DISCOVERING GAN: DIFFERENT APPROACHES

DLAI - E.RODOLÀ

Andrea Magnante - 1876801

Abstract. — The following study focuses on GAN approach (presented for the first time in 2014), used for the generation of new images. In the first chapter we will present the mathematical aspects behind this technique, while in the second we will present more concretely how GANs work. Presenting also the qualities that these have brought but at the same time those that are the problems from which they suffer. We will then see several of the various types of GANs (DCGAN, LSGAN, WGAN CDCGAN) presented so far and the changes from one to the other. Finally, starting from a chosen use-case, Lego Minifigures, we will see how these will behave in the creation of new samples, bringing what are the results of the experiments carried out.

Contents

1. Generative Adversarial Network

Generative Adversarial Networks (GANs) proposed by [*Goodfellow et al. 2014*] are based on two dueling networks, Generator G and Discriminator D (fig.1). Informally, the main idea consists of a game between two players where D has the role to try to distinguish between the generated sample of G and the ground truth, while at the same time G tries to fool D with the production of fake instance samples very similar to the real ones. Competition in this game drives both teams to improve their methods until the counterfeits are indistiguishable from the genuine articles.

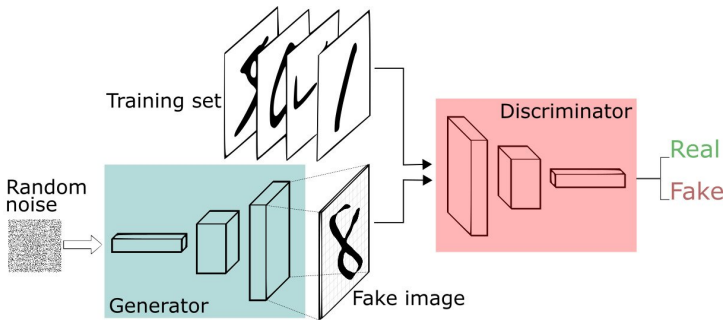


FIGURE 1. Generator vs Discriminator

More in depth, having:
— x: sample from the real distribution
— x': $D_\gamma(z)$: generated sample, good if $p_g(x') \approx p_g(x)$

$$\underbrace{\mathbb{E}_{\mathbf{x}} \log \Delta_{\boldsymbol{\delta}}(\mathbf{x})}_{\text{real data}} + \underbrace{\mathbb{E}_{\mathbf{z}} \log(1 - \Delta_{\boldsymbol{\delta}}(D_{\boldsymbol{\gamma}}(\mathbf{z})))}_{\text{fake data}}$$

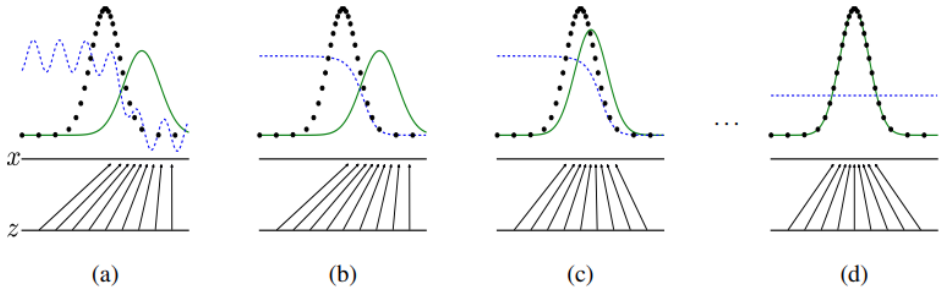
A good discriminator should yield $\Delta_\delta(x) \approx 1$ on the real instances and $\Delta_\delta(x') \approx 0$ on the fake instances. Train a classifier to distinguish between generated and real data:

$$\max_{\boldsymbol{\delta}} \; \mathbb{E}_{\mathbf{x}} \log \Delta_{\boldsymbol{\delta}}(\mathbf{x}) \; + \; \mathbb{E}_{\mathbf{z}} \log(1 - \Delta_{\boldsymbol{\delta}}(D_{\boldsymbol{\gamma}}(\mathbf{z})))$$

In contrast, the generator should make this value as small as possible:

$$\min_{\boldsymbol{\gamma}} \max_{\boldsymbol{\delta}} \; \mathbb{E}_{\mathbf{x}} \log \Delta_{\boldsymbol{\delta}}(\mathbf{x}) \; + \; \mathbb{E}_{\mathbf{z}} \log(1 - \Delta_{\boldsymbol{\delta}}(D_{\boldsymbol{\gamma}}(\mathbf{z})))$$

The generator competes against the adversarial discriminator and tries to minimize its success rate.



In the figure above we can see how the discriminator (blue line) and the generator (green line) behave during the training phase with respect to the data generating distribution (black dotted line). (a) Consider an adversarial pair near convergence: p_g is similar to p_d and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to the optimal Discriminator.
(c) After an update to G, gradient of D has guided G(z) to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_d$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$.

1.1. Global optimality of p_g=p_d. — This minimax game has a global optimum for $p_g = p_d$. First consider the optimal discriminator D for any given generator G.

Proposition 1. For G fixed, the optimal discriminator D is

$$\Delta_\delta(x) = \frac{p_d(x)}{p_d(x) + p_g(x)}$$

Plugging it back in the main functional:

$$J(G) = \mathbb{E}_{\mathbf{x}\sim p_d} \log \frac{p_d(\mathbf{x})}{p_d(\mathbf{x}) + p_g(\mathbf{x})} + \mathbb{E}_{\mathbf{x}\sim p_g} \log \frac{p_g(\mathbf{x})}{p_d(\mathbf{x}) + p_g(\mathbf{x})}$$

Let now define a new distribution $\rho = \frac{1}{2}p_d + \frac{1}{2}p_g$, it's like the arithmetic mean of the two distributions p_g and p_d and this applies to each point x. From this we get:

$$J(G) \propto \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_d} \log \frac{p_d(\mathbf{x})}{\rho(\mathbf{x})} + \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_g} \log \frac{p_g(\mathbf{x})}{\rho(\mathbf{x})} + \text{const.}$$

This transformation allows us to notice that the resulting expression is nothing more than the Kullback-Leibler divergence between $\mathrm{p_d}||\rho$ and $\mathrm{p_g}||\rho$.

$$= \frac{1}{2}KL(p_d\|\rho) + \frac{1}{2}KL(p_g\|\rho) + \text{const.}$$

Therefore, the optimal GAN generator is found by minimizing:

$$\min_{p_g} \ \frac{1}{2}KL(p_d\|\rho) + \frac{1}{2}KL(p_g\|\rho) + \text{const.}$$

But the previous expression can be rewritten in the form of Jensen–Shannon divergence between two distributions (measuring the distance between them), whose result is always non-negative except zero when they are equal; and the resulting property

$$\min_{p_g} \ \underbrace{\frac{1}{2}KL(p_d\|\rho) + \frac{1}{2}KL(p_g\|\rho)}_{JS(p_d\|p_g)}$$

$$p_d = p_g \ \Leftrightarrow \ JS(p_d\|p_g) = 0$$

This property is important because we know that every global optimizer for this problem, minimizing the JS divergence, will ensure that $\mathrm{p_g} = \mathrm{p_d}$, and that's what we want. And hence we can state that: within the GAN paradigm, the globally optimal generator has a data distribution exactly equal to the real distribution of the data.

2. Generate Data using GAN

A GAN is a type of neural network that is able to generate new data from scratch. You can feed it a little bit of random noise as input, and it can produce realistic images of bedrooms, or birds, or whatever it is trained to generate.

When training begins, the generator produces obviously fake data:

As training progresses, the generator gets closer to producing output that can fool the discriminator.

Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases:

Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

2.1. Discriminator. — The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying.

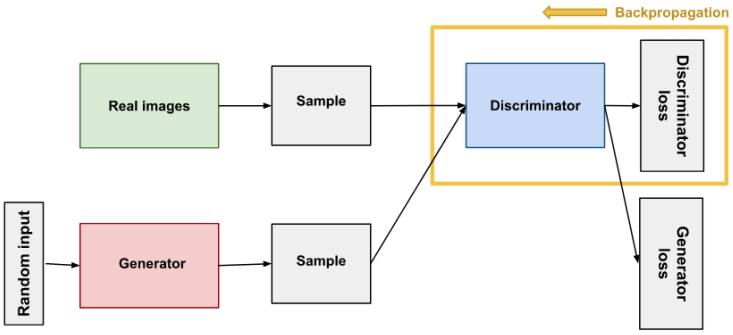


FIGURE 2. Discriminator

The discriminator's training data comes from two sources:

— Real data instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
— Fake data instances created by the generator. The discriminator uses these instances as negative examples during training

The discriminator connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss. During discriminator training:
— The discriminator classifies both real data and fake data from the generator.
— The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
— The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

2.2. Generator. — The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real.

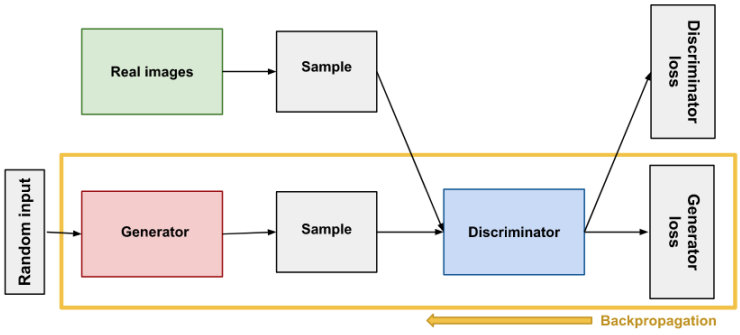


FIGURE 3. Generator

But what do we use as input for a network that outputs entirely new data instances?

In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, we can get the GAN to produce a wide variety of data, sampling from different places in the target distribution.

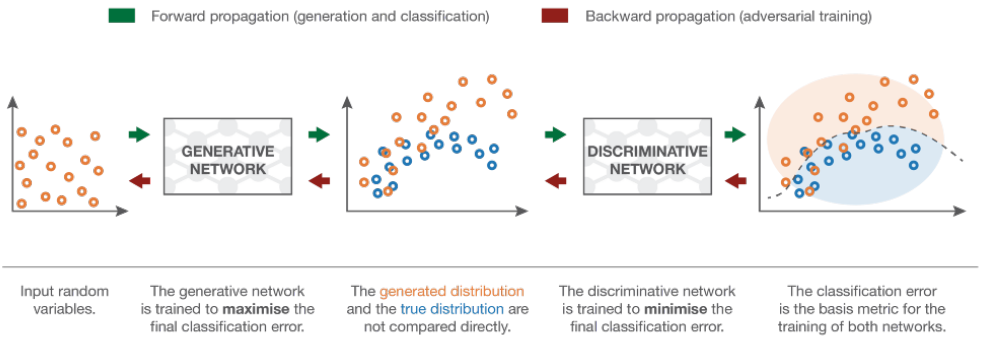
To train a neural net, we alter the net's weights to reduce the error or loss of its output. In our GAN, however, the generator is not directly connected to the loss that we're trying to affect. The generator feeds into the discriminator net, and the discriminator produces the output we're trying to affect. The generator

loss penalizes the generator for producing a sample that the discriminator network classifies as fake.

This extra chunk of network must be included in backpropagation. Backpropagation adjusts each weight in the right direction by calculating the weight's impact on the output. Impact of a generator weight depends on the impact of the discriminator weights it feeds into. Backpropagation starts at the output and flows back through the discriminator into the generator.

So the generator is trained with the following procedure:
— sample random noise
— produce generator output from sampled random noise
— get discriminator "Real" or "Fake" classification for generator output
— calculate loss from discriminator classification
— backpropagate through discriminator and generator to obtain gradients
— use gradients to change only the generator weights

2.3. GAN training. — The generator and the discriminator have different training processes (alternating training). We keep the generator constant during the discriminator training phase. As discriminator training tries to figure out how to distinguish real data from fake, it has to learn how to recognize the generator's flaws.
Similarly, we keep the discriminator constant during the generator training phase. Otherwise the generator would be trying to hit a moving target and might never converge. It's this back and forth that allows GANs to tackle otherwise intractable generative problems.



After several steps of training, if the Generator and Discriminator have enough capacity (if the networks can approximate the objective functions), they will reach a point at which both cannot improve anymore. At this point, the generator generates realistic synthetic data, and the discriminator is unable

to differentiate between the two types of input.
Since during training both the Discriminator and Generator are trying to optimize opposite loss functions, they can be thought of two agents playing a minimax game with value function V(G,D). In this minimax game, the discriminator is trying to maximize D(x), classify right the real data, and minimize D(G(z)), not the false ones; while the generator tries to maximize D(G(x)), making fake image classified as real.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

2.4. Loss function. — GANs try to replicate a probability distribution. Loss functions should be used to reflect the distance between the distribution of the data generated by the GAN and the distribution of the real data.
A GAN can have two loss functions: one for generator training and one for discriminator training.
The generator, as we said before, is not trained directly and instead is trained via the discriminator model. Specifically, the discriminator is learned to provide the loss function for the generator.
In the loss schemes, the generator and discriminator losses derive from a single measure of distance between probability distributions.

How do you capture the difference between two distributions in GAN loss functions? This question is an area of active research, and many approaches have been proposed like minimax loss (presented above), Wasserstein loss, etc.

3. GAN common problems

Why it's so hard to train GAN? Although GAN has shown great success in the realistic image generation, the training is not easy; The process is known to be slow and unstable. So let's see what are the major problems that afflict GAN models.

Non-Convergence. As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy (flips a coin). This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse.
For a GAN, convergence is often a fleeting, rather than stable, state.

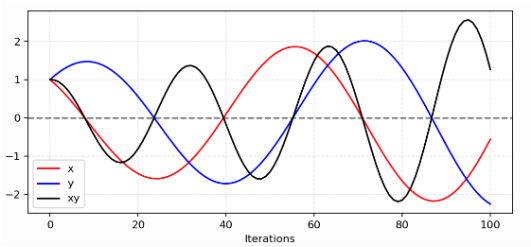
Hard to achieve Nash equilibrium. GAN is based on the zero-sum non-cooperative game (also called minimax); in short, if one wins, one loses and viceversa. GAN model converges when the discriminator and the generator reach a Nash equilibrium.

"A Nash Equilibrium is a set of strategies, one for each player, such that no player has incentive to change his strategies given what the other players are doing."

Also in this case, each model want to win and updates its cost independently with no respect to the other. Updating the gradient of both models concurrently cannot guarantee a convergence.
We can see an example taken from a paper which explain why it's difficult to find the Nash Equilibrium in a non-cooperative game.

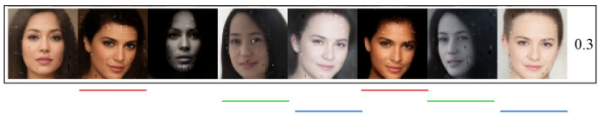
Example. Suppose one player takes control of x to minimize $f_1(x) = xy$, while at the same time the other player constantly updates y to minimize $f_2(y) =$ -xy. Because $\frac{f}{x} =$ y and $\frac{f}{y} =$ -x, we update x with x - * y and y with y + * x simultanously in one iteration, where is the learning rate. Once x and y have different signs, every following gradient update causes huge oscillation and the instability gets worse in time, as shown in figure below.

Vanishing Gradients. Research has suggested that if your discriminator is too good, then generator training can fail due to vanishing gradients. During backpropagation, gradient flows backward, from the final layer to the first layer.



As it flows backward, it gets increasingly smaller. Sometimes, the gradient is so small that the initial layers learn very slowly or stop learning completely. In this case, the gradient doesn't change the weight values of the initial layers at all, so the training of the initial layers in the network is effectively stopped. When the discriminator is perfect, we are guaranteed with $D(x)=1$, $\forall x \in p_d$ and $D(x)=0$, $\forall x \in p_g$. Therefore the loss function L falls to zero and we end up with no gradient to update the loss during learning iterations.

