FastGAN: Faster and Stabilized GAN

Authors:
Mauro Ficorella 1941639
Martina Turbessi 1944497
Valentina Sisti 1952657

Sapienza
May 2021

ABSTRACT

The main aim of FastGAN is to allow users with limited computing budget and resources to train a GAN. Moreover it eliminates the requirement of a big dataset for training. These are big advantages since traditional GANs required a lot of GPU computational power (i.e. one or more server-level GPUs with at least 16 GB of vRAM in StyleGAN2) and a large number of images for training. This implementation allowed to train from scratch on a NVIDIA GeForce RTX 2070 SUPER and a NVIDIA GeForce GTX 1050-Ti, obtaining good results also on a small dataset. The structure of FastGAN comprehends a Skip-Layer channel-wise Excitation (SLE) module and a self-supervised Discriminator trained as a feature-encoder.
We will show, through various experiments, that this GAN outperforms StyleGAN2 in terms of computational requirements and training time.

1 Introduction

State-of-the-art (SOTA) GANs, despite having a lot of usage (implications) in real life applications, such as photo editing, diseases diagnosis, image translation and artistic creation, their high cost in terms of computational power and dataset size made their usage very limited in contexts with a very small computational budget. More specifically, there are three main problems afflicting GANs' training:

- *Accelerate training*: this problem has been approached from various perspectives, but this brought only very limited improvements in training speed, while not enhancing quality within the shortened training time;
- *High resolution training*: this made GAN much harder to converge, since the increased model parameters lead to a more rigid gradient flow to optimize G, and since the target distribution made by images at high resolution is super sparse. There was some approaches trying to solve this problem, but they led to a slightly greater computational cost, consuming more GPU memory and requiring more training time;
- *Stabilize training*: Mode-collapse on generator G is a big challenge when training GANs, given fewer training samples and lower computational power and budgets (smaller batch-size). D is unable to provide consistent gradients to train G, since is more likely to be overfitting on the datasets. There was, also here, approaches that tryed to solve this problem, but they have limited using scenario and image domain, and worked only on low resolution images with unlimited computing resources.

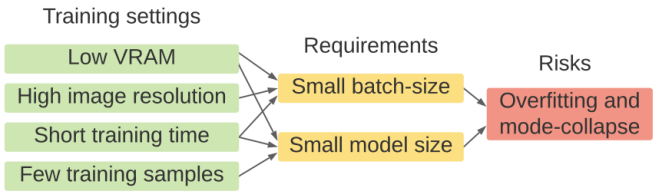


Figure 1: FastGAN training challenges.

In some situations a possible way to avoid these problems was the transfer-learning using pre-trained models, but this solution had the disadvantage of the lack of guarantee to find dataset compatible with the pre-training. Another way was fine-tuning, but this gave even worse results in terms of performance. Those approaches were not took in consideration by people who wanted to train their model from scratch, in order to avoid biases typic of the fine-tuned pre-trained models; in other cases there were the necessity to train models with datasets containing less than 100 images. A possible workaround for the situation of a small dataset was dynamic data-augmentation, but the cost of SOTA models remained very high.

This paper aims to learn an unconditional GAN requiring, at the same time, low computational power and small datasets for training. In order to do this, FastGAN has a "fast-learning" generator G and a discriminator D able to continuously return signals very useful to train G. FastGAN reaches the above objectives based on the following two techniques:

- **Skip-Layer channel-wise Excitation module**: revises channel responses on high-scale feature-maps using low-scale activations and allows to reach a faster training using a more robust gradient flow throughout the model weights;
- **Self-supervised discriminator D trained as a feature-encoder with an extra decoder**: this discriminator is forced to learn a feature-map that covers more regions from an image in input; in this way it gives richer signals in order to train G. It has been shown that auto-encoding is the best self-supervision strategy for D.

