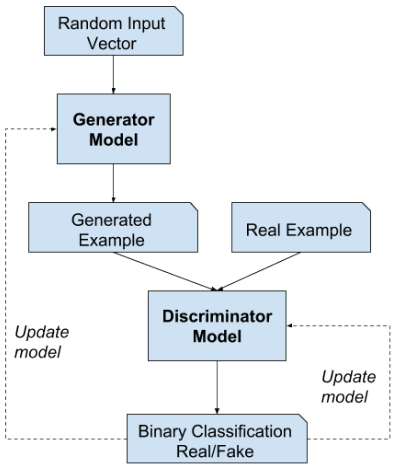


Figure 7: GAN structure [2].

The discriminator is a regular classification model, that will learn to distinguish between real data (coming from the dataset) or fake (generated). This sub-model will provide feedback to the generator, and will be discarded after training since it isn't needed in the generation process. The generator is trained based on how well its generated images have been able to fool the discriminator in the previous iteration.

The two models are trained and the same time and compete with each other, until the generator is able to consistently fool the discriminator. In an ideal case, the discriminator would predict the same probability of being real or fake for every example.

2.2 State of the Art

In the section below we will be reviewing some of the state of the art techniques applied in conditional image generation tasks, specially those involving attribute transfer.

2.2.1 StyleGAN2

The style-based GAN is one of the state of the art architectures for unconditional generation of images. It allows for an automatically learned and unsupervised separation of high-level attributes, such as pose or shape, as well as better interpolation properties and better disentanglement of the latent factors. The StyleGAN2 model [6] is an enhanced version of the original StyleGAN [5], which yields even better results, and which has become the base for many conditional and unconditional generative networks.

2.2.2 VOGUE

VOGUE [8] proposes a novel way to perform garment transfer. Given an image of a target person and an image of another person wearing a garment, this method generates the target person in the garment worn by the other one, like shown in Figure 8. It is based on the StyleGAN2 model, to which has been added an algorithm to automatically optimize the interpolation of the different areas of interest from each image.

Figure 8: Demonstration of the VOGUE capabilities. This method can switch both top and bottom garments.

Presented in January 2021, it is one of the most recent models available in the literature, which also yields the best results in terms of garment transfer. However, the official implementation of this model is not public yet, which makes impossible to verify its performance.

2.2.3 ADGAN

The Attribute-Decomposed GAN method [12] introduces a conditional generative model that produces realistic person images with the desired head, upper clothes and pants, in any given pose. These attributes can be provided via different inputs, thus allowing the possibility to perform pose transfer, like shown in Figure 9, or garment transfer (Figure 10).

Figure 9: Source person synthesized in different poses.

Figure 10: Garment transfer on both upper and bottom clothes.

This method does not require any labeled data and can achieve component separation in an unsupervised way, but it does require that the training dataset contains images of the same person wearing the same outfit in different poses, which makes applying this method to other use cases complicated. This model will be the baseline for our own method, explained in Section 3.2.

2.2.4 MixNMatch

MixNMatch [9] is a conditional generative model that uses a multi-factor disentanglement and encoding. This translates into a model capable of separating the shape, pose, texture and background of an image in an unsupervised way (it only requires bounding boxes around the objects of interest during the training phase). Then, the model can use these separated features to create completely new images, as seen in Figure 11.

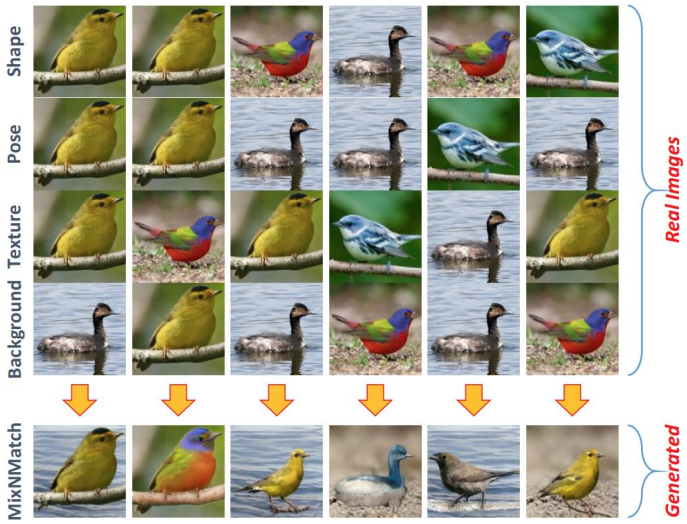


Figure 11: Example of usage proposed by MixNMatch.

2.2.5 Pix2Pix

Pix2Pix [3] is a model that seeks to provide a general solution for the image-to-image translation problem using Conditional Generative Adversarial Networks. The fact that GANs not only learn the mapping between input and output, but also a loss function to train said mapping, makes it possible to apply the same generic approach to different problems (e.g. Figure 12) that would have required very specific solutions using traditional methods like CNNs.

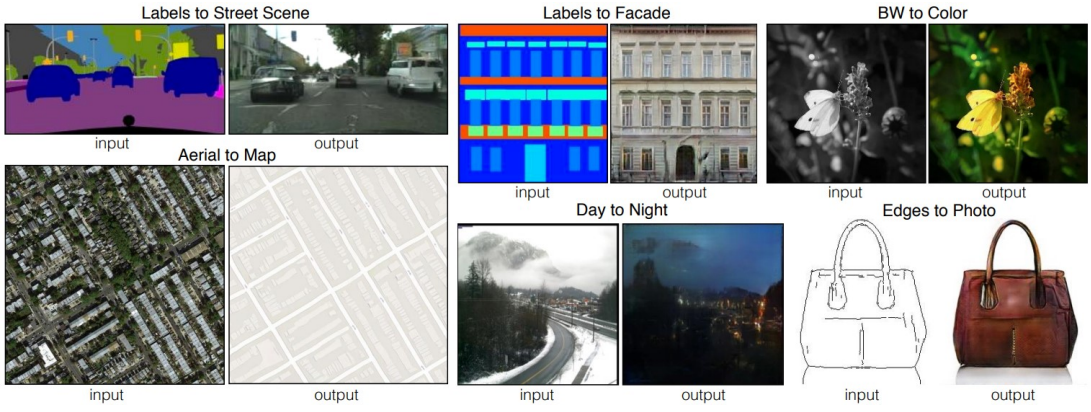


Figure 12: Examples of Image-to-Image translation.

2.2.6 SPADE

The Spatially-Adaptive Normalization (SPADE) model [13] proposes a method to synthesize photorealistic images given an input semantic layout (Figure 13).

The main difference with a traditional Image-to-Image translation approach is that instead of feeding the semantic layout through a Deep Neural Network consisting of normalization, convolution and nonlinear layers, the layout is used to modulate the activations of the normalization layers, which effectively propagates the semantic information throughout the network. This method produces better results, since it has been proven that the normalization layers tend to deteriorate the semantic information from the input.

Figure 13: Output generated by the SPADE model.

Furthermore, SPADE also supports style-guided generation (Figure 14). A style encoder is attached to capture the style of a target image, which will then be fed to the network instead of plain random noise, thus allowing to control the general appearance of the output image.

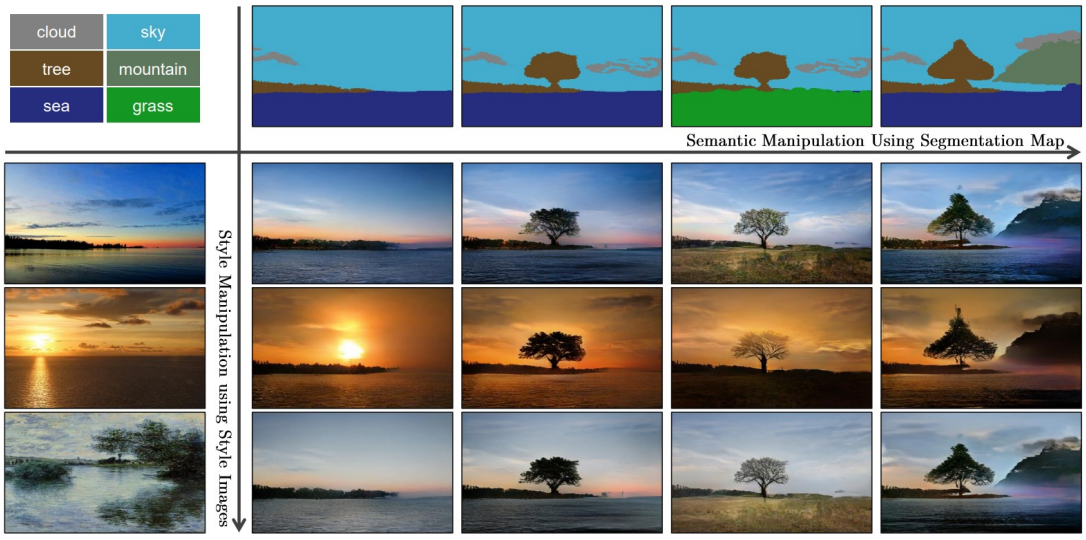


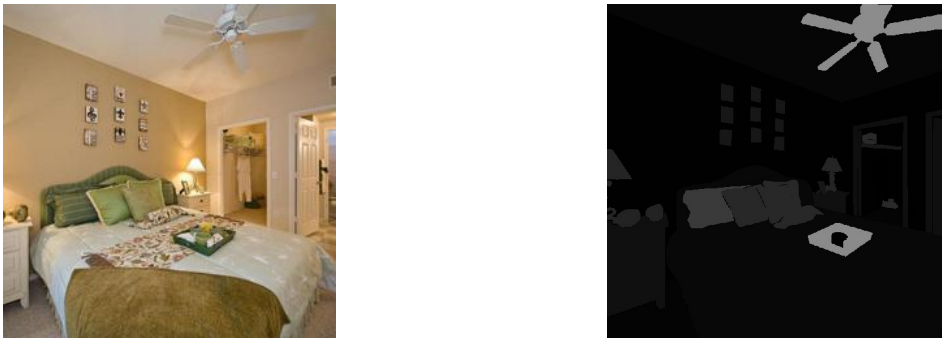
Figure 14: Different synthesized images, generated from the same semantic layout, depending on the image provided as style.