Mode Collapse. Mode collapse is one of the hardest problems to solve in GAN. A complete collapse is not common but a partial collapse happens often. The images below with the same underlined color look similar and the mode starts collapsing.



Usually you want your GAN to produce a wide variety of outputs. However, if a generator produces an especially plausible output, the generator may learn to produce only that output. In fact, the generator is always trying to find the one output that seems most plausible to the discriminator. If the next generation of discriminator gets stuck in a local minimum and doesn't find the best strategy. Each iteration of generator over-optimizes for a particular discriminator, and the discriminator never manages to learn its way out of the trap. It is also important to remember how the use of activation functions can lead to this type of problem more easily, see Sigmoid and Tanh, compared to ReLU, LeakyReLU.

Lack of a proper evaluation metric. GANs are not born with a good objection function that can inform us the training progress. Without a good evaluation metric, it is like working in the dark. Loss of discriminator and generator can not indicate the performance of generated samples. No good sign to tell when to stop or indicator to compare the performance of multiple models.

4. Evolution of GANs

Since 2014, when the first paper on gan was written, several GANs have been developed. In this study we are going to describe the motivations of finding new techniques to improve them. What changes and what are the pros/cons, going into the details of each GAN, presenting the architecture and development.

DCGAN

Deep Convolutional Generative Adversarial Network is one of the popular and successful network design for GAN. DCGANs are the 'image version' of the most fundamental implementation of GANs. It's a direct extension of the GAN, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively.

Defining the networks. The DCGAN architecture provides the use of a CNN architecture standard for the discriminative model. While for the generator, upconvolutions replace convolutions, so the representation at each layer of the generator is actually successively larger.

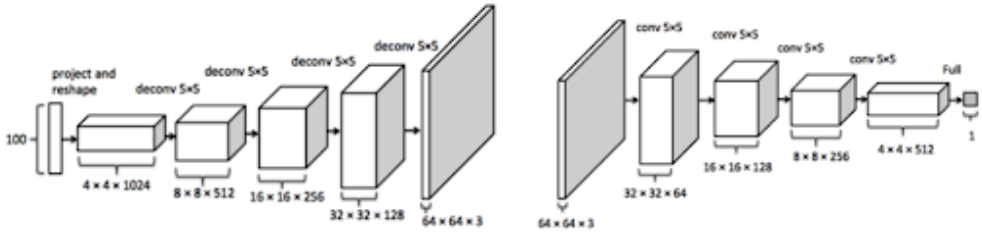


FIGURE 4. DCGAN architecture

The discriminator is made up of strided convolution layers, batch norm layers, and LeakyReLU activations. The input is a 3x64x64 input image and the output is a scalar probability that the input is from the real data distribution. For the discriminator, the last convolution layer is flattened and then fed into a single sigmoid output.
The generator is comprised of convolutional-transpose layers, batch norm layers, and ReLU activations. The input is a latent vector, z, that is drawn from a standard normal distribution and the output is a 3x64x64 RGB image. The strided conv-transpose layers allow the latent vector to be transformed into a volume with the same shape as an image.

In addition, the author of the paper on DCGANs gave some advice on how to set optimizers, calculation of loss functions and model weights initialization. Details from the paper: *No pre-processing was applied to training images besides scaling to the range of the tanh activation function [-1, 1]. All models were trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128. All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02. In the LeakyReLU, the slope of the leak was set to 0.2 in all models. While previous GAN work has used momentum to accelerate training, we used the Adam optimizer with tuned hyperparameters. We found the suggested learning rate of 0.001, to be too high, using 0.0002 instead. Additionally, we found leaving the momentum term 1 at the suggested value of 0.9 resulted in training oscillation and instability.*

Loss functions and Optimizers. For DCGAN we will use the Binary Cross Entropy loss (BCELoss) function which is defined in Pytorch as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right]$$

where N is the batch size and note that the targets $0 \leq y \leq 1$. This function provides the calculation of both log components, log(D(x)) and log(1-D(G(x))) and it measures the binary cross-entropy between the target and the output. We define also real labels (1) and fake labels (0), used when calculating the losses of D and G (as done in original GAN paper).
Then we set up the optimizers, one for D and one for G. As specified befor, both are Adam optimizers with learning rate 0.0002 and Beta1 = 0.5.

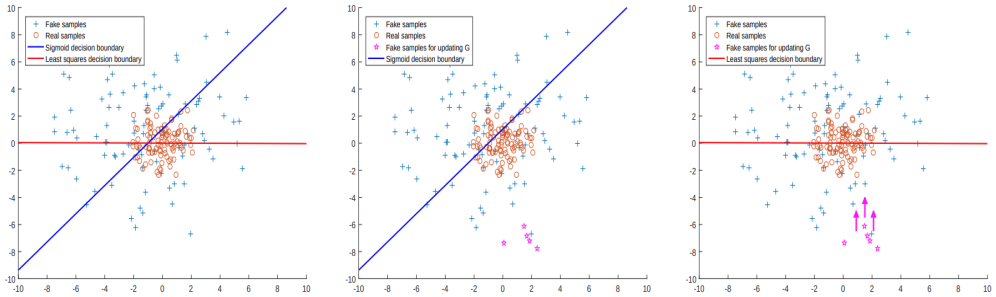
The novelties introduced. Changes affected by DCGAN are:
— Replace all max pooling with convolutional stride
— Use transposed convolution for upsampling.
— Eliminate fully connected layers.
— Use Batch normalization except the output layer for the generator and the input layer of the discriminator.
— Use ReLU in the generator except for the output, which uses tanh.
— Use LeakyReLU in the discriminator.

LSGAN

In regular GANs the discriminator is related to the sigmoid cross entropy loss function, which may lead to the vanishing gradients problem during the learning process. Least Squares Generative Adversarial Networks is designed to solve this problem. The idea is to adopt the least squares loss function for the discriminator.
There are two benefits using LSGANs instead regular GANs:

— generation of higher quality images
— perform more stable during the learning process

The main problem that this technique intends to solve is certainly the possibility of improving the fake samples that the generator creates. In the generic case, in fact, the sigmoid cross entropy loss function will lead to the problem of vanishing gradients; fake samples will be in the correct side of the decision boundary but they will still be far from being similar to the real ones (figure a, b). The idea is therefore to place them closer to real samples, using the least squares loss function. It's able to move the fake samples toward the decision boundary and it penalizes samples that lie in a long way on the correct side of the decision boundary (figure c).



(a) (b) (c)

The least squares loss function will penalize the fake samples and pull them toward the decision boundary even though they are correctly classified. This allows to state that LSGAN manages to generate examples more similar to the real ones.

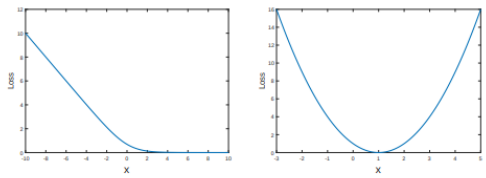
Defining the networks. The paper's authors propose a generator and discriminator model architecture and use interleaving upsampling and normal convolutional layers in the generator model.

Loss functions and Optimizers. LSGAN wants to use loss function that provides smooth and non-saturating gradient in discriminator D. We want D to pull data generated by generator G towards the real data manifold, so that G generates data that are similar to the real ones. So what we need is to avoid sigmoid function in discriminator and instead we work with:

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x})-b)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-a)^2\right]$$
$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-c)^2\right],$$

Let's outline the modifications done by LSGAN to the original GAN:
— Remove log from D
— Use L_2 loss instead of log loss

The novelties introduced. Benefits introduced by LSGAN are:
— penalization of samples that, despite having been correctly classified, are far from the real distribution. This allows the generator to generate samples that follow the decision boundary.
— penalizing the samples lying a long way to the decision boundary can generate more gradients when updating the generator, which in turn relieves the problem of vanishing gradients. This allows LSGANs to perform more stable during the learning process. This benefit can also be derived from another perspective: LS loss function is flat only at one point, instead sigmoid will saturate when x is relatively large (figure 6).