2 Method

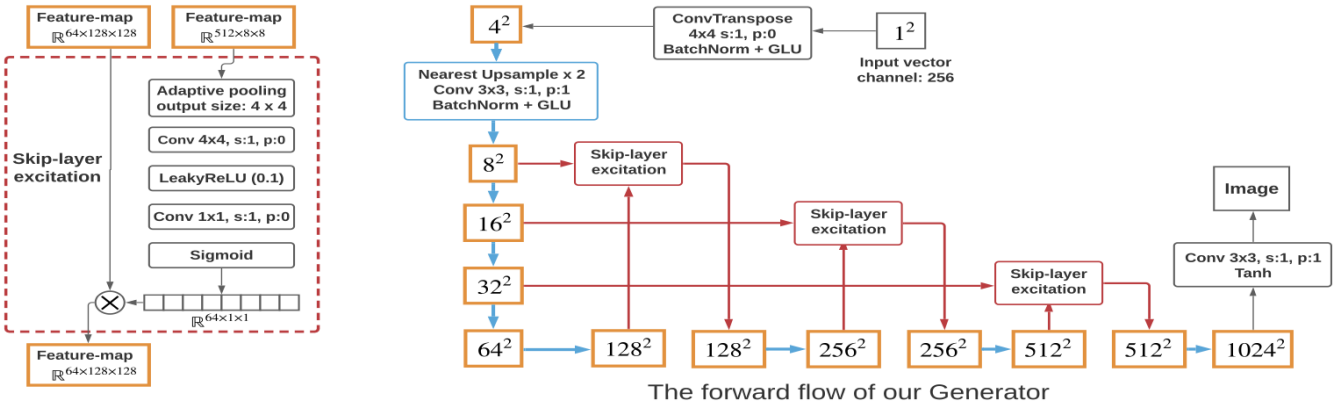


Figure 2: On the left is represented the structure of the skip-layer excitation, on the right is represented the structure of the generator G. Feature-maps are the yellow boxes, up-sampling structures are represented by blue boxes and blue arrows, SLE modules are the red boxes.

In order to optimize GANs with respect to the SOTA models, in FastGAN is adopted a lightweight model, using a single convolutional layer on each resolution in the generator G; moreover, on the high resolutions in both generator G and discriminator D, only three channels for the convolutional layers are applied in input and output. As said before this structure helps making FastGAN faster to train, remaining, at the same time, strong on small datasets.

2.1 Skip-Layer Channel-Wise Excitation

In the traditional GANs the generator G required more convolutional layers (and so a deeper model) to synthesize high resolution images, and this led to longer training times. In order to train such a deep model it was used the Residual structure (ResBlock) increasing the computational cost. Traditionally ResBlock implements the skip-connection as an element-wise addition between the activations from different convolutional layers, but with the requirement that the spatial dimensions of the activations are the same. Moreover, previously, skip-connections were only used within the same resolution.

FastGAN, instead, uses a different approach through the two following strategies:

- In order to eliminate the complex computation of convolution, is used multiplication between activations, instead of addition, giving to one side of activations the spatial dimension of 1^2.
- Instead of performing skip-connection only within the same resolution, here is performed between resolutions with a much longer range (8^2 and 128^2, 16^2 and

256^2, 32^2 and 512^2); this follows from the fact that an equal spatial-dimension is no longer required.

Using the above techniques, SLE keeps the advantages of ResBlock with a shortcut gradient flow, while not requiring an extra computation load.