3 Methodology

3.1 Datasets

For the purpose of our research we needed a dataset consisting of images of indoor environments, specially those containing bedrooms. Ideally, the dataset would also contain the ground truth semantic segmentation of all its images.

The only dataset that fullfilled every requirement was the ADE20K [19] (see Figure 15 for an example), from the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), which includes 150 semantic categories. The dataset contains 25,574 training images and 2,000 validation images, with dimensions $256x256$. Of those, 1,130 training images and 204 validation ones correspond to bedrooms.



(a) Image. (b) Semantic segmentation.

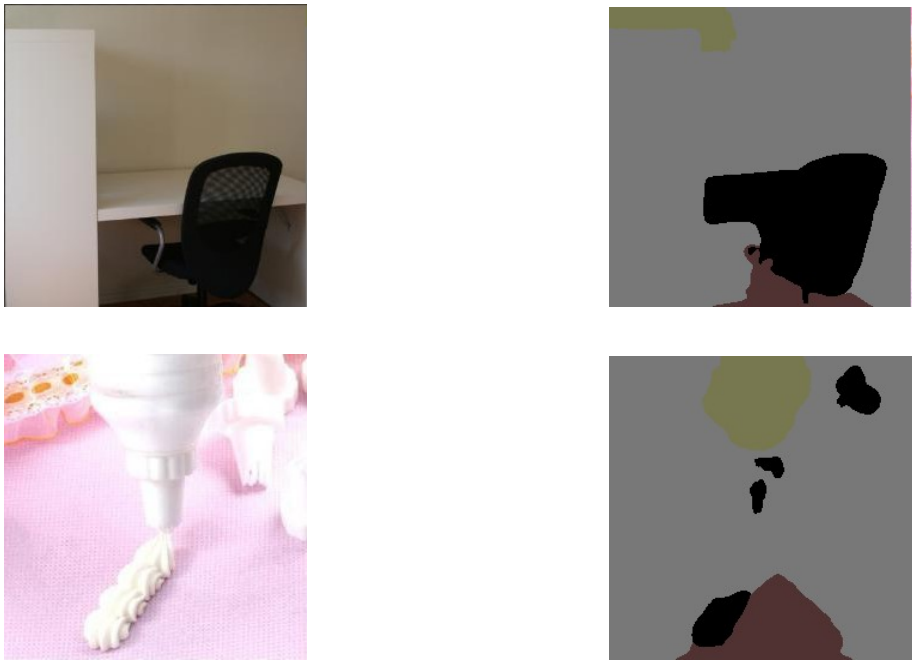
Figure 15: Example of an entry on the ADE20K dataset.

These numbers were enough to start experimenting with the dataset and obtain reasonable training results, but we realized we would need a lot more images if we aimed to obtain respectable test results. Since there were not any other datasets that matched our requirements, we decided to synthesize our own. To find the most number of bedroom images we used the LSUN Bedrooms dataset [17], which consists of 196,224 images.

To obtain the semantic segmentation of said images, we used the CSAIL semantic segmentator [18] with the ResNet18 pretrained model, which offers a good balance between accuracy and speed, something important due to the high number of images to process. We resized all the images to the same $256x256$ dimensions of the ADE20K dataset. Overall, the dataset contains high quality pictures of regular bedrooms, with which the segmentator performs great (e.g. Figure 16). However, some of the images have a poor quality, only show a small part of a room (not even a bedroom) or even have no relation to a room (e.g. Figure 17)

(a) Images. (b) Semantic segmentations.

Figure 16: Example of the generated semantic segmentations.



(a) Images. (b) Semantic segmentations.

Figure 17: Example of bad quality entries on the dataset.

We managed to improve the consistency of the dataset by removing those images in which the segmentator found people and those for which the total of the image that corresponded to a bed was too small (less than 5% of the image). Unfortunately, there were a few outliers remaining which we couldn't automatically remove since they didn't check the two conditions just mentioned (e.g. Figure 18).

(a) Image. (b) Semantic segmentation.

Figure 18: Example of bad quality entries that could not be automatically removed.

3.2 RoomGAN

RoomGAN is our very own method to perform attribute transfer over bedrooms. Our model is capable of synthesizing a bedroom with arbitrary components (bed, windows, wardrobe, ...) in a certain disposition. RoomGAN takes as input two images, a source image, from which the different elements will be taken, and a target image which will be used as the layout for the generated output.

Our method is based on the ADGAN model [12], which is able to produce realistic person images with desired human attributes and in any given pose. The feature that interested us the most is not actually the component transfer part of the model, which can mix characteristics from two different people, but the pose transfer. Our main objective was to use the pose transfer ability of this network to change the layout of a room, thus generating a new bedroom with the same elements but a different layout.

To do so, we performed several changes to the original ADGAN's structure (Figure 19) so as to implement our method.

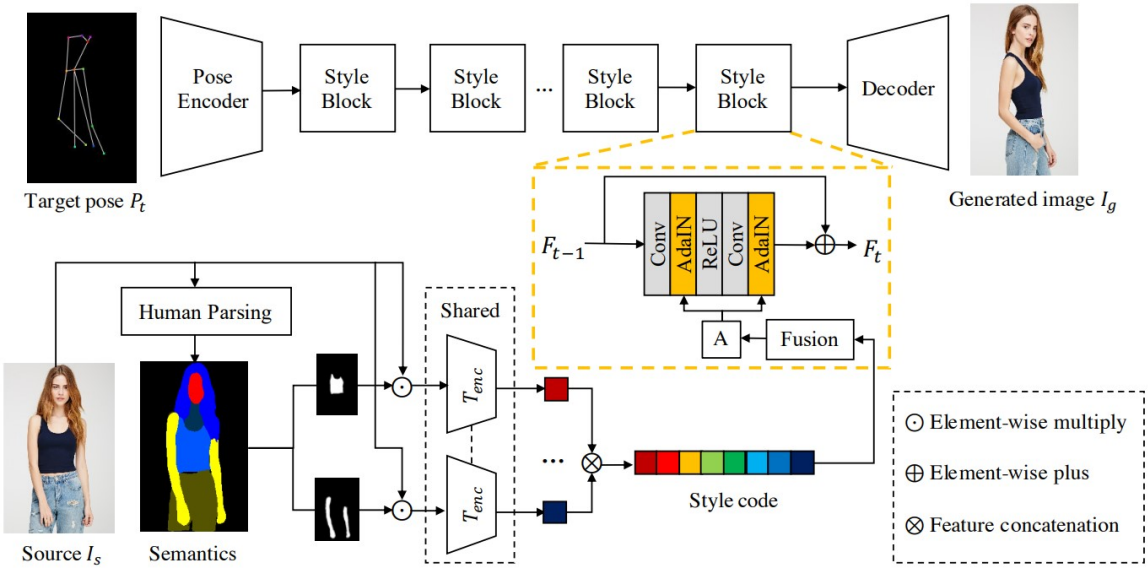


Figure 19: Structure of the ADGAN model.

3.2.1 Layout

During test time, ADGAN takes two inputs to perform the pose transfer: source image, from which we extract the textures, and target image, which gives the desired pose. These poses $P \in R^{18xHxW}$ (e.g. Figure 20) represent the location of 18 joints of the human body, and can be automatically extracted from an image using a pose estimation method [1]. Our first change was to find a suitable substitute for the poses, that could be extracted from a room. Originally, we planned to use RoomNet [7] or similar methods to extract a 2D layout estimation based on keypoints such as floors, walls and ceiling. However, we felt that these few keypoints could not capture the dispositions of all the elements in the room so we finally discarded the idea. Instead, we opted for encoding the semantic segmentation of the room. To do so, we use a layout $L \in R^{NxHxW}$, where N are the number of objects to encode. Ideally we would want to perform the layout transfer to every elements of the room, but since there could be an unlimited number of them we selected the most frequent ones, according to their appearance on the ADE20K dataset, that made sense to be found in bedrooms. We ended up picking 18 objects, out of the 150 available, same as the number of joints used in the original pose:

Figure 20: Examples of pose encoding used by ADGAN.

- Bed
- Window
- Lamp
- Wall
- Chair
- Pillow

- Floor
- Door
- Curtain
- Blanket
- Cabinet
- Ceiling
- Table
- Mirror
- Plant
- Shelf
- Painting
- Carpet

To create a layout L (Figure 21), we begin by splitting the semantic segmentation of an image into 18 different masks, each one corresponding to an object. The mask will be a binary image where 1's represent the section of the original image in which the object was present. In case the image did not contain some of the objects, their masks won't be created since their values would be all 0's. Then, we proceed to stack the masks into the final layouts $L \in R^{18x256x256}$, in which every object has a fixed channel.