(a) (b)

FIGURE 5. (a) Sigmoid cross entropy loss f.; (b) Least squares loss f.

WGAN

The Wasserstein Generative Adversarial Network was presented in 2017 (M. Arjovski). It is an extension of the GAN that seeks an alternate way of training the generator model to better approximate the distribution of data observed in a given training dataset. WGAN wants to improve the stability in training phase and provides a loss function that correlates with the quality of the generated images.
One principal aspect is that, unlinke GAN, this requires a conceptual shift away from a discriminator, that predicts the probability of a generated image saying if it is "real" or "fake", toward the idea of a critic model that scores the "realness" of a given image.
This conceptual shift is possible using the earth mover distance (EMD), or Wasserstein distance, to train the GAN that measures the distance between the data distribution (real data) and the generating data distribution (new data).

Paper's authors demonstrate that a critic neural network can be trained to approximate the Wasserstein distance, and, in turn, used to effectively train a generator model: ".. we define a form of GAN called Wasserstein-GAN that minimizes a reasonable and efficient approximation of the EM distance, and we theoretically show that the corresponding optimization problem is sound".

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[\, \|x - y\| \,\big]$$

where (x,y) indicates how much "mass" must be transported from x to y in order to transform the distribution P_r into the distribution P_g. The EM distance then is the "cost" of the optimal transport plan.
The Wasserstein distance has the properties that it's continuous and differentiable and continues to provide a linear gradient, event after the critic is well trained.

The substantial differences mentioned between discriminator and critic are the following:

- The discriminator learns very quickly to distinguish between fake and real, and as expected provides no reliable gradient information.
- The critic, however, can't saturate, and converges to a linear function that gives remarkably clean gradients everywhere.

The loss of the discriminator appears to relate to the quality of images created by the generator. Specifically, the lower the loss of the critic when evaluating generated images, the higher the expected quality of the generated images. This is important as unlike other GANs that seek stability in terms of finding an equilibrium between two models, the WGAN seeks convergence, lowering generator loss.

Defining the networks. Although the theoretical aspects shown so far seem to suggest the opposite, there are few changes that are made compared to the DCGAN, of which we will use part of the architecture.
The actual changes are as follows:

— Use a linear activation function in the output layer of the critic model (instead of sigmoid).
— Use Wasserstein loss to train the critic and generator models that promote larger difference between scores for real and generated images.
— Update the critic model more times than the generator each iteration (e.g. 5).

Loss functions and Optimizers. The Wasserstein loss function seeks to increase the gap between the scores for real and generated images. So the loss function's task is to measure the Wasserstein distance between p_r and p_g:

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r}[D(\boldsymbol{x})] - \mathbb{E}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_g}[D(\hat{\boldsymbol{x}})]$$

where D is the set of functions that are 1-Lipschitz (a function f is 1-Lipschitz if |f(a) - f(b)| ≤ |a - b| for all a and b in the domain of f).
In order to enforce the Lipschitz constraint, the author make use of f clipping the weights of the critic between [-c, c] for some positive constant c.l However, weight clipping can result in undesired behaviour such as exploding or vanishing gradients; look at WGAN-GP (Gradient Penalty) for other solution.

We can summarize the function as it is described in the paper as follows:

— Critic Loss = [average critic score on real images] – [average critic score on fake images]. In the case of the critic, a larger score for real images results in a larger resulting loss for the critic, penalizing the model. This encourages the critic to output smaller scores for real images. For example, an average score of 20 for real images and 50 for fake images results in a loss of -30; an average score of 10 for real images and 50 for fake images results in a loss of -40, which is better, and so on.
— Generator Loss = -[average critic score on fake images]. In the case of the generator, a larger score from the critic will result in a smaller loss

for the generator, encouraging the critic to output larger scores for fake images. For example, an average score of 10 becomes -10, an average score of 50 becomes -50, which is smaller, and so on.

Where the average scores are calculated across a mini-batch of samples. The sign of the loss does not matter in this case, as long as loss for real images is a small number and the loss for fake images is a large number. The Wasserstein loss encourages the critic to separate these numbers.

In the paper is also specified that: "WGAN training becomes unstable at times when one uses a momentum based optimizer such as Adam. We therefore switched to RMSProp". Use the RMSProp version of gradient descent with small learning rate and no momentum (e.g. 0.00005).

The novelties introduced. The benefits introduced by WGANs concern the better stability in the training phase. It's also less sensitive to model architecture and choice of hyperparameter configurations. Two are the most important aspects to emphasize:

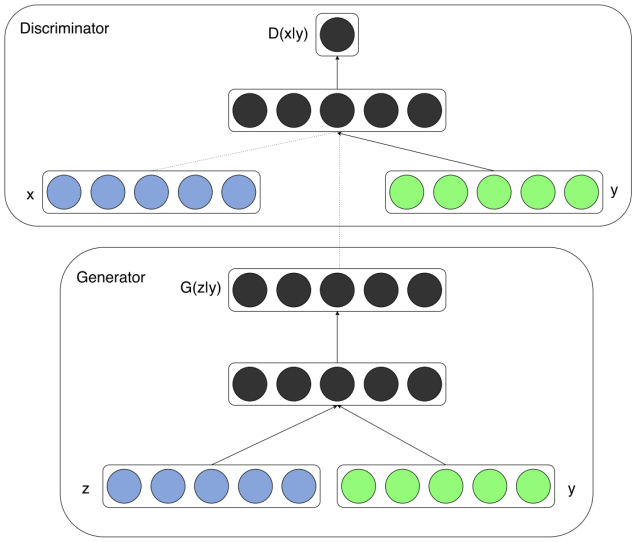
— there is no presence of collapse mode in the various experiments carried out,

— the generator continues, however, to learn, despite the discriminator/critic performs well.

CDCGAN

The Conditional Generative Adversarial Networks was presentend in latest 2014 by M. Mirza. The idea behind is that we want to condition, using features, the models in performing better. In CGAN, labels act as an extension to the latent space z to generate and discriminate images better.

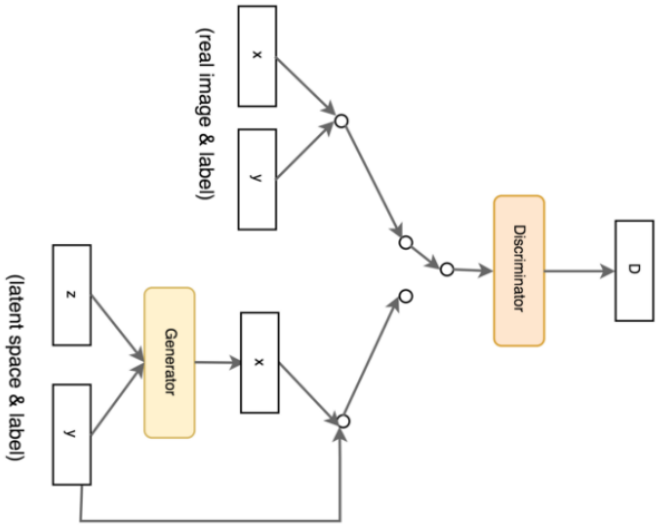
In GAN, there is no control over modes of the data to be generated. The conditional GAN changes that by adding the label y as an additional parameter to the generator. We also add the labels to the discriminator input to distinguish real images better. In the conditional GAN, we concatenate the conditional variable y to the input of every layer, for both the generator and discriminator. A conditional GAN is a supervised network that needs for each data point x a label y.



In the generator the prior input noise $p_z(z)$, and y are combined in joint hidden representation. In the discriminator x and y are presented as inputs and to a discriminative function.

The goal for the generator is then to generate samples x that are both realistic and match the context y. It is possible to use various types of data for y, such as discrete labels, text or even images. In CGAN, we can expand the mechanism to include other labels that the training dataset may provide (y and then z, w, k, etc.).