The Skip-Layer Excitation module is defined as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}_{low}, \{\mathbf{W}_i\}) \cdot \mathbf{x}_{high} \tag{1}$$

In function $\mathcal{F}$ there are the operations on $\mathbf{x}_{low}$; module weights to be learned are represented by $\mathbf{W}_i$; $\mathbf{x}$ is the input feature-map of the SLE module, while $\mathbf{y}$ is its output feature-map. As shown in the Figure 2, the SLE module is composed by the feature-map $\mathbf{x}_{low}$ at resolution 8×8 and the feature-map $\mathbf{x}_{high}$ at resolution 128×128.
First of all $\mathcal{F}$ contains an adaptive average-pooling layer that down-samples $\mathbf{x}_{low}$ into 4×4; then there is the convolutional-layer that further down-samples it into 1×1. Moreover non-linearity is modeled by a LeakyReLU, and after that another convolutional-layer is used to let $\mathbf{x}_{low}$ having the same channel size as $\mathbf{x}_{high}$. Finally there is a Sigmoid function that does a gating operation, and then the output from $\mathcal{F}$ is multiplied with $\mathbf{x}_{high}$ along the channel dimension, returning $\mathbf{y}$ with the same shape as $\mathbf{x}_{high}$.

The above Skip-Layer Excitation works between feature-maps that are far away from each other, brings the benefit of the channel-wise feature re-calibration and makes the whole model's gradient flow stronger; moreover it enables the generator G to automatically disentangle the content and style attributes. It has also be shown that, starting from another synthesized sample, and replacing $\mathbf{x}_{low}$ in the Skip-Layer Excitation, the generator G can generate an image in the same style of the new replacing image and with the content unchanged.

2.2 Self-Supervised Discriminator

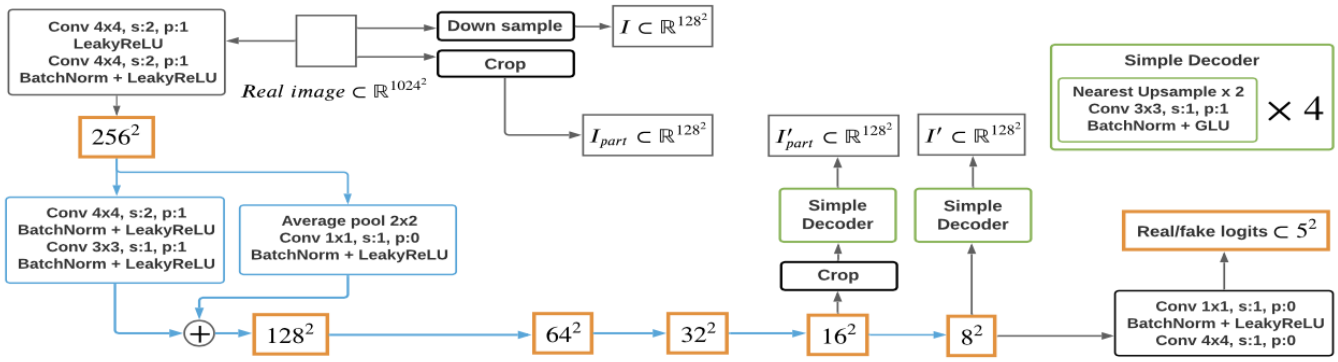


Figure 3: Here is represented the structure and the forward flow of the discriminator. The same residual down-sampling structure is represented by the blue boxes and blue arrows, the same decoder structure is represented by the green boxes.

The discriminator D is handled as an encoder and trained using small decoders. In this way the decoders can give good reconstructions of the images thanks to the image features that the decoder D is forced to extract. Moreover a simple reconstruction loss, which is trained only on real samples, is used to optimize the decoders together with the decoder D:

$$\mathcal{L}_{recons} = \mathbb{E}_{\mathbf{f}\sim D_{encode}(x),\ x\sim I_{real}}[\| \mathcal{G}(\mathbf{f}) - \mathcal{T}(x) \|] \tag{2}$$

More specifically:

- $\mathbf{f}$ represents the intermediate feature-maps from discriminator D;
- $\mathcal{G}$ is the function that contains the processing on $\mathbf{f}$ and the decoder;
- $\mathcal{T}$ is the function that represents the processing on the sample x from the real images I_{real}.