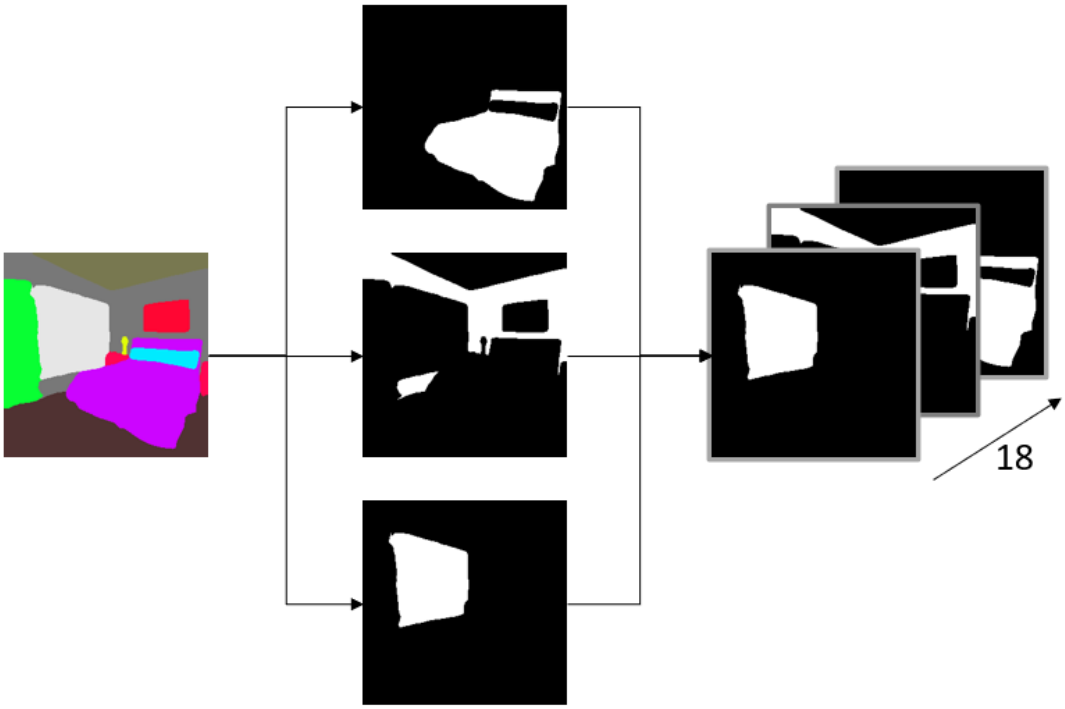
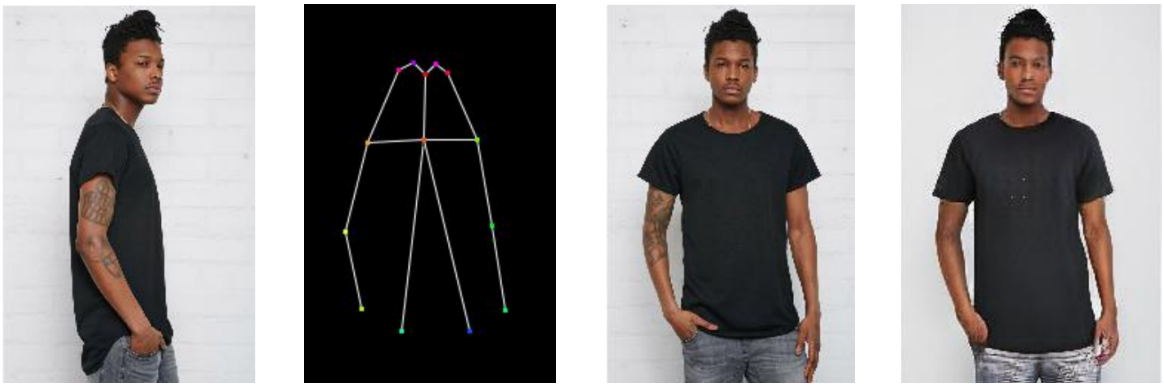


Figure 21: Semantic segmentation to layout process.

3.2.2 Training inputs

At training time, ADGAN takes three inputs (Figure 22): a source image I_s, representing the person whose pose will be altered, a target pose P_t, containing the desired pose of the generated image, and the target image I_t, which is the ground truth of the person shown in I_s posing like P_t.



(a) I_s (b) P_t (c) I_t (d) I_g

Figure 22: Inputs and output of the ADGAN method at training time.

It follows a GAN structure (see Figure 7) with one generator and two discriminators D_p and D_t. D_p is used to guarantee the alignment of the pose of the generated image I_g with the target pose P_t. It takes two pairs of inputs, the target pose P_t concatenated with the generated image I_g as a fake input, and the same pose P_t concatenated with the target image I_t as a real input.

D_t is used to ensure the similarity of the appearance texture of the source image I_s with I_g. It also takes two input pairs, I_s concatenated with I_g as fake and the same I_s concatenated with I_t as real.

One of the main differences between RoomGAN and ADGAN, substitution of P_t by target layout L_t aside, is the use of the target image I_t. ADGAN uses ground truth images of the source person in the desired target pose P_t, while RoomGAN does not have that option since the datasets we are using do not contain images of the same room in different layouts or even from different perspectives. Therefore, our I_t is the image of the room from which we extract the layout. Adapting to this difference was one of the most challenging parts of our research.

We started by removing D_t, because unlike in the case of ADGAN, we could not use the discriminator to ensure the similarity of the textures since our I_t did not have the same appearance as the I_s.

The resulting structure of RoomGAN is presented in Figure 23.

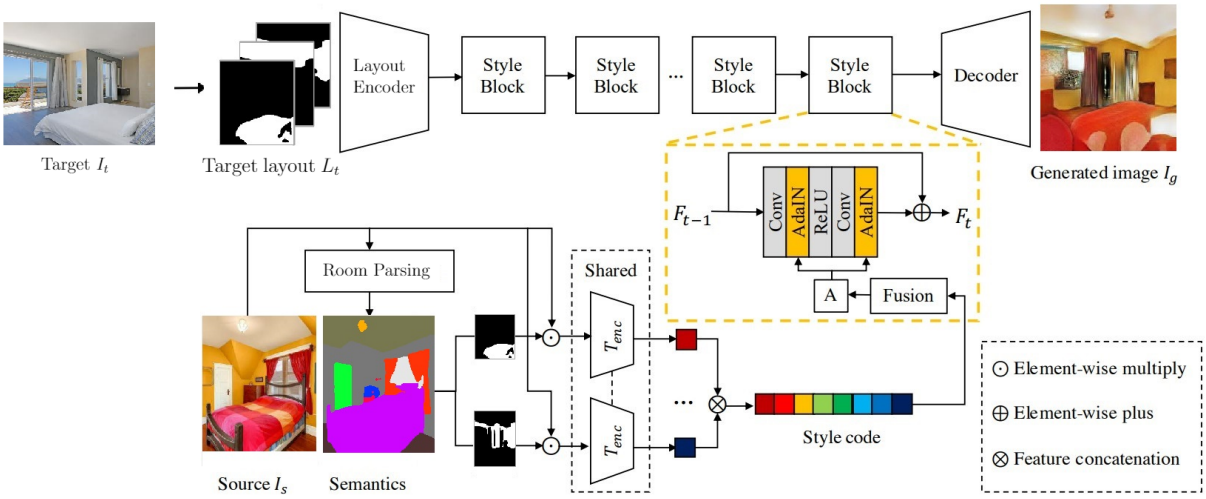


Figure 23: Structure of the RoomGAN method.

3.2.3 Losses

The training losses of the generator also suffered multiple changes. The model uses four different losses: Adversarial (1), Reconstruction (2), Perceptual (3) and Contextual (4).

$$L_{adv} = \mathbb{E}\left[\log\left(D_t\left(I_s, I_t\right) \cdot D_p\left(P_t, I_t\right)\right)\right] + \mathbb{E}\left[\log\left(\left(1 - D_t\left(I_s, I_g\right)\right) \cdot \left(1 - D_p\left(P_t, I_g\right)\right)\right)\right] \tag{1}$$

$$L_{rec} = \left\|I_g - I_t\right\|_1 \tag{2}$$

$$\mathcal{L}_{\text{per}} = \frac{1}{W_l H_l C_l} \sum_{x=1}^{W_l} \sum_{y=1}^{H_l} \sum_{z=1}^{C_l} \left\|\phi_l\left(I_g\right)_{x,y,z} - \phi_l\left(I_t\right)_{x,y,z}\right\|_1 \tag{3}$$

$$\mathcal{L}_{CX} = -\log\left(CX\left(\mathcal{F}^l\left(I_g\right), \mathcal{F}^l\left(I_t\right)\right)\right) \tag{4}$$

The adversarial loss tells the generator how real its outputs are perceived by the discriminators. The loss computes the binary cross-entropy between the outputs of the discriminators when fed with the real pairs and fake pairs. It increases if the generator can not fool the discriminator, forcing the former to improve the realism of its outputs. The reconstruction loss is the $L1$ distance between I_g and I_t. Those two losses were not modified, but the perceptual and contextual losses did.

The perceptual loss exploits deep features extracted from certain layers of the pretrained VGG network (used in the texture encoder, see Figure 19) for texture matching, which has been proven to be effective in image synthesis. It computes the $L1$ distance between the output features from layer 3 of the VGG (ϕ_3) for inputs I_g and I_t. Similar to what happened with the discriminator D_t, in the case of RoomGAN the I_t used does not have

the same textures as I_s, thus making the image useless in terms of texture matching. We decided to substitute I_t for I_s as one of the inputs, due to the fact that even though the alignment won't be the same, the network could still take advantage of the textures.

Since we could not use I_t the way it was originally intended, we decided to add a second kind of perceptual loss that would use I_g and I_t, but instead of using the layer 3 would use the layer 14. Shallower layers tend to extract high-level features such as colors, while deeper layers extract more complex features, like detailed shapes. Our goal was to use this second perceptual layer to better define the shapes of the objects in I_g.

The contextual loss [11] is designed to measure the similarity between two non-aligned images for image generation. It computes the logarithm of the similarity metric CX of the feature maps F^l extracted from layer l of the pretrained VGG. It does not require pixel-to-pixel alignment, thus allowing spatial deformations with respect to the target, getting less texture distortion and more reasonable outputs.

Since the loss does not penalize pixel-to-pixel dealignement as much as a traditional $L1$ loss, we decided to substitute again I_t for I_s, since the textures between I_g and I_s are the ones that should match.

Finally, we realized that for the training to make sense we had to select pairs of image that included the same elements, otherwise the model would try to learn to transfer textures of objects that are not present in the target image. We developed a script that classified each image according to the elements present on its segmentation (out of the 18 used to generate the layouts). That way images can only be paired if they have the same objects present, thus optimizing the training and avoiding possible misbehaviours.