Defining the networks. So far we have talked about CGAN, but to be more precise, the model we used refers to a Conditional DCGAN, which brings with it the use of the label y with the architecture of a Deep Convolutional GAN, which as we have seen is more suitable in the generation of more detailed images.



In the figure above is possible to understand better the conditional adversarial network used. There are many ways to encode and incorporate the class labels into the discriminator and generator models. A best practice involves using an embedding layer followed by a fully connected layer with a linear activation that scales the embedding to the size of the image before concatenating it in the model as an additional channel or feature map.

Loss functions and Optimizers. The cost function for CGAN is the same as GAN, with the addiction of y both in the generator and discriminator loss.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

Parameters for G are trained to minimize log(1-D(G(z)), and parameters for D are trained to minimize logD(x), following the above two-player min-max game with value function V(D,G)

The novelties introduced. There are two motivations for making use of the class label information in a GAN model.

— Improve the GAN.

— Targeted Image Generation.

Additional information that is correlated with the input images, such as class labels, can be used to improve the GAN. This improvement may come in the form of more stable training, faster training, and/or generated images that have better quality. Class labels can also be used for the deliberate or targeted generation of images of a given type.
Alternately, a GAN can be trained in such a way that both the generator and the discriminator models are conditioned on the class label. This means that when the trained generator model is used as a standalone model to generate images in the domain, images of a given type, or class label, can be generated.

5. Use-case: Lego Minifigures

The use case, chosen to evaluate the techniques presented above, concerns the generation of new images starting from a given dataset. As reference images we have chosen the Lego Minifigures. It has been possible to scrape the inventory page from Bricklink, where are collected most of the minifigures created until now; approximately two thousand.

FIGURE 6. Lego minifigures dataset

From the picture above (fig. 7) you can see the various minifigures that fill the dataset and discover that in Lego world there aren't only the classic yellow lego minifigure but a series of characters coming from various themes

like ninja, harry potter, heroes, etc. This implies the possibility of characters with different sex and color, or wearing hat, cloak, helmet, etc.
For this reason, given the various differences between characters, we decided to test the various GANs on two different datasets, one with minifigures and the other with faces, making a crop on the first ones.

FIGURE 7. Lego faces dataset

We decided to evaluate both cases because, on one hand, the minifigures contain more details and differences in their entirety than the individual faces, allowing them to have different colors and aspects; but at the same time this could lead to a high noise in the generation of new images; in contrast, with faces, GANs should be able to detect/reconstruct the individual details better.

6. Experiments

The project wants to verify the quality of the various techniques presented so far in the reconstruction/generation of new images depicting possible new lego minifigures/faces. As said before, we will do two experiments, the first using the dataset of the minifigures, while the second will use the dataset of the faces of the minifigures.
Before going into the details of the various experiments, it is necessary to understand how to evaluate the results obtained by each individual GAN.

6.1. GAN's evaluation mode. — First of all, we must immediately remember that there is no way to objectively assess the progress of the training and the relative or absolute quality of the model from loss alone.
For this reason over time a series of evaluations have been developed that can help to assess the performance of a GAN model based on the quality and diversity of the generated images. So let's see some of the most important techniques used so far:

— *Manual evaluation.* Many GAN practitioners fall back to the evaluation of GAN generators via the manual assessment of images synthesized by a generator model. This involves using the generator model to create a batch of synthetic images, then evaluating the quality and diversity of the images in relation to the target domain. This may be performed by the researcher or practitioner themselves.

— *IS.* The inception score computes the KL divergence between the conditional class distribution and the marginal class distribution.It measures the quality of generated images and their diversity. Let's first talk about entropy in the sense of a random variable or a probability distribution. Entropy can be viewed as randomness. If the value of a random variable x is highly predictable, it has low entropy. On the contrary, if it is highly unpredictable, the entropy is high. Now let's look at the equation to compute IS.

$$\mathbf{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_a} D_{KL}(p(y|\mathbf{x}) \parallel p(y)) \right)$$

1. Conditional Probability: We want the conditional probability P(y|x) to be highly predictable (low entropy). i.e. given an image, we should know the object type easily. So we use an Inception network to classify the generated images and predict P(y|x) — where y is the label and x is the generated data. This reflects the quality of the images.

2. Marginal Probability: The Marginal probability is computed as P(y)=

$$\int_z p(y|x = G(z))dz$$

If the generated images are diverse, the data distribution for y should be uniform (high entropy).

One shortcoming for IS is that it can misrepresent the performance if it only generates one image per class. p(y) will still be uniform even though the diversity is low.

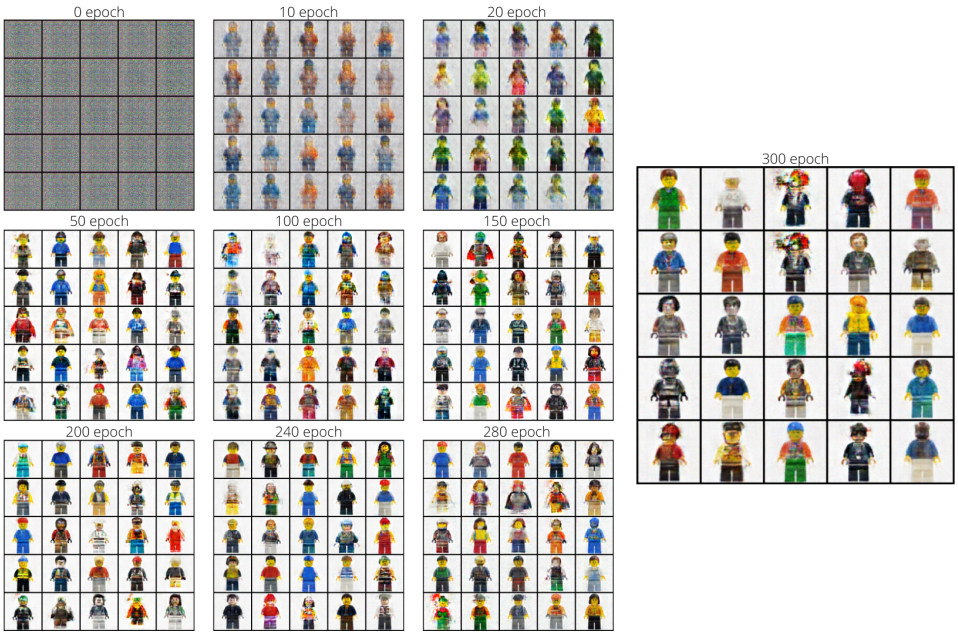
— *FID.* Fretchet Inception Distance is a more principled and comprehensive metric and has been shown to be more consistent with human evaluation in assessing the realism and variation of the generated samples. We use the Inception network to extract features from an intermediate layer. Then we model the data distribution for these features using a multivariate Gaussian distribution with mean µ and covariance . The FID between the real images x and generated images g is computed as: where

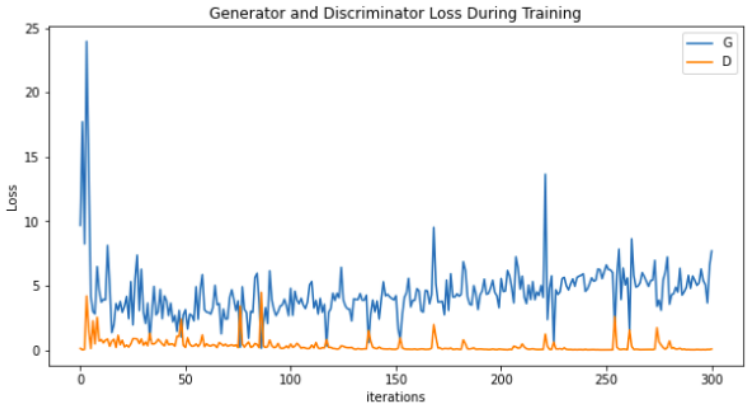
$$\texttt{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{\frac{1}{2}})$$

Tr sums up all the diagonal elements.
Important: Lower FID values mean better image quality and diversity. Furthermore FID is sensitive to mode collapse.
FID is more robust to noise than IS. If the model only generates one image per class, the distance will be high. So FID is a better measurement for image diversity. FID has some rather high bias but low variance.