As shown in Figure 3, in the self-supervised discriminator D two decoders are employed, on two scales, for the feature-maps $\mathbf{f}_1$ on 16^2 and $\mathbf{f}_2$ on 8^2. In order to output images at the resolution of 128×128, the decoders consist of four convolutional-layers: this approach causes little extra computations. Then $\mathbf{f}_1$ is cropped, in a random way, at $\frac{1}{8}$ of its height and width; the same crop is made on the real image in order to output I_{part}, while the real image is downsampled in order to output I. After this, starting from, respectively, the cropped $\mathbf{f}_1$ and $\mathbf{f}_2$, I'_{part} and I' are generated by the decoders. Lastly, in order to minimize the loss in the equation 2, the discriminator D and the decoders are trained together, by matching I'_{part} to I_{part} and I' to I.

The discriminator D, thanks to the above described training, extracts a better representation from the input, that includes both the detailed textures from $\mathbf{f}_1$ and

the overall compositions from $\mathbf{f}_2$. Here, as a method for self-supervised learning, is used the auto-encoding approach, which improves model's robustness, generalization ability and performances. Moreover, this auto-encoding training regards only the regularization of the discriminator D, and not involves the generator G in any way.

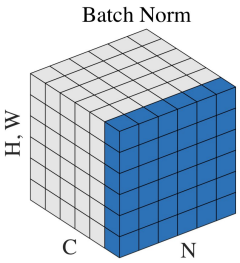
In order to train, iteratively, D and G, in FastGAN is used the hinge version of the adversarial loss, that is the fastest in terms of performances:

$$\mathcal{L}_D = -\mathbb{E}_{x\sim I_{real}}[min(0, -1 + D(x))] - \mathbb{E}_{\hat{x}\sim G(z)}[min(0, -1 - D(\hat{x}))] + \mathcal{L}_{recons} \quad (3)$$

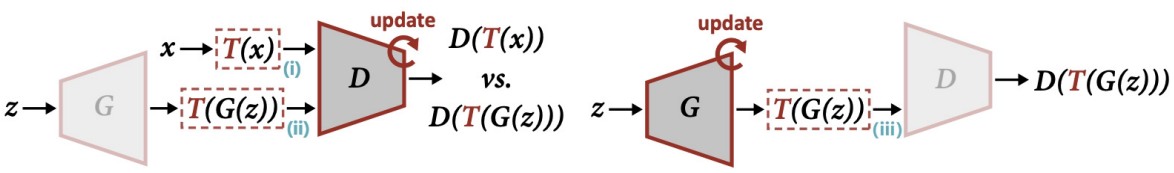
$$\mathcal{L}_G = -\mathbb{E}_{z\sim\mathcal{N}}[D(G(z))] \quad (4)$$

The model is built starting from a baseline model with the two above proposed techniques (SLE and self-supervised discriminator). The baseline model is made of the following techniques (based on Deep Convolutional GAN):

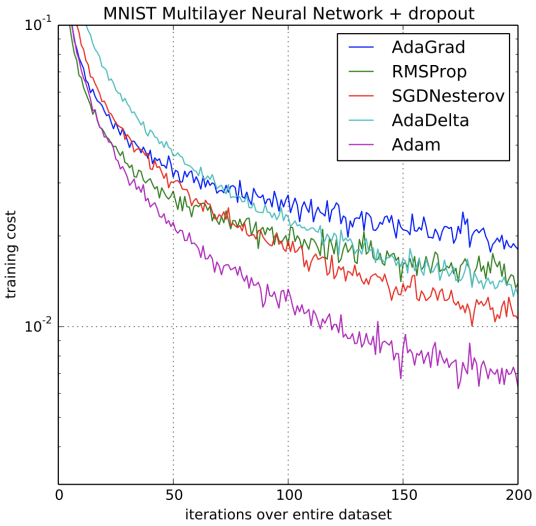
- *Batch-normalization:* aims to avoid unstable gradients, to reduce internal covariate shift, and so to accelerate the training of deep neural networks. All of this is achieved by a normalization step that fixes the means and variances of layer inputs. This has also a beneficial effect on the gradient flow through the network, reducing dependence of gradients on the scale of the parameters or of their initial values. This allows to use much higher learning rates without risk of divergence. Ideally batch-normalization should also normalize each layer based on the entire dataset; instead it normalizes using mini-batch statistics.



