Multiple Input Transfer Learning GAN - MITLGAN - Making a chess-like game a bit differently

Elo Zsombor
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
elozsombor@gmail.com

Abstract. The goal of this project is to experiment with the opportunity of using pretrained, simple GANs (more precisely their generators), to make a larger GAN, which combine the simple models' results into something else. I used as an example images of chess-like board games (chess, checkers, shogi, xiangqi etc.), because I really like them, and because they can be easily separated into multiple parts. In this case those parts were boards and pieces. Hopefully it is another way, to improve GANs training in specific situations.

1 Introduction

Generative Adversarial Networks [1] (from now on: GAN) are an unsupervised learning technique for generating new data and discriminating between generated and original data. The former usage of this technology is the one, we try to applicate most of the time, this is our case now as well. A GAN has (usually) two main parts: a Generator and a Discriminator. These two parts are playing a mini-max game between them. In this project we build 3 GANs: one GAN for generating chess-like boards, one GAN for generating chess-like pieces and finally one GAN (MITLGAN) (using the other two) to generate chess-like board games (more specifically an image of the beginning of a board game, at step 0). We are trying to understand, if it is better, to make a GAN that way, instead of making a more "traditional" GAN (e.g.: DCGAN, WGAN etc.).

2 Data sets

2.1 Making of the data sets

Originally there weren't any data set, that matched my ideas, so I had to make one myself. It took a lot of time, and honestly the data sets aren't that large and clean as I would have liked them to be. I downloaded many images from the internet and tried to select the images, that were somehow usable. It was necessary, to filter them once before actually making the data sets, because many of them were not even about board games or chess-like games. After this initial step I made 3 separate data sets: one for boards, one for pieces and one that has both of them. The making of the latter was more simple, because I only had to crop some images. The one for boards had some images, that originally

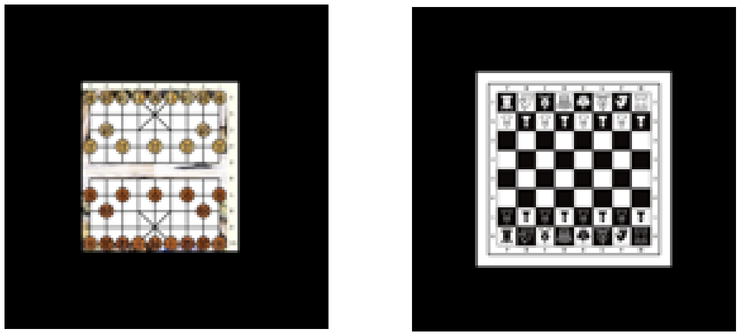

Figure 1: Samples of the "filled" boards data set.

didn't have any pieces on them, but in many cases I had to remove the pieces (sometimes one-by-one) from the board with some image editor (I used GIMP most of the time). The one data set of pieces was unfortunately very hard to

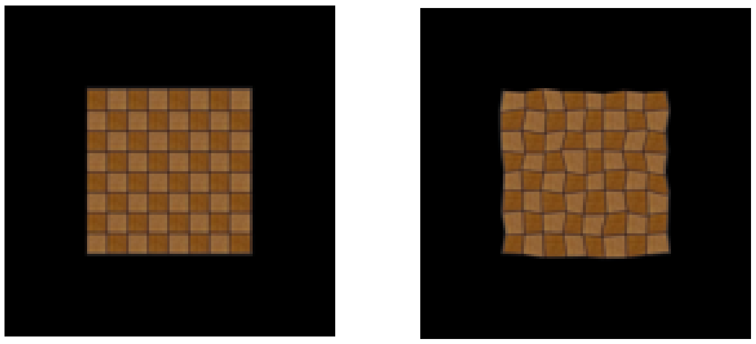

Figure 2: Samples of the empty boards data set.

make, because many images were using not uniform colors, they looked like they were jittered in some way. I erased the board under the pieces, so it would look like piece sets on a white canvas, more or less positioned the same way they were on the original images. I used image augmentation as well, because that almost always improve the training process.[2] The process to make the 3 data sets took many weeks, because I tried to be as precise as I can.