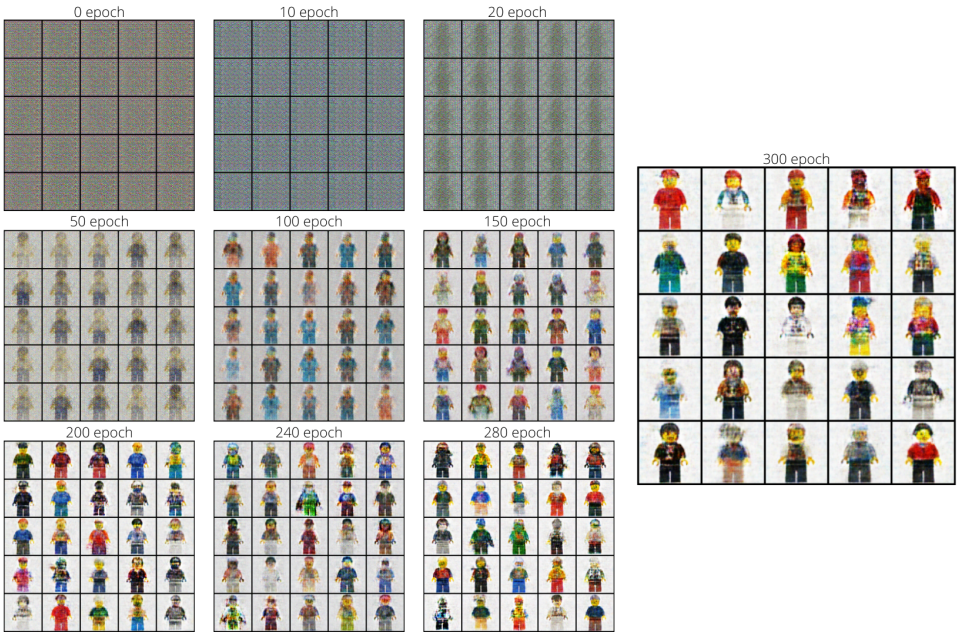
6.2. Lego minifigures. — This first experiment was carried out using the LEGO minifigures dataset (complete body). The dataset consists of 2291 images with a resolution of 150x150 pixels.
Below are the results obtained by the various GANs, in terms of images generated during the various training epochs (300) and the FID score obtained, to fully assess their capabilities and similarities with real samples.

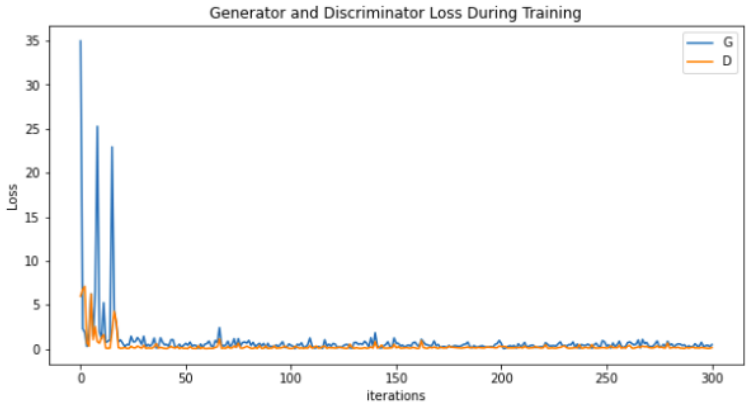
DCGAN



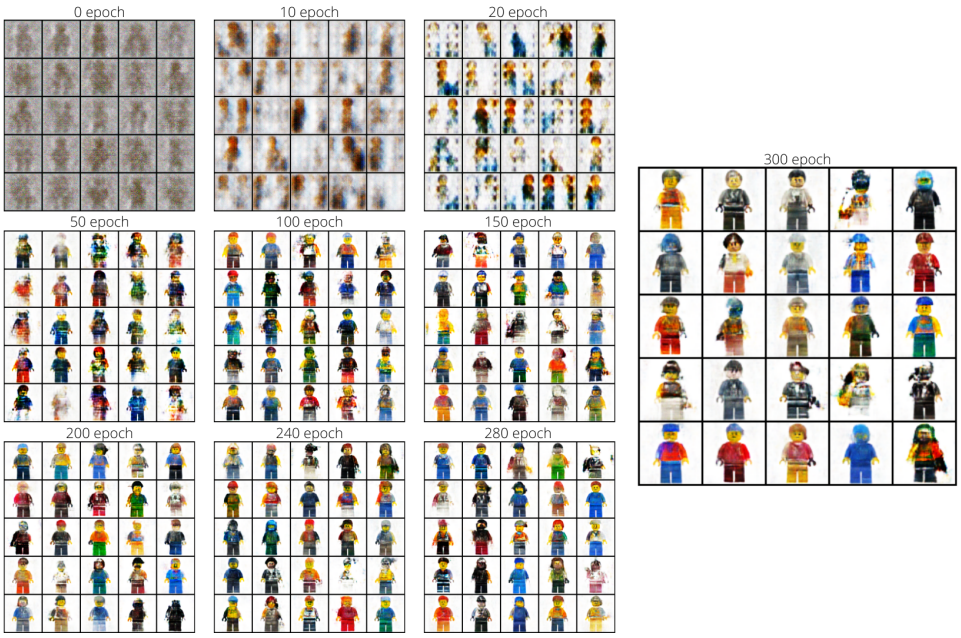


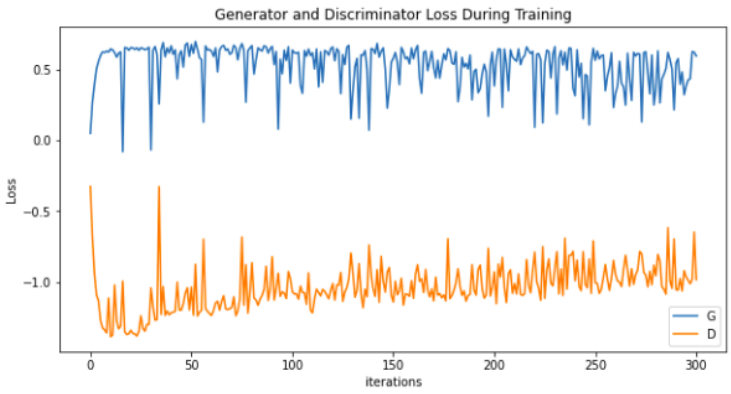
LSGAN


0 epoch
10 epoch
20 epoch
300 epoch
50 epoch
100 epoch
150 epoch
200 epoch
240 epoch
280 epoch


Generator and Discriminator Loss During Training
G
D
Loss
iterations
0
5
10
15
20
25
30
35
0
50
100
150
200
250
300