3.3 MixNMatch

Our main goal with this model was to use it to perform a similar task to the RoomGAN model, which is performing attribute transfer throughout the different elements of a bedroom. By selecting the color from one of the elements of the source image, and the shape, pose and background of the same element from the target image, we wanted to check whether the MixNMatch method would perform better than the RoomGAN. MixNMatch can only act over one object at a time, so our first approach was to pick the most common element in a bedroom, the bed.

As explained in the State of the Art section, the MixNMatch model is trained in an unsupervised way, but it does require bounding boxes around the objects of interest to model the background. The bedroom datasets at our disposition do not contain bounding boxes, but it quite straightforward to transform the semantic segmentations into bounding boxes. To get the bounding boxes as accurate as possible, we decided to use the ADE20K dataset, which provides ground truth masks. To convert the semantic layout into a bounding box around the bed (Figure 24), we select only the pixels corresponding to the bed label. Then, from that set of pixels, we select the maximum and minimum positions values of X and Y, and get the corners of the bounding box according to the next criteria:

- Top right corner: (x_{max}, y_{max})
- Bottom right corner: (x_{max}, y_{min})
- Top left corner: (x_{min}, y_{max})
- Bottom left corner: (x_{min}, y_{min})

(a) Semantic layout.

(b) Mask corresponding to bed.

(c) Bounding box drawn around bed.

Figure 24: Conversion from semantic layout to bounding box.

Finally, we convert the corner positions into an XYWH notation (center of the bounding box and width and height of said box), which is the one used by the model.

3.4 Pix2Pix

The Pix2Pix model comes pretrained in a couple of different datasets, that allows the model to perform different Image-to-Image translation tasks (e.g. Figure 12).

The code also allows to train the model using different datasets. It accepts any kind of paired data A/B, such as label map/photo or BW image/color image, and can be trained to perform the translation in both directions (A to B or B to A). For our original purpose, we can use the model to translate semantic segmentations from existing rooms into new bedrooms that keep the same layout but have a different appearance.

Since this is a less conditioned model, we wanted to improve its performance at test time by using the LSUN synthesised dataset, which will allow the model to see significantly more images. The split will consist of 10,000 images for training, 1,000 for validation and 500 for testing.

4 Experiments and results

4.1 Implementation

In this section we are going to test and compare four different Conditional GAN models, that allow different levels of control over the generated output, from the one that allows the most control to least control: RoomGAN, MixNMatch, SPADE and Pix2Pix. We have chosen to work with the Pytorch implementations, which were luckily available for all the models. They all required an NVIDIA GPU and CUDA/CuDNN to work properly, plus some of them required some extra Python packages that will be detailed bellow.

Our very own code for the RoomGAN method is available at github: `https://github.com/imatge-upc/roomGAN`

4.2 Parameters

Ideally, we would use the exact same parameters for every model so we could compare them in all fairness. However, some of the implementations work different from others, specially in the case of MixNMatch, and do not allow to tweak every parameter. Apart from that, one of our main goals is to achieve the best possible results, so we finally decided to leave most of the parameters as default, since those are the ones that proved to work better for the original models, and adjust some of them as we thought necessary after analyzing the results of the experiments.

The exact parameters used to experiment with every model will be indicated in their respective subsections below.

4.3 Evaluation Metrics

It is quite difficult to pick any evaluation metric that would accurately represent how well any of the models perform with respect to the rest, specially because they all perform different levels of conditioned image generation. We could compare some of the common losses for most of them, such as the Adversarial loss, but that would not capture the real performance of the model.

Even though we will compare how well the models learn for this exact task by looking at the tendencies of their losses, the final evaluation and comparison of their performances will be subjective according to the quality of the final generated images.

4.4 RoomGAN

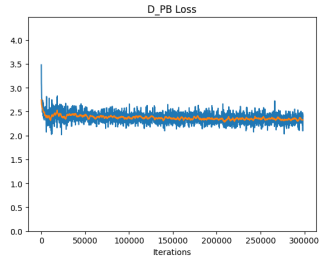
4.4.1 First experiment

The very first experiment we performed was also the first time we trained the model. We used the ADE20K Dataset, since in that moment we had not even began to synthesize the semantic segmentations for the LSUN Dataset. The RoomGAN model had not suffered yet all the changes described in the Methodology section. For instance, the perceptual loss was not divided into two different losses, and it still used I_t to improve the textures.

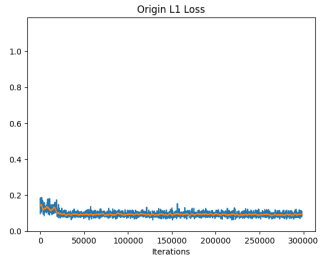
Initially we focused on finding the optimal λ for every training loss, so we ran the training phase for a day with each of the combinations that we thought would work best, adjusting them on the go. After a week, we decided to choose a fixed set of *lambdas* and let the model train for a prolonged period of time, while the rest of the hyperparameters were set as default. The values for this set of lambdas were:

- Lambda L1: 1.0
- Lambda Adversarial: 5.0
- Lambda Perceptual: 1.4
- Lambda Contextual: 3.0

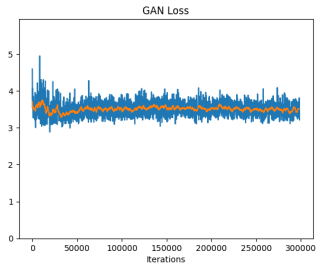
Most of the losses (Figure 25) seemed to decrease over time, specially the contextual. The generated images were promising, although far from realistic. In Figures 26(b), 26(d), 26(e) we can observe how the model properly transfers the predominantly colors of the ceiling, walls and floor. Occasionally it even manages to transfer more concrete textures, like in Figure 26(a). However, the model can not properly replicate some of the elements and it clearly fails in delivering realistic outputs.



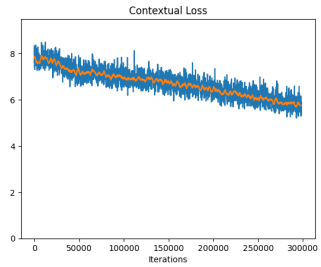
(a) D_p Loss



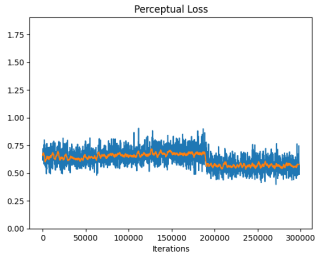
(b) L1 Loss



(c) Adversarial Loss



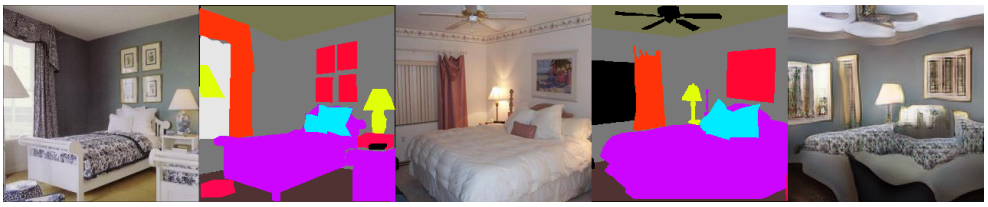
(d) Contextual Loss



(e) Perceptual Loss

Figure 25: Training losses for a batch size of 1. Blue lines correspond to the actual losses, while the orange lines are a smoother version.

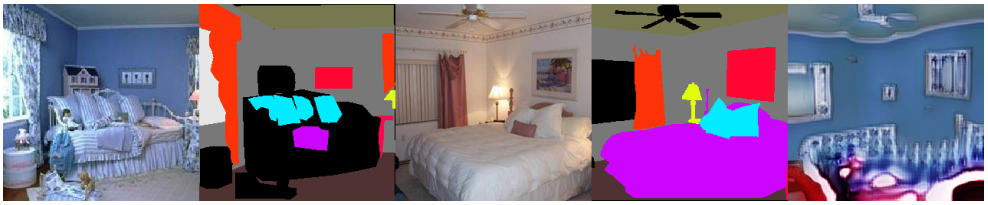
(a)



(b)

(c)

(d)



(e)

(f)

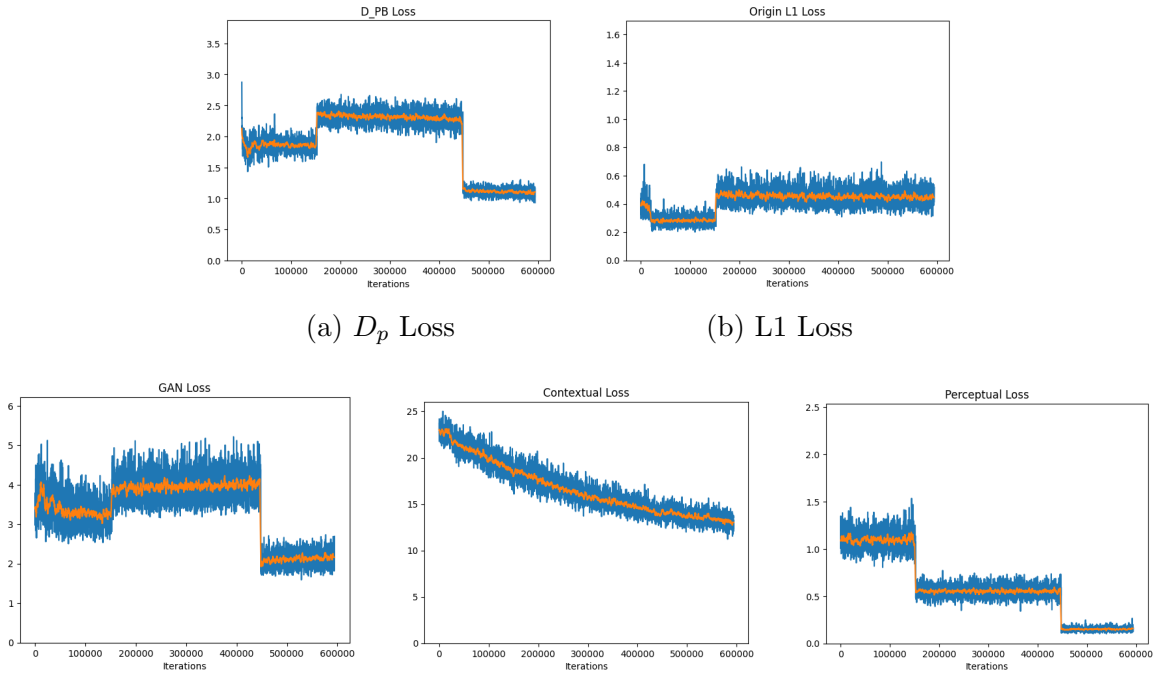
Figure 26: From left to right: source image, semantic segmentation of I_s, target image, semantic segmentation of I_t and generated output.