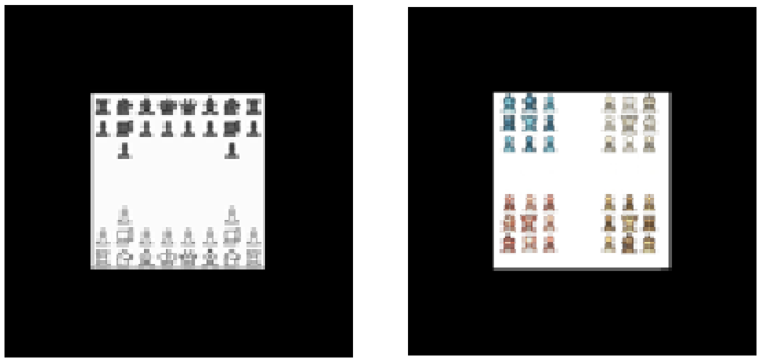

Figure 3: Samples of the pieces data set.

2.2 Parameters of the data sets

I used Google Colab[1] for training (as they have more powerful GPUs, than I have in my laptop (although I tried to use mine, but it was way too small and slow)), which has a time limit for their users, therefore I made the biggest images, that were big enough to see something on them, but not bigger. As we will see it later, it worsened my results in some way, but more of that later. In the end I made the images into 128x128, RGB images, so they can be easily fed into a model. The size of the data sets were: 1416 images for the boards, 1164 images for the pieces, and 1077 images for the "filled" boards.

3 WGAN

3.1 Wasserstein distance

GANs have shown great results in many generative tasks over the past years, but the original process is known to be slow and unstable, therefore in recent years, many new GANs have been made, to make the process better and more stable. WGAN provides a better solution, to make the training more stable. The problem in the original process is for quantifying the similarity between two probability distributions, which is a measure of our loss function. [3] Instead of (one of the original loss functions) JS Divergence, we will use the Wasserstein Distance. It is called Earth Mover's distance as well, short for EM distance, because it can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution. The cost is quantified by: the dirt moved multiplied by how far we moved the dirt. A discrete visualization is Figure 4.

The next formula shows, how to mathematically interpret the above:

[1]`https://colab.research.google.com/`

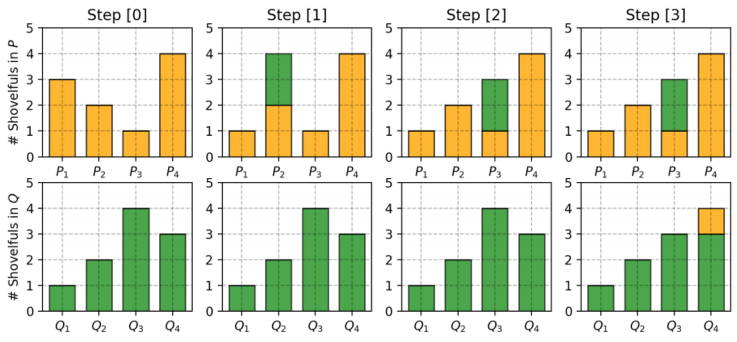

Figure 4: A discrete visualization of Wasserstein Distance.

$$W\left(p_r, p_g\right)=\inf_{\gamma \sim \Pi\left(p_r, p_g\right)} \mathbb{E}_{(x, y) \sim \gamma}[\|x-y\|] \tag{1}$$

In the formula, $\Pi\left(p_r, p_g\right)$ is the set of all possible joint probability distributions between p_r and p_g, where p_r is the data distribution over a real sample and p_g is the generator's distribution over a data.

Wasserstein Distance is better than JS and KL Divergence, because it doesn't have sudden jumps, and differentiable everywhere, even when the two distributions are fully overlapped. It is very helpful, to stabilize the training process using the gradient descents.

If we want to use the Wasserstein Distance as our loss function, then there is a transformed version of it:

$$W\left(p_r, p_g\right)=\frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_r}[f(x)]-\mathbb{E}_{x \sim p_g}[f(x)] \tag{2}$$

In the transformed version there is a demand to satisfy. The function f in the new form of Wasserstein metric is demanded to satisfy $\|f\|_L \leq K$, meaning it should be *K-Lipschitz continuous*.

A real-valued function $f: \mathbb{R} \rightarrow \mathbb{R}$ is called *K-Lipschitz continuous* if there exists a real constant $K \geq 0$ such that, for all $x1, x2 \in \mathbb{R}$,

$$\left|f\left(x_1\right)-f\left(x_2\right)\right| \leq K\left|x_1-x_2\right| \tag{3}$$

K is the Lipschitz constant for function $f(.)$. If you want to know more about WGAN, then i recommend [3].