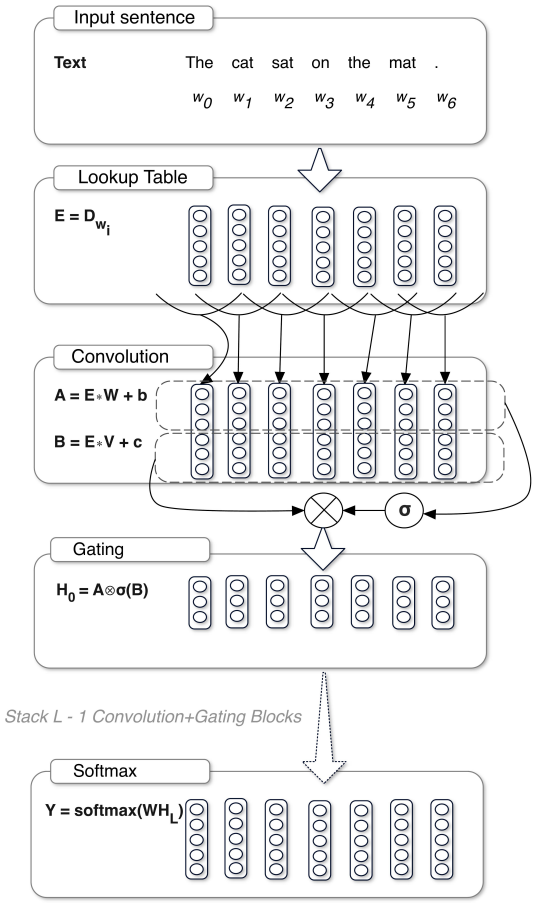
- *Differentiable-augmentation:* is a set of differentiable image transformations used to augment data during training. The transformations are applied to both real and generated images. It enables the gradients to be propagated trough the augmentation back to the generator, regularizes the discriminator without manipulating the target distribution and maintains the balance of training dynamics. There are three choices of transformation: Translation, CutOut and Color.



- *ADAM optimization on G:* ADAM is a replacement optimization algorithm for stochastic gradient descent for training deep learning models; it combines the best properties of AdaGrad and RMSprop algorithms, to provide an optimization algorithm that can handle sparse gradients on noisy problems.
  In other words, ADAM can be looked at as a combination of both RMSprop and Stochastic Gradient Descent (SGD) with momentum: it uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.



- *GLU instead of ReLU in G:* Gated Linear Unit computes $GLU(a, b) = a \otimes \sigma(b)$, where b is the gate that controls what information from a is passed up to the following layer. This gating mechanism allows selection of features that are important for predicting the next feature; it also provides a mechanism to learn and pass along just the relevant info.



3 Implementation and metrics choices

3.1 Implementation

Traditional GANs always required a lot of computational power in order to obtain good results.
Conversely, FastGAN has the big advantage of obtaining good results in a reasonable time also on non server-level hardware. In fact we got decent results on NVIDIA GeForce RTX 2070 SUPER and, specially, on NVIDIA GeForce GTX 1050-Ti, respectively equipped with 8GB and 4GB of VRAM. More specifically we were able to train the GAN for 1000/2000 epochs: this took $\sim$ 10hrs on the 2070 SUPER, and $\sim$ 24hrs on the 1050-Ti. Those are very good results because, with the traditional GANs, mostly on the 1050-Ti, probably it would have been impossible even just letting the train start.

Regarding the code implementation, while the original paper proposed a PyTorch version, we decided to reimplement it using TensorFlow.