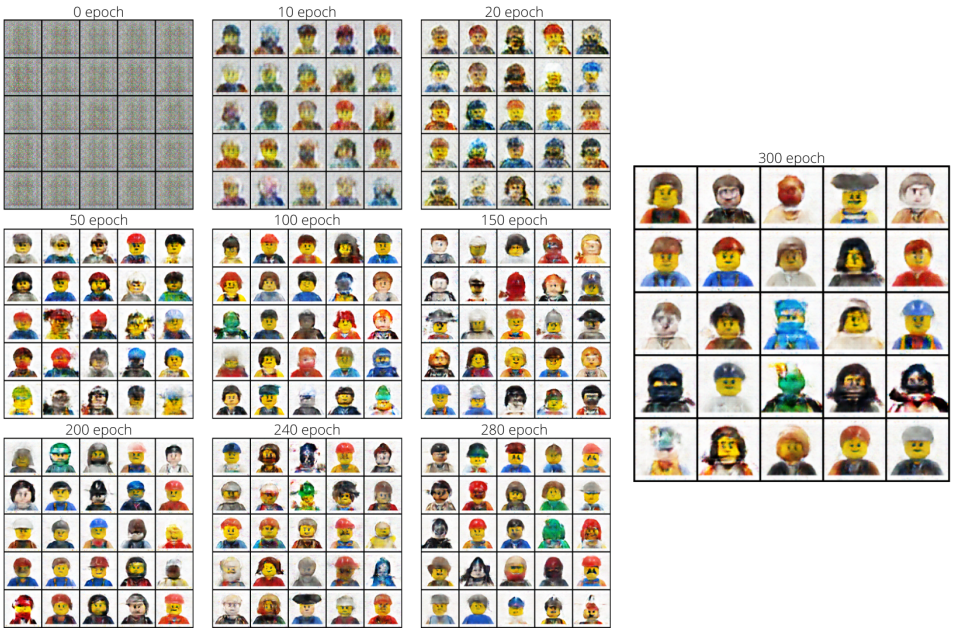
WGAN

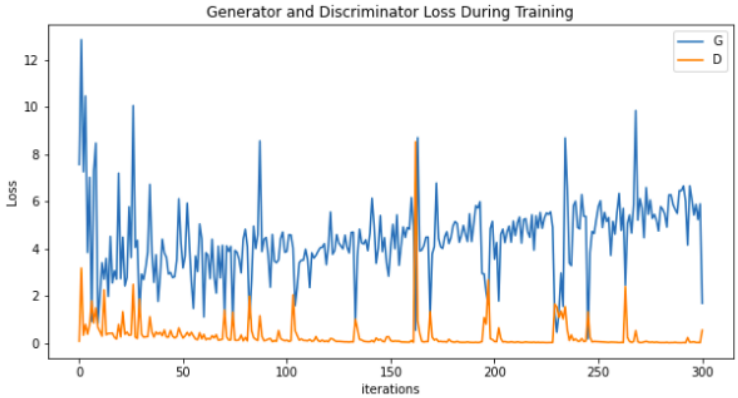

0 epoch
10 epoch
20 epoch
300 epoch
50 epoch
100 epoch
150 epoch
200 epoch
240 epoch
280 epoch


Generator and Discriminator Loss During Training
Loss
iterations
G
D
0.5
0.0
−0.5
−1.0
0
50
100
150
200
250
300

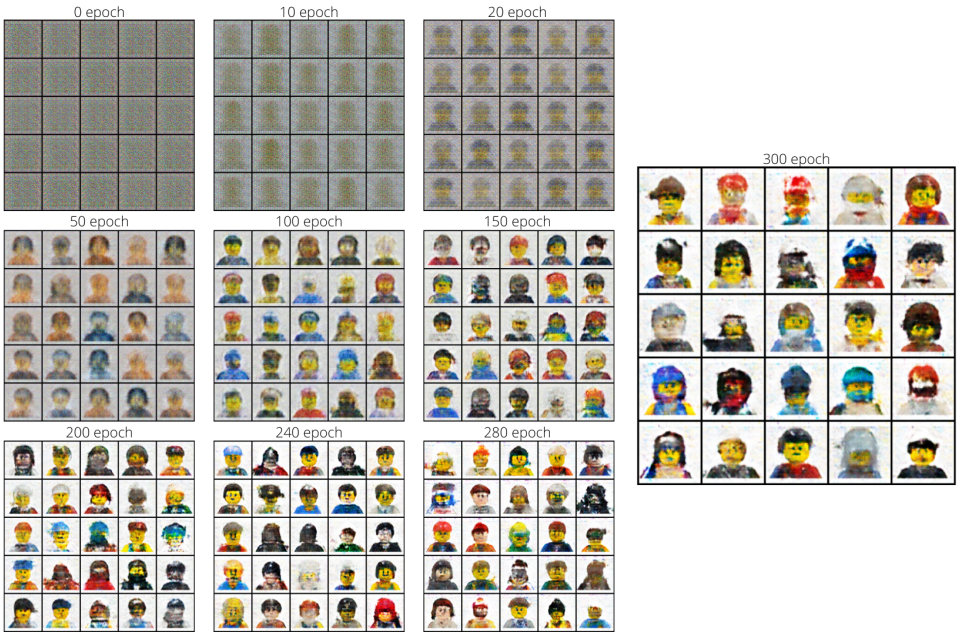
6.3. Lego faces. — The second experiment involves the use of a dataset containing 2291 face images croped from the LEGO minifigures previously used. The resolution of each image is 100x100 pixels.
Below are the results obtained by the various GANs, in terms of images generated during the various training epochs (300) and the FID score obtained, to fully assess their capabilities and similarities with real samples.

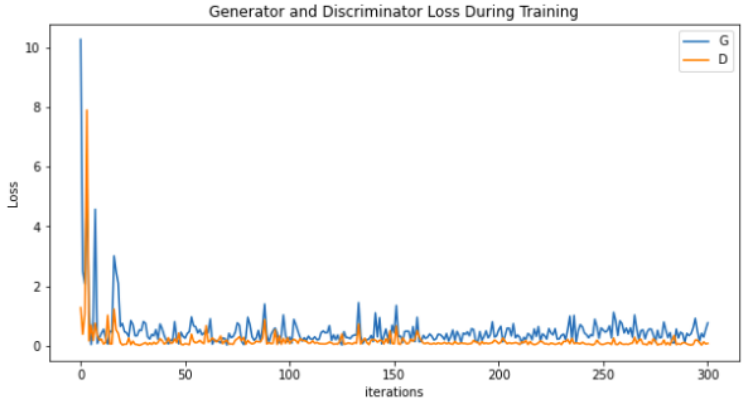
DCGAN



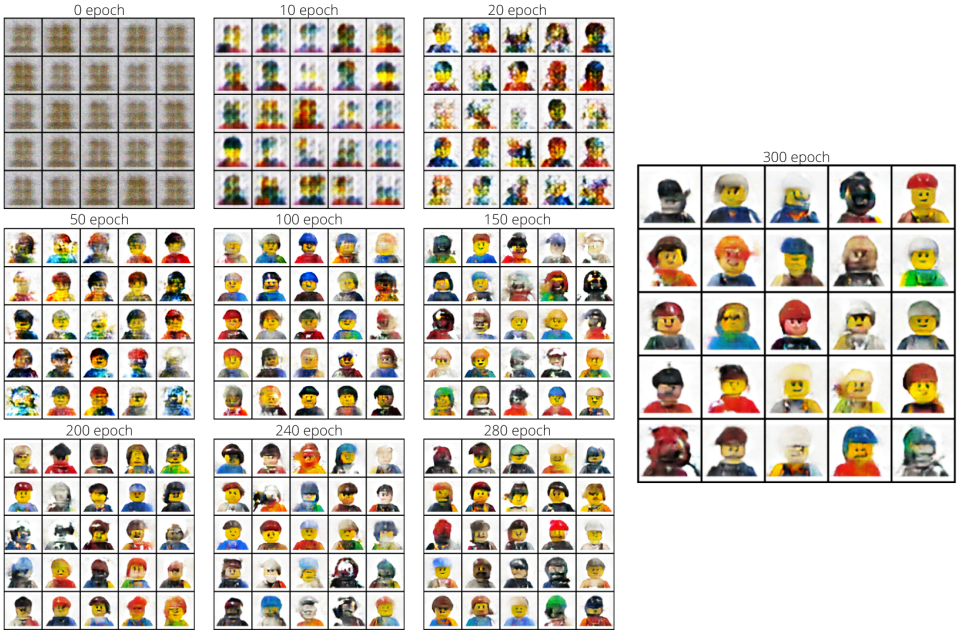


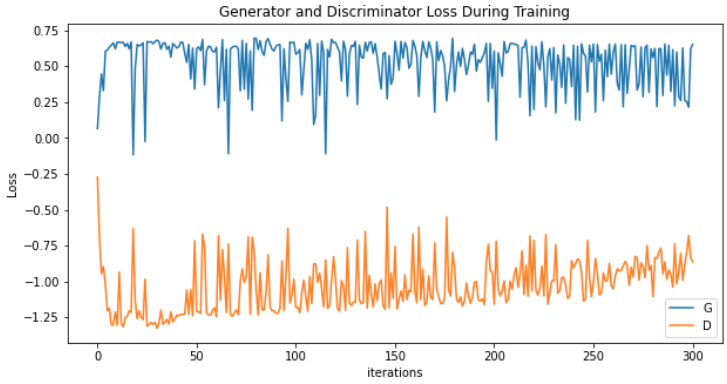
LSGAN


0 epoch
10 epoch
20 epoch
300 epoch
50 epoch
100 epoch
150 epoch
200 epoch
240 epoch
280 epoch