3.2 The smaller GANs: Boards and Pieces

Both of the smaller models were implementations of WGAN. As I think, both of them have more or less the same complexity, therefore I made them symmetric to each other: the models look the same. The generator was a simple Convolutional Network with some easy convolutional steps to make the noise into an image. The noise's size was 1516 (128*128*3 as I saw it fit for making easier the convolutional steps), and the output was a 128x128x3 (RGB) image. The discriminator was simple as well, it used transposed convolutional steps for a while and after flattening that, in the end a dense layer of 1. I used some dropout as well, to make some nodes more responsible for specific features. The generators had 2417796 trainable parameters, the discriminators 2448097. Many inspiration taken from this [4].

4 Multiple Input Transfer Learning GAN - MITLGAN

4.1 The idea behind it

The idea came from transfer learning [5] and from how I see the world. I think, that if we make many little steps, instead of a big one, than we can achieve more precisely what we want in general. Transfer learning is used normally, to broaden the capabilities of a model (e.g.: making a cat classifier to classify dogs as well). I think, that I can use that technique to combine two capabilities, features as well. It is practically the same, but the model is trained on an entirely new data set it has never seen before, but many very similar to it in one way or another. The higher the similarity, the higher the chances, that this is working. This model I have never seen before, so I took the liberty, and named it Multiple Input Transfer Learning GAN (MITLGAN), so I can refer to it somehow. Multiple input, because we combine multiple inputs in it, and transfer learning to refer to the technique, that gave me the idea.

4.2 Model

Firstly I wanted to use only 1 to 2 dense layers, to concatenate the generators, but there would have been too small amount of trainable parameters, and i was afraid that it wouldn't work, because there are many things to "consider" for a model, when combining images into something entirely new. But after a while I got a brilliant idea: I got two images from the two generators... And if I got images, I can translate them into a new one, using Image to Image Translation [6]. So I freezed the two generators and concatenated the outputs of them (128, 128, 6) and then made it into one image with a dense-like convolutional layer (128, 128, 3). I concatenated them instead of adding them, because I think the model should learn, how much impact should each generator have. After that I translated that image with a ResNet-like structure to get an output of (128, 128, 3) in the end. The discriminator was just like it was in the other models, because if we combine them, like the generators, then it would have 2 inputs, but we have only one output. So it remained the same for this model as well.

5 Experiment

5.1 Numbers

As I didn't have all the time with me (limits, other projects, limited remaining time of the semester), I only trained the smaller models for 300-300 epochs (5 - 5 hours) on the boards and pieces, and the MITLGAN for 330 epochs (20 hours roughly) on the "filled" boards. The batch size was 32, as the GPU couldn't handle more at a time.

5.2 Results

The WGAN for the boards performed quite well in the earlier stages of the training, but with time it didn't really changed the output much unfortunately. Many images look like boards, that are melting away a bit at the edges, but the most of the image resembles boards.

The WGAN for the pieces performed poorly. The final model outputs many noise-like images. It only learned, that the different colored pieces usually are on the other side of the board, nothing more. I thought of leaving this part out of my last model, but I gave it some tries in the end.

The last model performed really interesting. It improved very fast in the earlier stages, but the improvement got slowed after like 100 epochs. But not like in the case of the GAN for boards, it still improved, and I think, that if I could gave it some more training it could improve more. In the end some pieces were on the board (some dots in zigzag). That means, that in some ways, the thought is not worthless, it has a chance to really work.

6 Evaluation

In the end I evaluated the GANs with Frechet Inception Distance (FID). It is an evaluation method, which uses a pretrained classifier model (in our case the Inception V3) without the top of it, to embed images. The embeddings of two sets of images are compared then , to see, how much the same they are. The Inception V3 embeds the images into the shape of (2048,), so the score can be anything between 1 and 2048. 1 means, that the sets of images are the same, 2048 means, they are as far apart, as they can be.

Table 1: Frechet Inception Distance results

Used Data Set	**FID**
`Boards-WGAN`	1618.1466
`Pieces-WGAN`	1434.0299
`Final-MITLGAN`	1449.0072

All of my GANs performed quite poorly on this evaluation, but I think, that it has got something with the Inception V3 model, I used. That model was not trained on any similar data to chess-boards before, so of course it doesn't recognize that very well.