4.4.2 Second experiment

In the next experiment we trained the model without the discriminator D_p, but still with the ADE20K dataset. The contextual loss was also changed as described in the Methodology section.

Apart from this change, we also tried to find better values for lambdas, so we tried different combinations of them during the same training, which as we can see in Figure 27 caused some steep increases and decreases in the losses. We finally ended with the same combination used in the previous experiment, since it turned out to be the most effective one. In total, we trained the model for 217 epochs.

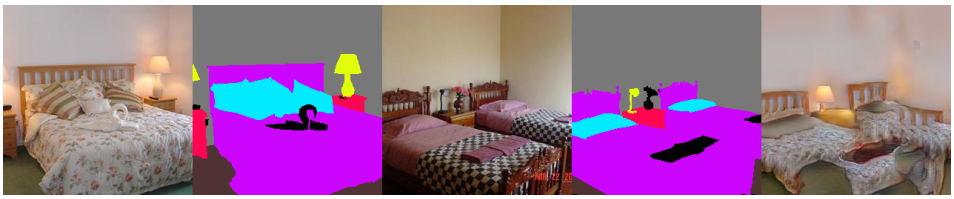
The contextual loss was the one that decreased the most while the other losses seemed to decrease slowly over time, except the Adversarial loss, which increased a little. We obtained better texture transfer than in the previous experiment and slightly more realistic images. The model did manage to transfer the lighting and shape of lamps (Figure 28(a-b)) and even the wooden headboard of the bed (Figure 28(a)).



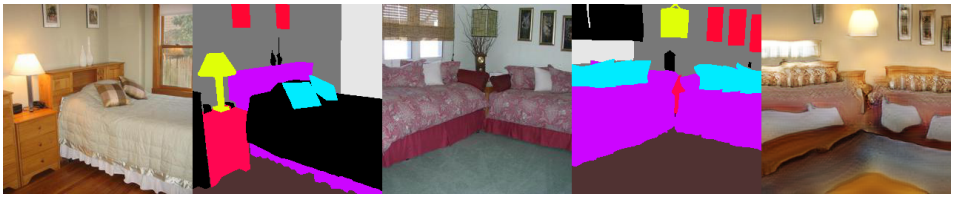
(a) D_p Loss (b) L1 Loss

(c) Adversarial Loss (d) Contextual Loss (e) Perceptual Loss

Figure 27: Training losses for a batch size of 4. Blue lines correspond to the actual losses, while the orange lines are a smoother version.



(a)



(b)

(c)

Figure 28: Examples of generated images at training time.

4.4.3 Third experiment

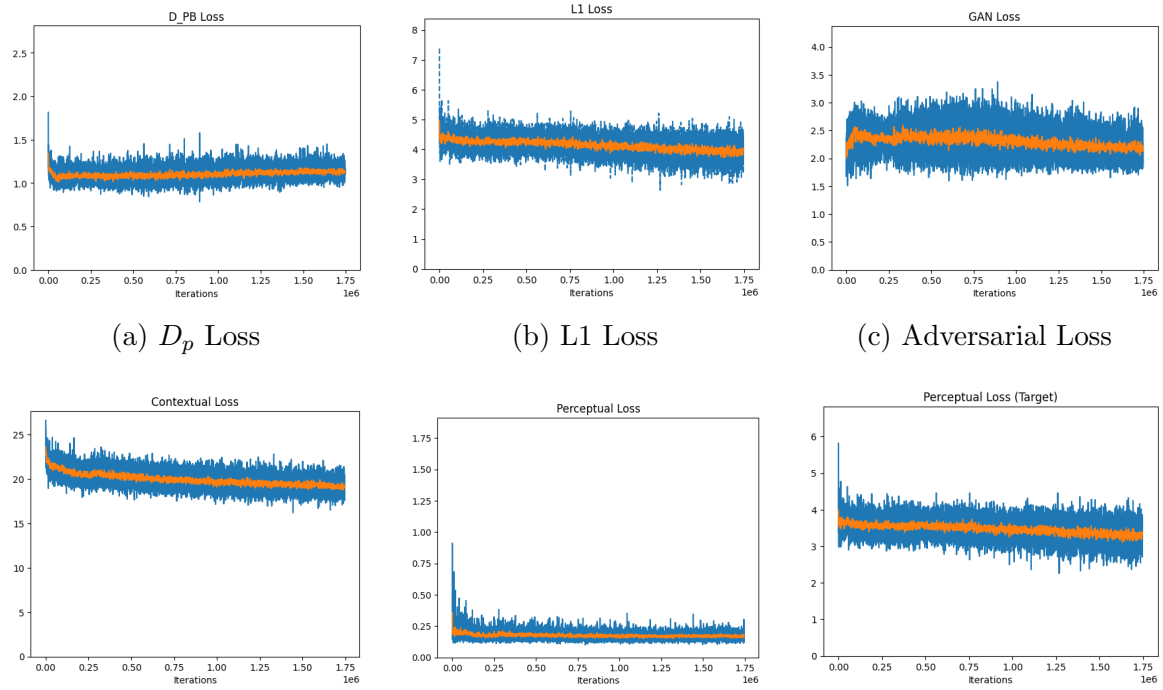
In order to improve the performance, specially at test time, we decided to use the much bigger LSUN dataset, maintaining the same parameters as the ones used in this last training phase. The training developed as usual, but around the 10 epoch mark the losses values turned to NaN. This also resulted in the model generating completely black images for every input. At this point, we had not performed the dataset cleanse described in the Methodology section. These results (Figure 29) were actually the reason we decided to perform said cleanse, since one of the most probable motives behind the losses turning NaN (due to a gradient explosion) could be the outliers we saw in some of the iterations (e.g. Figure 17-18).

Figure 29: The model only generates black images.

4.4.4 Fourth experiment

Finally, once the dataset was cleansed and we checked that the model no longer generated black outputs, we ran one last training phase that included all the changes described in the Methodology. The main difference with the last training, dataset apart, is the split of the perceptual loss.

In Figure 30 we can see how the $L1$, Contextual and the new perceptual loss using I_t all decrease, even though in a slower pace than in previous experiments. The Adversarial loss also decreases, which is a huge improvement from the last training phase in which the loss showed an increasing tendency. The discriminator and the percentual loss that uses I_s do not decrease, although the perceptual loss is already at very low values.



(a) D_p Loss (b) L1 Loss (c) Adversarial Loss

(d) Perceptual Loss (Target) (e) Perceptual Loss (Source) (f) Perceptual Loss (Target)

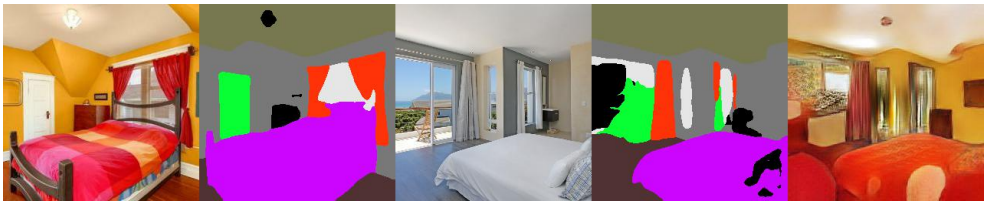
Figure 30: Training losses for a batch size of 4. Blue lines correspond to the actual losses, while the orange lines are a smoother version.

The results, shown in Figure 31, are promising. The model does exceptionally well on transferring the overall appearance as well as the predominant colors, specially those of the walls. Nonetheless, it fails to properly transfer the texture of some beds, specially if they contain complex patterns, and the general look of the outputs is not very realistic.

We noticed the importance of the quality of the semantic segmentation of the inputs in the quality of the result. In Figure 31(e) we can see how the lamp is present in the segmentation and the model is able to properly transfer it, even the fact that it is on. In Figure 31(f) however, the lamp is not present in the mask (black void) and therefore the model is unable to generate it in the result.

(a)

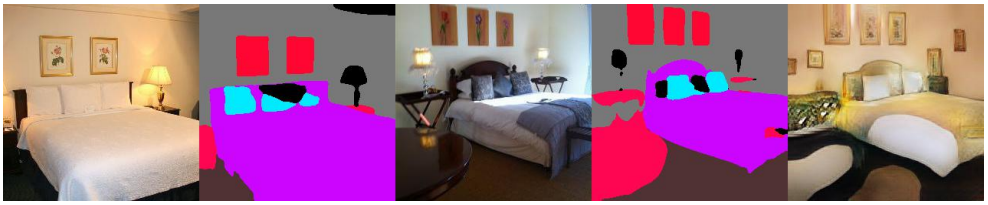
(b)



(c)

(d)

(e)



(f)

Figure 31: From left to right: source image, semantic segmentation of I_s, target image, semantic segmentation of I_t and generated output.