Generator and Discriminator Loss During Training
G
D
Loss
iterations
0
2
4
6
8
10
0
50
100
150
200
250
300

WGAN


0 epoch
10 epoch
20 epoch
300 epoch
50 epoch
100 epoch
150 epoch
200 epoch
240 epoch
280 epoch


Generator and Discriminator Loss During Training
Loss
iterations
G
D
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
0
50
100
150
200
250
300

6.4. Final evaluation. — The results obtained demonstrate how the GANs studied can achieve good results in image generation. Images recreated, to human eye, have good details with a bit of noise and probably it could reach, with more training sessions, an almost optimal level.
Looking instead at the FID results obtained by the GANs in the experiments, we can declare as "winner" the WGAN, both in the case of mini-figures and faces. The table below shows the results obtained by each GANs.

Dataset	*LEGO minifigures*	*LEGO faces*
DCGAN	195.8117	235.1506
LSGAN	247.4799	256.0163
WGAN	168.7412	190.9532

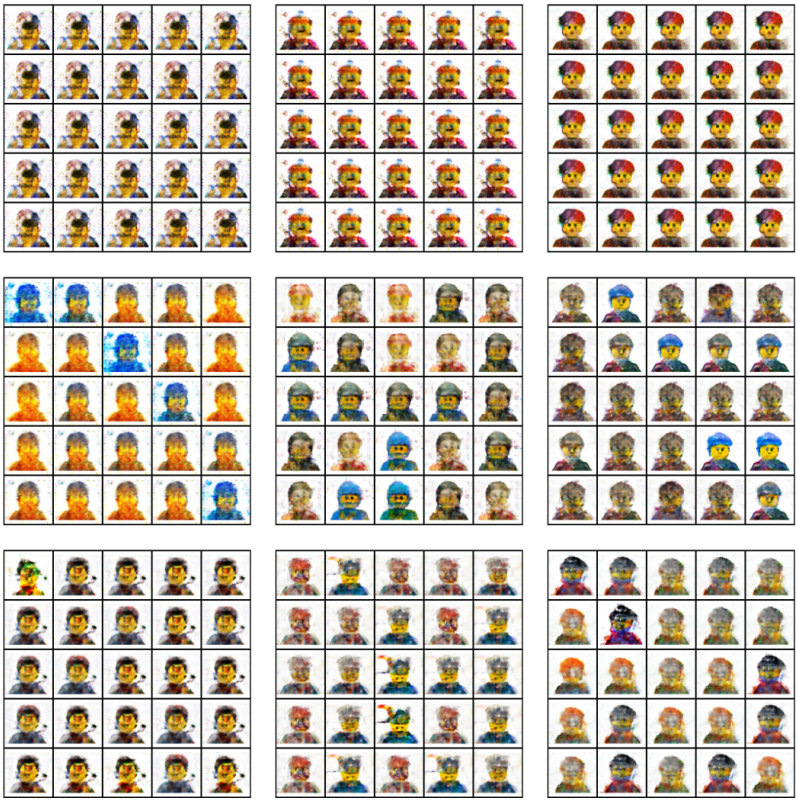
Clearly, as mentioned above, to human eye, faces have more similarities than the whole minifigures, full of details and with more noise. But it's at the same time curious to evaluate from the values obtained in the table of FID that, if we don't just stop and look, the distance is actually smaller in the minifigures than for the faces.

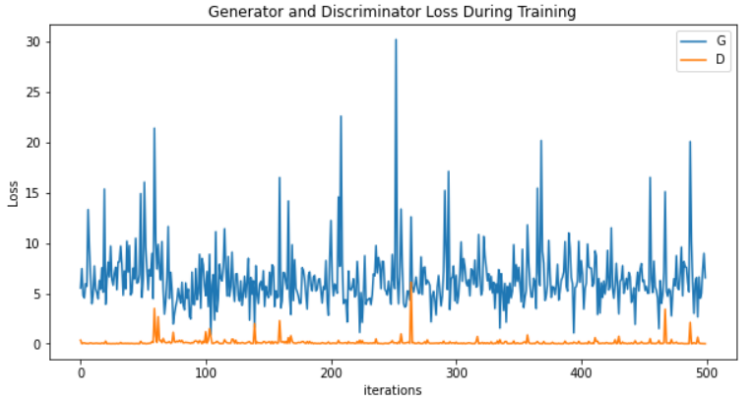
It would therefore be interesting to see, if more training could be done, if the qualities of our generators improves even more. The use of minifigures not exactly "simple", so different from the usual LEGO yellow man may have given some problems, so another aspect is to study if the overall quality improves using only classic minifigures. It would not seem to be present, if not minimally, mode collapse, because there is still a great diversity between the various samples generated.

6.5. Lego faces label. — Separate mention for the experiment performed using Conditional DCGAN. In this case we used only the LEGO faces dataset. The dataset was then divided into four subcategories: bald, hair, hat and helmet.

This aspect, as seen in previous experiments, particularly damages and compromises the creation of clean faces, being the one selected the feature that best divides the various characters. Below an image of the four subcategories.

CDCGAN




Generator and Discriminator Loss During Training
G
D
Loss
iterations
0
5
10
15
20
25
30
0
100
200
300
400
500

7. Conclusion and Future Work

The development of the project has allowed us to go into the details of some of the most famous GANs, knowing the implementation's details and their pros and cons. The world of GANs, despite being very young, allows us to discover many different fields of generation, with the possibility to choose among the many, and various, techniques studied over these few years. The results obtained, however, allow to highlight the strength of these techniques in being able to recreate, in a very similar way to the original, well made copies.

Regarding the future aspects, it would be interesting to go deeper and discover the 'Cross-Domain Image Generation' field. They study the problem of transferring a sample in one domain to an analog sample in another domain. Applied to our project, new lego faces could be created starting from humans face's samples. Until now human to emoji has been tested, with convincing results.

Literature and sources

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio — *Generative adversarial nets.* — NIPS, 2014.

[2] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, S. Gelly — *Are GANs Created Equal? A Large-Scale Study.* — NIPS, 2014.

[3] M. Pieters, M. Wiering — *Comparing Generative Adversarial Network Techniques for Image Creation and Modification.* — IEEE, 2018.

[4] S. Hitawala — *Comparative Study on Generative Adversarial Networks.* — IEEE, 2018.

[5] A. Radford, L. Metz, S. Chintala — *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* — ICLR, 2016.

[6] J. Curtò, I. Zarza, F. Torre, I. King, M. Lyu — *High-resolution Deep Convolutional Generative Adversarial Networks.* — IEEE, 2020.

[7] X. Mao, Q. Li, H. Xie, R. Lau, Z. Wang, S. Smolley — *Least Squares Generative Adversarial Networks.* — ICCV, 2017.

[8] X. Mao, H. Xie, R. Lau, Z. Wang, S. Smolley — *On the Effectiveness of Least Squares Generative Adversarial Networks.* — IEEE, 2018.

[9] M. Mirza, S. Osindero — *Conditional Generative Adversarial Nets.* — 2014.

[10] J. Gauthier — *Conditional generative adversarial nets for convolutional face generation.* — IEEE 2015.

[11] M. Arjovsky, S. Chintala, L. Bottou — *Wasserstein GAN.* — IEEE 2017.

Andrea Magnante - 1876801