The outputted images aren't that great as well, but I think they can be improved with as little, as a little more time (and of course with other things as well).

7 Conclusion and Future Work

I think, that this idea is not bad at all, but it should be compared with a simpler solution as well, to understand, if it is worth it or not. Based on the trainings I saw, I think that the idea could come in handy sometime in the future, but improvements should be made.

In the future I would like to broaden and clean the data sets to be absolutely sure, that it won't be a problem. After that I would like to compare this model with others, to see the differences and understand them somehow. If those are done, I would like to make rules for the generated games (simple ones), and recognize the pieces on the generated game.

8 Acknowledgments

The author would like to express his thanks to Pomazi Krisztian Daniel for his support as a scientific advisor. This work has been supported by the Department of Automation and Applied Informatics Budapest University of Technology and Economics.

References

[1] M. M. B. X. D. W.-F. S. O. A. C. Y. B. Ian J. Goodfellow, Jean Pouget-Abadieâ, "From gan to wgan." `https://arxiv.org/pdf/1406.2661.pdf`.

[2] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning." `https://link.springer.com/article/10.1186/s40537-019-0197-0`.

[3] L. Weng, "From gan to wgan." `https://arxiv.org/pdf/1904.08994.pdf`.

[4] D. Szurko, "Tensorflow 2.0 wgan-gp." `https://github.com/drewszurko/tensorflow-WGAN-GP`.

[5] Tensorflow, "Transfer learning and fine-tuning." `https://www.tensorflow.org/tutorials/images/transfer_learning`.

[6] T. Q. S. M. I. Yingxue Pang, Jianxin Lin and I. Zhibo Chenâ, Senior Member, "Image-to-image translation: Methods and applications." `https://arxiv.org/pdf/2101.08629.pdf`.

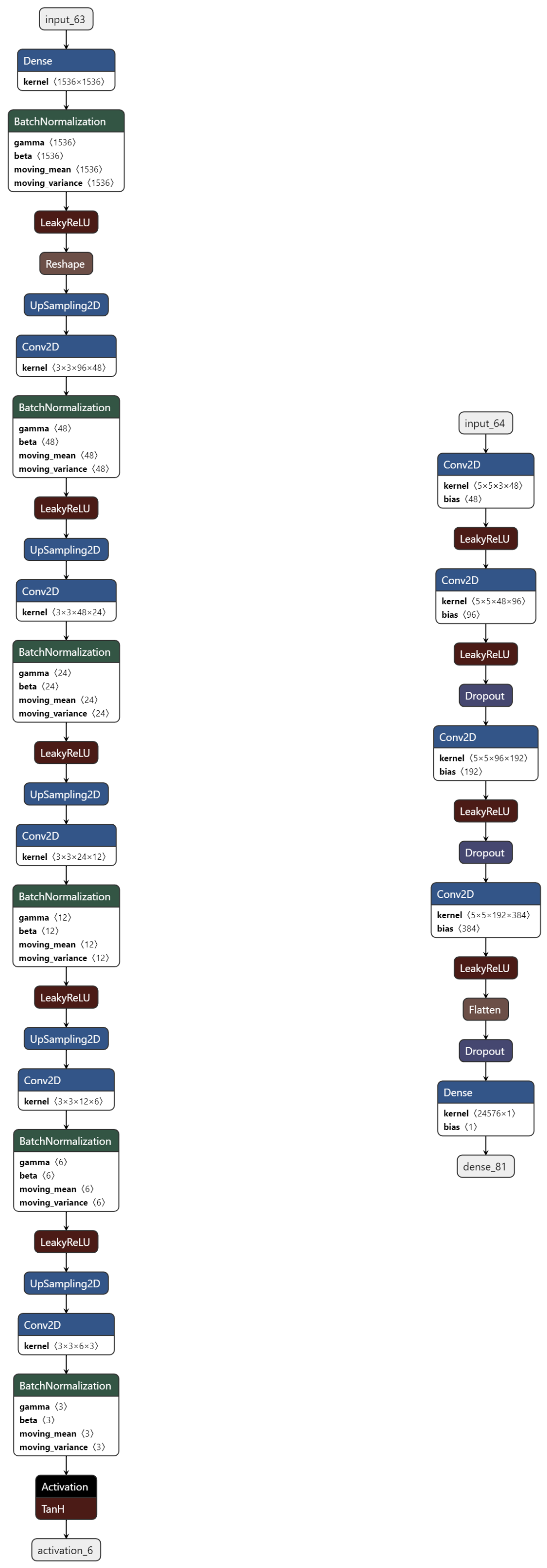

Figure 5: The WGAN models. The generator is on the right, the discriminator is on the left.