4.4.5 Inference Time

The average total inference time of our model is 22.8 seconds. The tests were run in the GPI computing server, which has the following specs:

- Ubuntu 18.04.
- 3 Intel CPUs.
- 10 GB of RAM memory.
- 2 NVIDIA Tesla V100 GPUs.

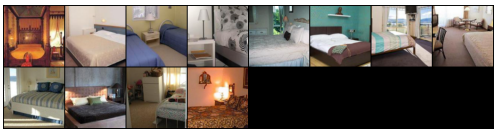
More detailed information about the inference time can be found in the thesis of Aleix [16].

4.5 MixNMatch

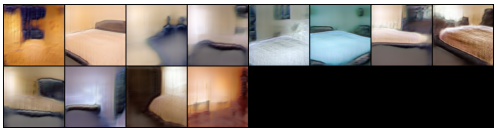
The first experiment we ran with this model uses the ADE20K dataset, consisting of aroung 1,000 images, which contains ground truth semantic segmentations, thus making the bounding boxes needed for the training extremely accurate.

The training phase consists of two stages. In the first stage the encoders learn to disentagle the features, while in the second stage the model focuses on extracting the shape and pose features. These stages are trained sequentially, and the user has control over the iterations that each of them perform. Since the implementation provided by the authors does not allow to retrain a stage once it has finished, our first goal was to find the maximum number of iterations for each stage that could be run withing the 24 hours limit on the GPI server. These values turned out to be 1,200 iterations for the first stage and 2,400 for the second one.

Another important hyperparameter to tune are the super categories and the fine categories. Super categories tell the model the number of expected different shapes that the network should try to learn, while fine categories are the different textures to expect. We selected these values to be 100 and 400 respectively. The rest of settings are left as default: an Adam optimizer with a learning rate of .0002 and a batch size of 12.

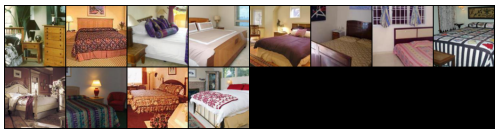


(a) Real images.



(b) Generated images.

Figure 32: Results on the first stage.



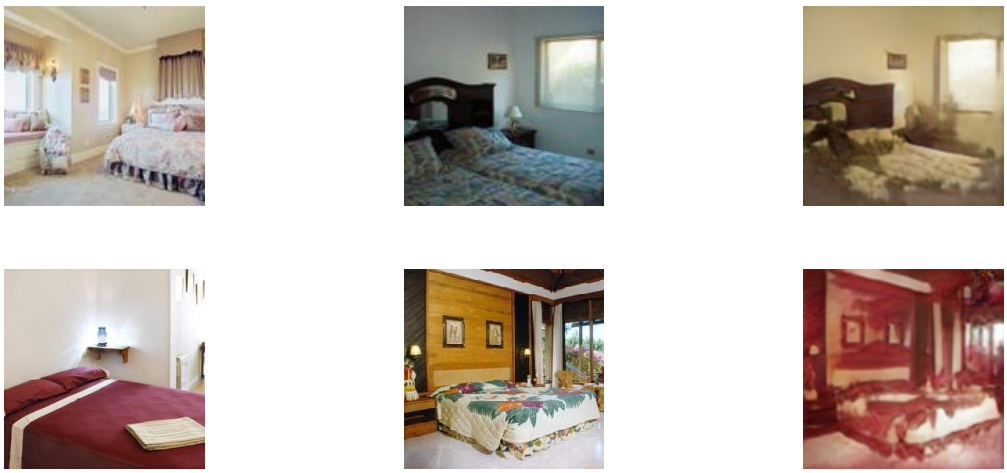
(a) Real images.

(b) Generated images.

Figure 33: Results on the second stage.

Both stages produce images (Figure 32-33) along the training, and although the model does not provide feedback on any training loss, we can use the generated images to follow the training process. The model outputs $128x128$ images.

To perform the attribute transfer task, we will select the texture from an image (source) and the pose, shape and background from another image (target). The goal is to obtain a picture that preserves the pose, shape and background of the bed in the target image, while the bed's texture has been swapped with the one of the source image. As we can see in Figure 34, the results are not promising.



(a) Source images (b) Target images (c) Generated images

Figure 34: MixNMatch results on the first experiment.

The model does great at keeping the pose, shape and background from the target images, but it transfers the bed's colors to the whole image rather than only to the bed.

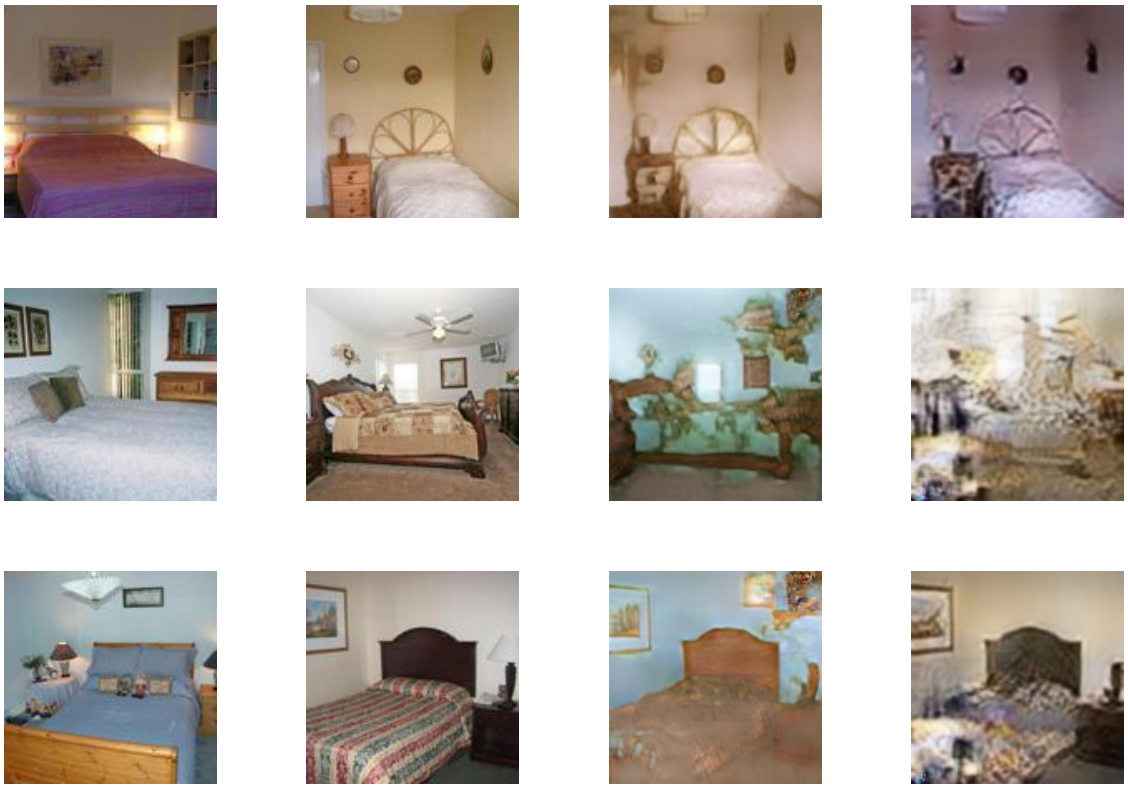
Since the main problem seemed to be that the network was not capable of identifying the object of interest, we decided to resize the whole dataset to the 128x128 format used in the output, to check whether or not that was causing this behaviour. The results were even worse. Compared to the model trained used 256x256 inputs (Figure 36), the network fails to retain the shape and pose and keeps being unable to transfer the texture to the object of interest only.

(a) Source image (b) Target image (c) 256x256 input (d) 128x128 input

Figure 35: Comparison between 256x256 and 128x128 input size.

Finally, we wanted to check whether feeding the model with a substantially bigger number of images could improve its performance. Instead of using the ADE20K dataset (around 1,000 images) we trained the model with the synthesized LSUN dataset, selecting only 50,000 images from the total available. To fit each stage in one day we had to decrease the number of iteration over ten times, 100 iterations for the first stage and 200 for the second.

As a reference, we will be comparing the results to the ones generated by the model when trained with an input of 256x256 since it proved to perform better than its 128x128 counterpart. In Figure 35 we can see how the model does not perform nearly as well. Although the previous training failed to deliver a proper texture transfer in the object of interest, it does much better at transferring the shape and background. The model trained with the LSUN dataset fails to acurrately recreate the shapes of the target image, let alone transfer any kind of realistic textures. The outliers and the less accurate masks failed to outweight the benefits of the increased number of images of the LSUN dataset.



(a) Source image (b) Target image (c) ADE20K (d) LSUN

Figure 36: Comparison between training with the ADE20K or the LSUN dataset.

We believe that the reason why the model could not perform as well as showcased in the paper [9] for other kinds of objects (birds, cars and dogs) is due to the fact that beds do not stand out against the background as much, and have similar shapes as other objects around them, thus making it harder for the model to disentangle its components from the rest of the image.

4.6 SPADE

The SPADE model is already pretrained in the whole ADE20K Indoor dataset. We wanted to test its performance on the subset of bedrooms from the very same ADE20K. This method takes as input a semantic segmentation, but it also accepts an optional style image, from which the model will try to extract a general appearance. This feature adds some extra control over the overall style of the generated output. However, it requires an attached encoder that is not provided in the official code, and therefore we won't test it.

The results, shown in Figure 37, are fantastic. Unlike RoomGAN the model generates every object defined in the segmentation, and does it with a high quality in terms of shapes and textures, giving the final result a much more realistic look.

(a) Ground truth (b) Semantic segmentation (c) Output

Figure 37: From left to right: real image, semantic segmentation of said image and synthesized image produced by SPADE using the segmentation as input.