Since the training needed a lot of time to execute, we used a checkpoint approach: in this way we were able to save the training status and resume the training whenever we wanted. More specifically we saved the current epoch number and the current weights of both the generator and the discriminator.

3.1.1 Requirements

- Python 3.8.x
- TensorFlow 2.x
- NVIDIA CUDA 11
- NVIDIA cuDNN 8

The above requirements are only the basic ones; in order to replicate the environment that we used, we have created the "environment.yml" file that includes all the packages needed. We were able to execute the project both on Windows and MacOS, but we ran the training only on Windows, in order to exploit the performance boost given by the CUDA hardware.

3.2 Metrics

Differently from the original paper, we used only Fréchet Inception Distance (FID) which is a metric that calculates the distance between feature vectors calculated for real and generated images. The score summarizes how similar the two groups

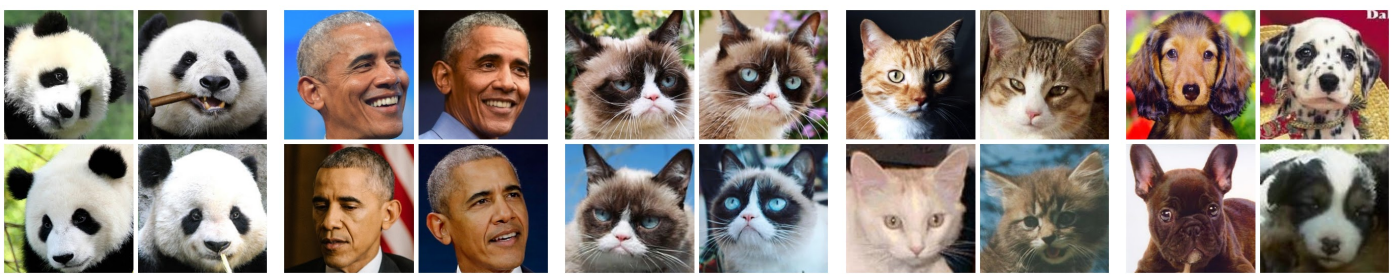
are in terms of statistics on computer vision features of the raw images. Lower scores indicate the two groups of images are more similar, or have more similar statistics, with a perfect score being 0.0 indicating that the two groups of images are identical. The FID score is used to evaluate the quality of images generated by generative adversarial networks, and lower scores have been shown to correlate well with higher quality images. More specifically we calculated FID during training for each epoch, and we used it in order to save model's weights only when it got better (lower) with respect to the previous best saved FID score: this helps in maintaining high performances.

4 Dataset

The authors of the original paper used multiple datasets on both 256×256 and 1024×1024 resolutions. More specifically they used:

- Animal-Face Dog and Cat
- 100-Shot-Obama
- Panda
- Grumpy-cat
- Flickr-Face-HQ
- Oxford-flowers
- Art paintings from WikiArt
- Photographs on natural landscape from Unsplash
- Pokemon
- Anime face
- Skull
- Shell

Since our hardware was not powerfull enough to do the training on photos with a resolution of 1024×1024, we decided to use only datasets of resolution 256×256, and, more in details, we used the following datasets: Panda, Obama, Animal-Face Dog and Cat, Grumpy-cat.



5 Experiments

	NVIDIA RTX 2070 SUPER				**NVIDIA GeForce GTX 1050-Ti**
Dataset	**Panda**	**Dogs**	**Obama**	**Grumpy Cat**	**Cats**
Number of images	100	389	100	100	159
Resolution	256×256	256×256	256×256	256×256	256×256
Batch size	4	8	8	8	4
Epochs	1000	1400	1700	1000	1800
Epoch Time	∼15s	∼35s	∼15s	∼14s	TODO
FID	∼28	∼35	∼75	∼129	∼108

Table 1: title

6 NOME

7 NOME

8 NOME

9 NOME