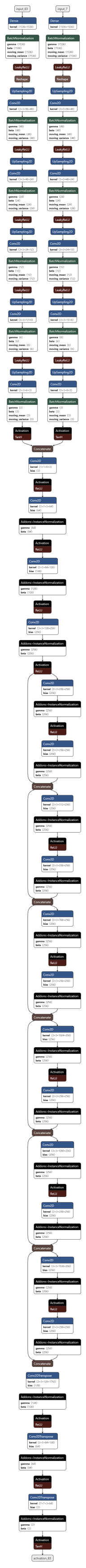

Figure 6: The generator of the MITLGAN. It is a bit big, but the main idea can be seen.

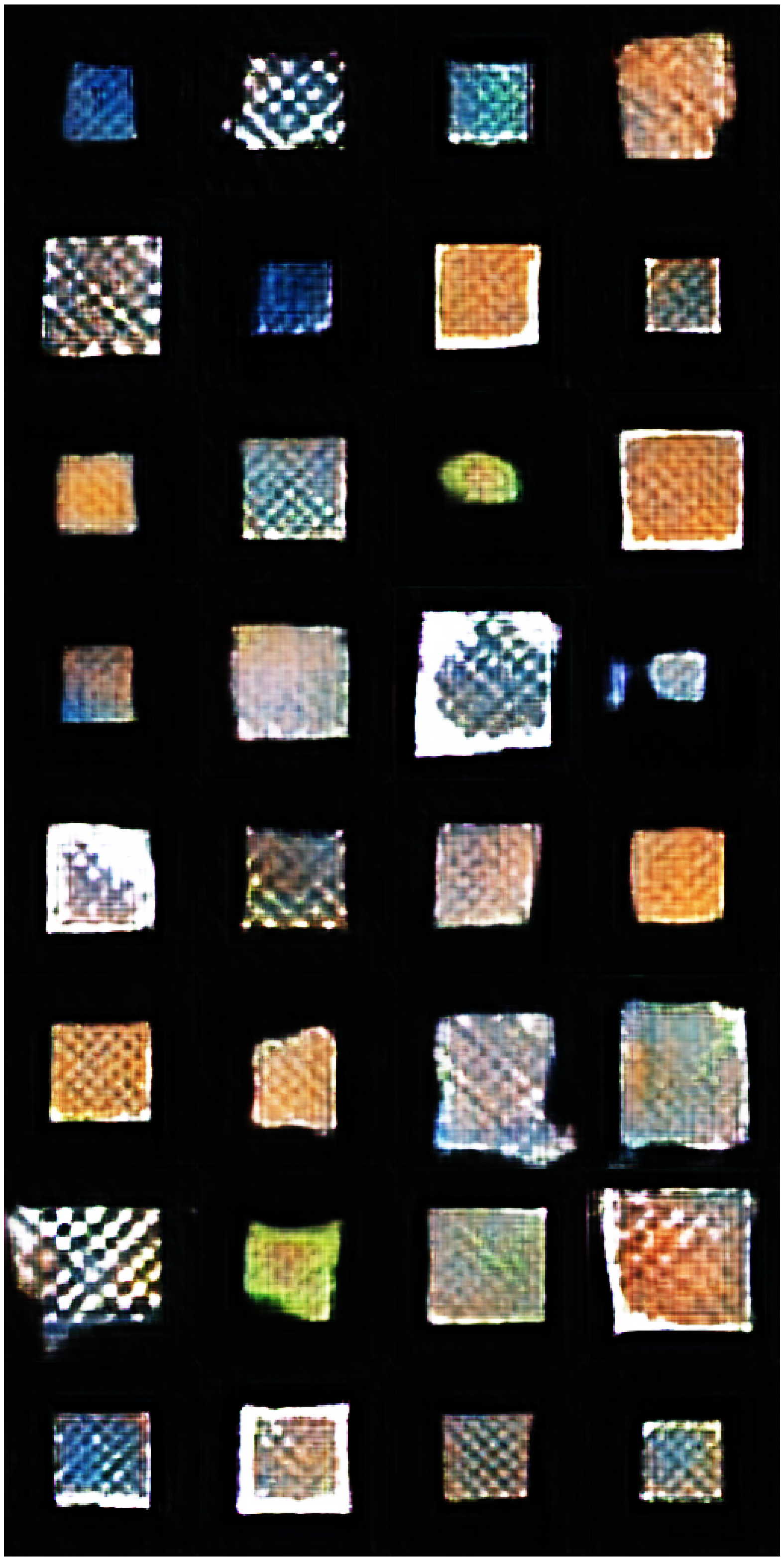