4.7 Pix2Pix

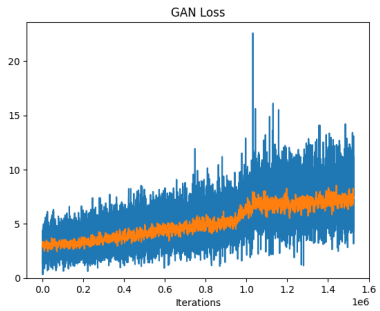
In the case of the Pix2Pix model, another reason to use the Pytorch implementation [4] was that it allegedly outperforms the original TensorFlow implementation.

Pix2Pix is the least conditioned model, in the sense that it does not allow any control over the style of the generated image. Therefore, it will be the model that is expected to perform better in terms of photo-realistic results, since it has less constrains to take into account.

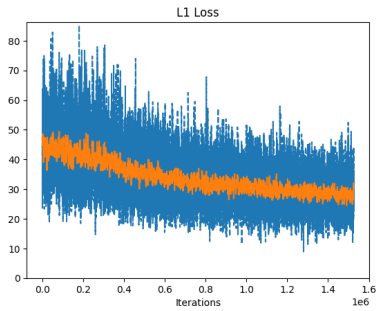
This model does provide feedback of the four losses it uses for training:

- Adversarial Loss
- $L1$ Loss
- Discriminator loss for real images
- Discriminator loss for fake images

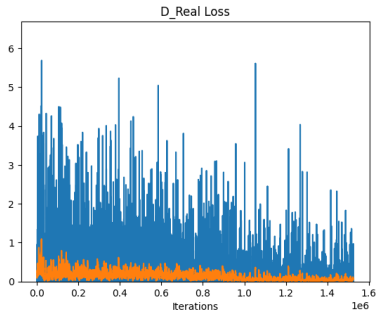
The first experiment will use the dataset and split mentioned in the methodology section. It will run using all the default options, including a batch size of 1. The model performs around 30 epochs per day, so it took a couple of days to get conclusive results. The model run for a total of 120 batches (around 1,500,000 iterations). As we can observer in Figure 38, all the training losses are quite noisy, but on average the tendency shows that the model is indeed learning, with the exception of the Adversarial loss.



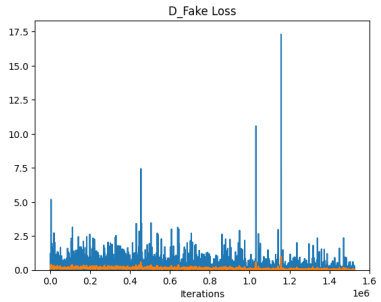
(a) Adversarial Loss



(b) L1 Loss



(c) Discriminator Loss (Real Images)

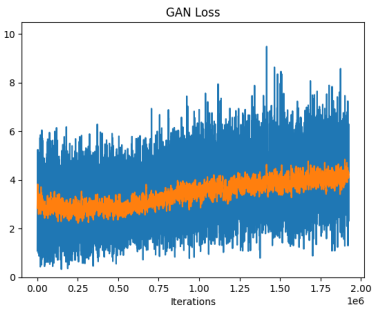


(d) Discriminator Loss (Fake Images)

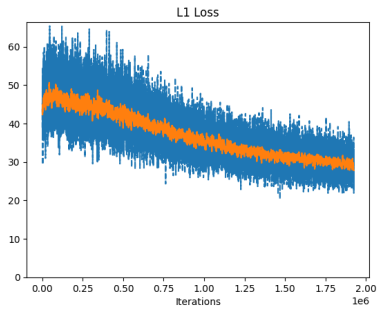
Figure 38: Training losses for a batch size of 1. Blue lines correspond to the actual losses, while the orange lines are a smoother version.

The Adversarial Loss tells the generator how plausible are the generated images to belong to the target domain. In other words, how 'real' the images seem. In this particular case, the fact that the loss is increasing could be due to the fact that some of the generated masks are not perfectly generated, or it could also be caused by some of the outliers in the dataset. Either way, one of the ways of reducing the impact of these isolated entries is to increase the batch size.

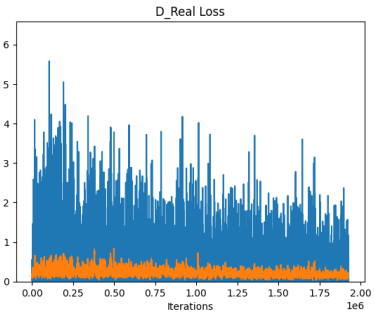
As a first approach, we increased the batch size to 4. As shown in Figure 39, the results are considerable better. The $L1$ loss continues to show a decreasing tendency, while both Discriminator losses only show a residual worsening. As for the Adversarial Loss, even though we did not manage to make it decrease, has clearly improved.



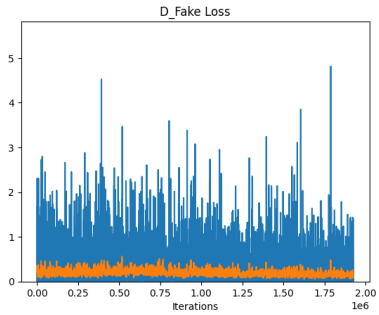
(a) Adversarial Loss



(b) L1 Loss



(c) Discriminator Loss (Real Images)

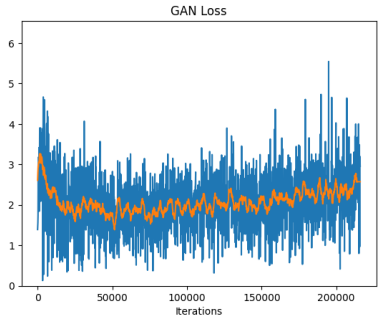


(d) Discriminator Loss (Fake Images)

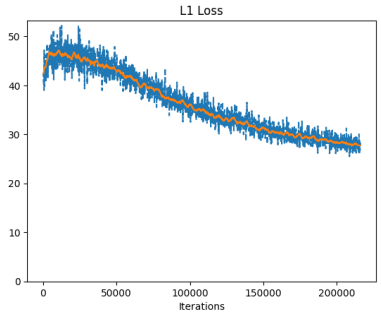
Figure 39: Training losses for a batch size of 4.

To further confirm this approach, we then increased the batch size to 32 and used two GPUs to perform the training. We gave the model the same training time (around 3 days), which resulted in less total iterations (around 200,000) due to the bigger batch size.

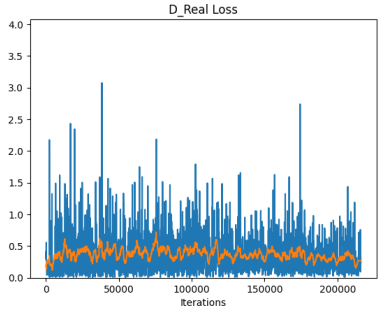
As expected, increasing the batch size considerably smoothens the losses (Figure 40). It does not have much effect on the Discriminator losses, but we can clearly appreciate an improvement on the Adversarial loss, which even decreases until the 50,000 iterations mark and does not spike after. The L1 loss does not suffer much improvement apart from a huge noise reduction.



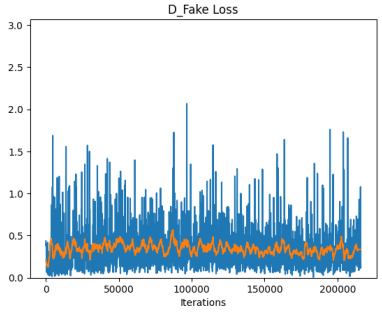
(a) Adversarial Loss



(b) L1 Loss



(c) Discriminator Loss (Real Images)

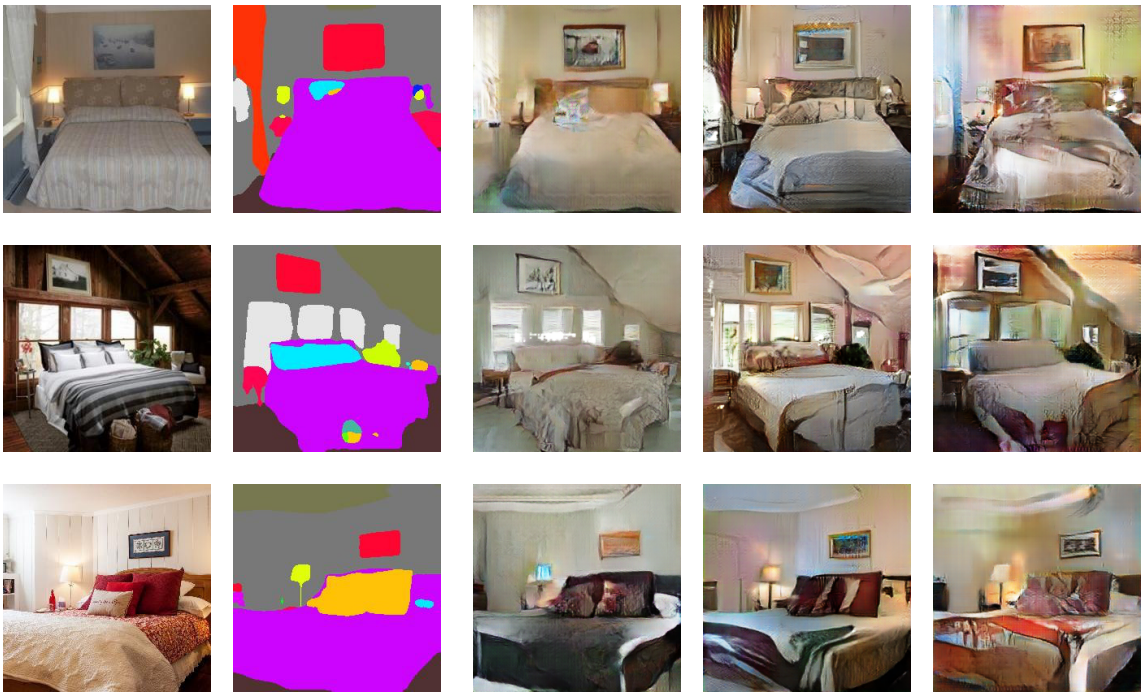


(d) Discriminator Loss (Fake Images)

Figure 40: Training losses for a batch size of 32.