Figure 7: Some generated images of the WGAN for empty boards.

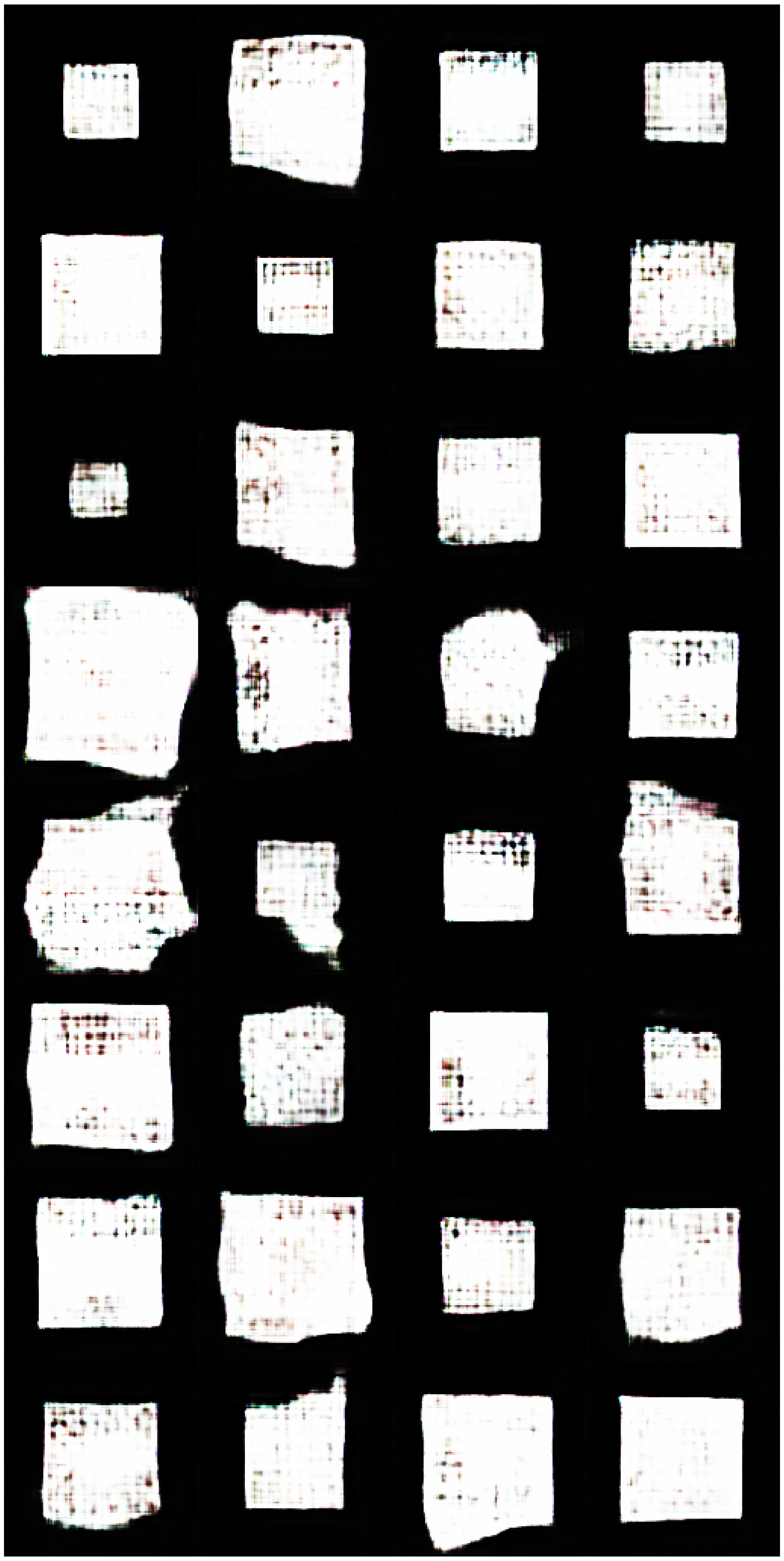
Figure 8: Some generated images of the WGAN for pieces.