These results confirm our hypothesis that the Adversarial loss was being heavily affected by some of the outliers in the dataset.

To compare how well the model performs when trained with the different batch sizes, we ran the method on 100 images from the test set. The results (Figure 41) are somewhat similar. The smaller batch sizes result in simpler images, with less complex textures and less defined shapes. However, the bigger batch sizes tend to apply too many textures resulting in a less realistic image. This is quite contradictory with our expectations on the Adversarial Loss, which saw an improvement when using bigger batch sizes.



(a) Ground truth (b) Input (c) Batch size 1 (d) Batch size 4 (e) Batch size 32

Figure 41: Comparison between different batch sizes.

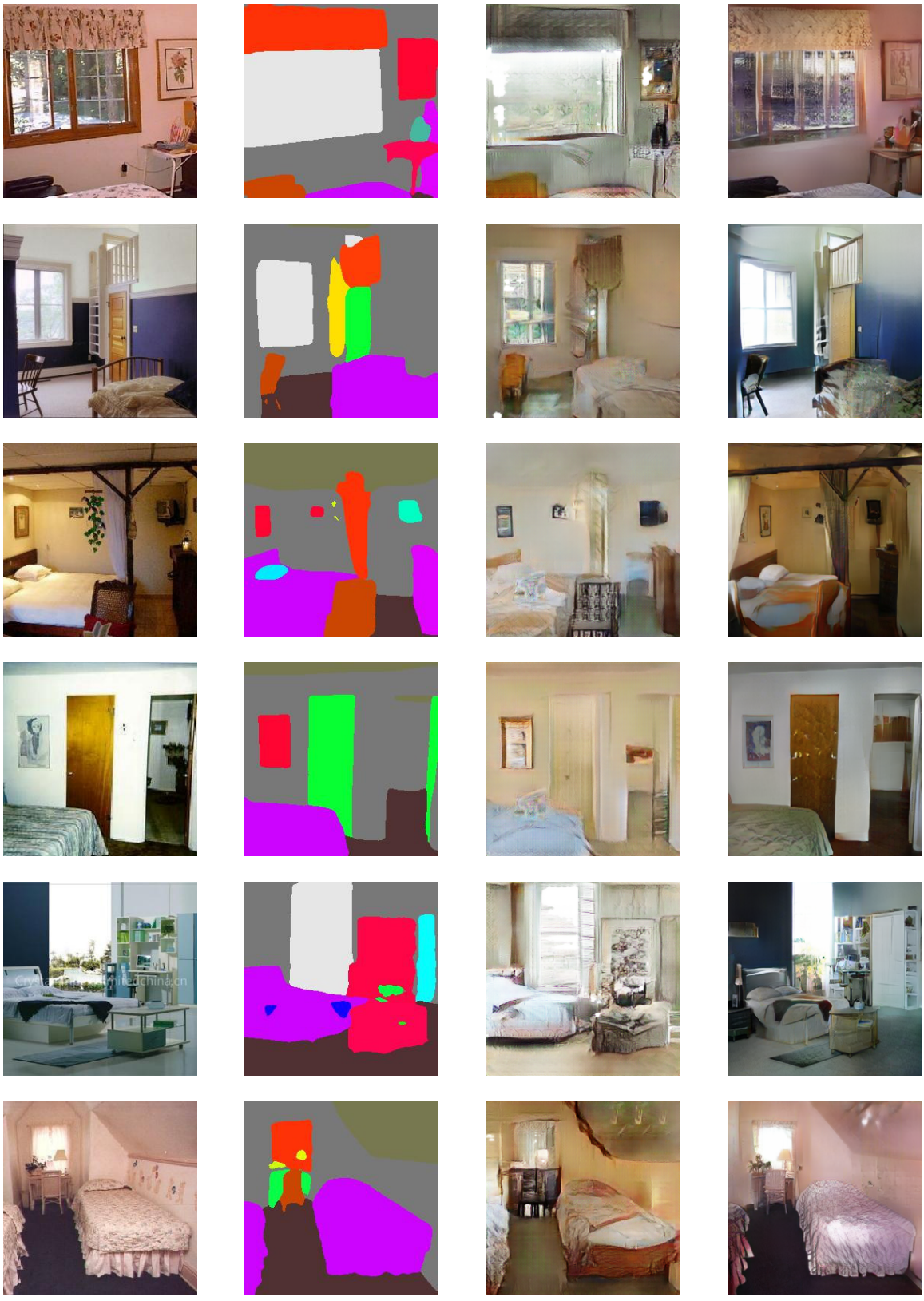
In the end, results show that using a moderate batch size ends up producing more realistic results, even though the training losses would suggest that increasing the batch size does not have any drawbacks.

4.8 Comparison

As mentioned in the beginning of this section, it is certainly difficult to establish a fair way to compare the four models discussed. There is not an evaluation metrics that could be used, and even if it was, it would not be fair to compare RoomGAN, which is a highly conditioned method, to SPADE or Pix2Pix, which perform simpler tasks.

However, we can try to give a subjective comparison of the results. We will leave MixN-Match out of the comparison since it did not prove to achieve any significant results.

We will start by comparing SPADE and Pix2Pix (trained with a batch size of 4 as it showed the best performance) since they perform the same task of translating a semantic segmentation into an image. The main difference, architectures aside, is that SPADE is trained using the whole ADE20K Indoor dataset, that contains images of different indoor rooms, while we trained Pix2Pix using bedroom images exclusively. This fact could lead us to think that Pix2Pix would perform better on test time when given only bedroom segmentations, but the results point otherwise.



(a) Ground truth (b) Segmentation (c) Pix2Pix (d) SPADE

Figure 42: Comparison between Pix2Pix and SPADE.

In Figure 42 we can appreciate the difference between the generated images both model produce when fed the same input. Although Pix2Pix manages to produce quite realistic results, the style of most images tends to resemble that of a painting. And even though Pix2Pix manages to properly synthesize every object present in the output, it fails to delimit their shapes as nicely as SPADE. Overall, both models perform well for the task of translating semantic segmentations to images, but SPADE clearly outperforms Pix2Pix, despite the fact that the second was trained specifically for bedrooms.

Finally, we also wanted to compare the performance of RoomGAN. We can not make a fair straight comparison with the rest of the models, since RoomGAN performs a much more complex task, but we want to compare the quality of the generated images to give a reference of how far RoomGAN positions from other state of the art methods.

In Figure 43 we wanted to show some of the outputs that better represent the biggest flaw of RoomGAN. It achieves good results in the task of properly transfering the style of the image as well as the textures of walls, floors, ceiling and in general big objects with simple textures. However, it struggles with smaller objects that have complex textures, as it is often the case with beds. Overall, it produces pleasant results, but far away from being as realistic as the outputs of methods like Pix2Pix and SPADE.

Figure 43: From left to right: source image, semantic segmentation of I_s, target image, semantic segmentation of I_t and generated output.

5 Conclusions

The aim of this thesis was to research the state of the art Generative Adversarial Networks and their applications on Attribute Transferring, with our final goal being the development of a method capable of transferring the shapes and textures of a certain set of objects from one image to another, and apply that method to create an app that would allow the user to change the visual style of their own bedroom.

We began by researching the state of the art methods used to perform similar tasks in hopes of finding a model that could be used as the foundation for our own method, and we succeeded by picking ADGAN, which has proven to serve as the base on top of which we have developed RoomGAN.

RoomGAN performs well on the task of transfering the shape, pose and textures of a bedroom's objects over another image, but it sadly fails in delivering photo-realistic results, specially compared to those of the state of the art models researched.

From the very beginning we always had in mind the development of an app in which deploy our model, and we successfully managed to adapt the model so it could be used as part of an end-to-end application with a low delay of around 20 seconds.

Out of our six original objectives we have managed to successfully complete five of them.

5.1 Future Work

The next steps we want to take involve improving the model performance and the app, so as to make it more viable for deployment and daily use:

- Find a new discriminator capable of improving the realism of the generated images, which fits the available data that we already have.
- Reduce even further the delay of the end-to-end process.
- Improve the quality of the synthesized dataset by finding a better segmentator or look up for a new dataset with ground truth segmentations and a big volume of images.

References

[1] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields, 2017.

[2] Google. Gan structure, 2021. URL `https://developers.google.com/machine-learning/gan/gan_structure`.

[3] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation in pytorch. *Available at* `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`, 2018.

[5] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[6] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proc. CVPR*, 2020.

[7] Chen-Yu Lee, Vijay Badrinarayanan, Tomasz Malisiewicz, and Andrew Rabinovich. Roomnet: End-to-end room layout estimation, 2017.

[8] Kathleen M Lewis, Srivatsan Varadharajan, and Ira Kemelmacher-Shlizerman. Vogue: Try-on by stylegan interpolation optimization, 2021.

[9] Yuheng Li, Krishna Kumar Singh, Utkarsh Ojha, and Yong Jae Lee. Mixnmatch: Multifactor disentanglement and encoding for conditional image generation, 2020.

[10] Machine Learning Mastery. Max pooling vs averaga pooling, 2020. URL `https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks`.

[11] Roey Mechrez, Itamar Talmi, and Lihi Zelnik-Manor. The contextual loss for image transformation with non-aligned data, 2018.

[12] Yifang Men, Yiming Mao, Yuning Jiang, Wei-Ying Ma, and Zhouhui Lian. Controllable person image synthesis with attribute-decomposed gan. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[13] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[14] Towards Data Science. Example of a classic convolutional neural network structure, 2021. URL `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks`.

[15] Towards Data Science. Convolutional operation using a 3x3 kernel, 2021. URL `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks`.

[16] Aleix Clemens Suriol. Attribute transfer in image generation using conditional gans. Final Degree Thesis, UPC.

[17] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop, 2016.

[18] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[19] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset, 2018.