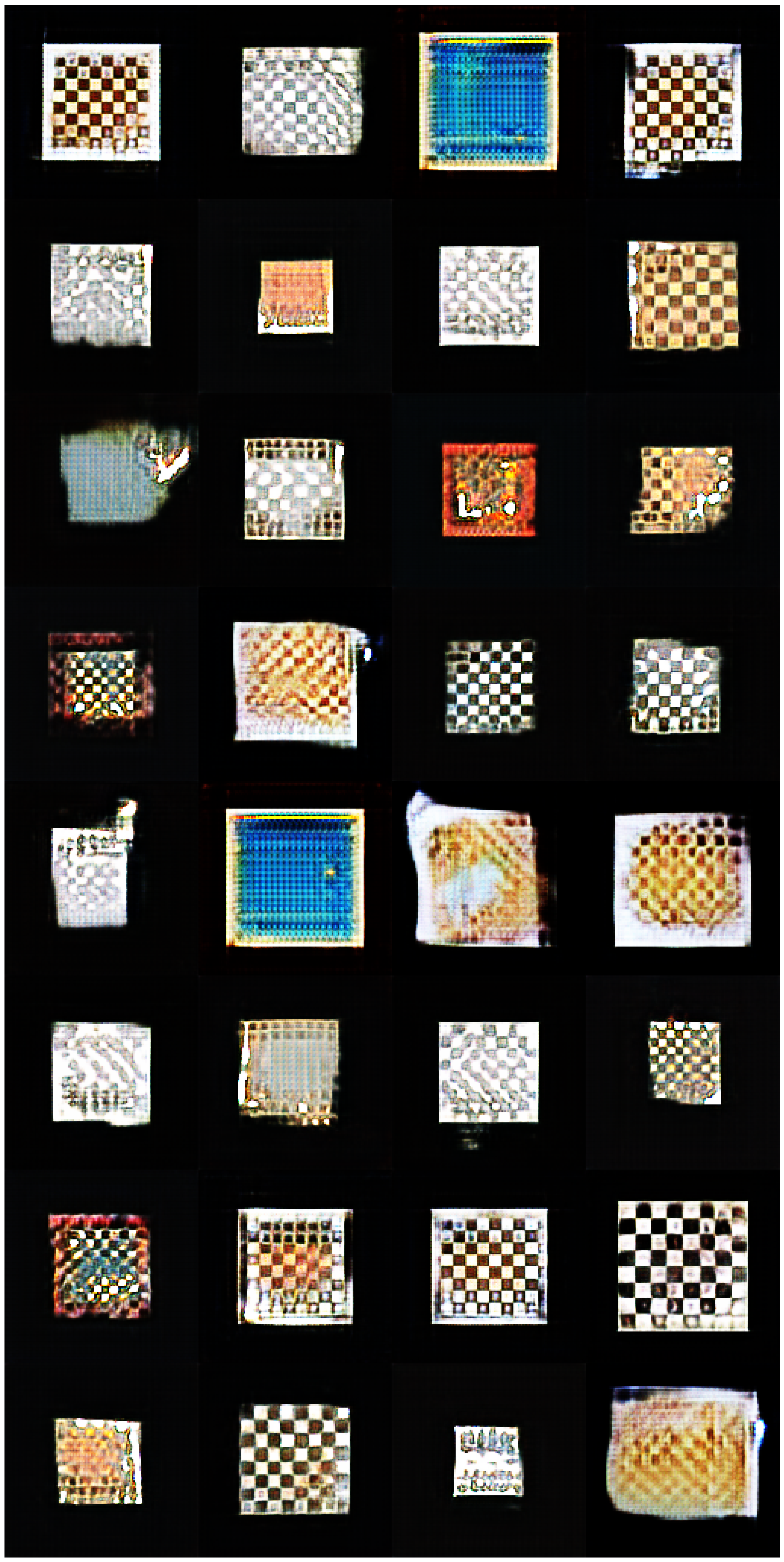

Figure 9: Some generated images of the MITLGAN.