Deep Convolutional Generative Adversarial Networks

Théo VIEL

MVA, Ecole des Ponts ParisTech

theo.viel@eleves.enpc.fr

Abstract. Generative Adversarial Networks (GANs) generate a an image following a learned distribution, by using a random vector in a latent space. GAN have proven to be powerful but difficult to handle synthesis tools. The Deep Convolutional Generative Adversarial Network (DCGAN) fully convolutional GAN which achieved state of the art results when it was released in 2016. Several important implementation details are given in the DCGAN paper which improve convergence. In this project, we propose to implement a DCGAN to generate dogs, using a subset of the ImageNet dataset. We aim at producing good quality outputs, and follow the implementation techniques provided by the authors of the DCGAN paper. The code is publicly available at `github.com/TheoViel/DCGAN_Dogs`.

Keywords: Generative Adversarial Networks · Deep Learning · Convolutional Neural Networks.

1 Introduction

1.1 Generative Adversarial Networks

GANs were introduced by Ian Goodfellow et Al. in [4]. The idea is to jointly train two neural networks on a dataset : a generator and a discriminator.

- The generator $\mathcal{G}$ aims at generating samples of good quality similar to those in the dataset by learning its distribution. It takes as input a noise vector z that will be generated from a distribution p_z (classically, $p_z = \mathcal{N}(0, 1)$), and outputs a fake image.
- The discriminator $\mathcal{D}$ aims at distinguishing images that were generated by the generator from images of the dataset. It will either take as input images generated by the generator, or real images following the dataset distribution p_{data}, and return the probability that the chosen sample is real.

Then, the discriminator is a simple image classifier that will learn the probability that the image is real, using classically the binary cross entropy (BCE). On the

other hand, the generator will use the discriminator to assess the quality of its generated samples, and aim at producing images that the discriminator estimates to be from the original data, i.e. maximise the output probability.
Following this principle, the following loss is jointly optimized :

$$\min_G \max_D \quad \mathbb{E}_{x\sim p_{\text{data}}}[\log \mathcal{D}(x)] \; + \; \mathbb{E}_{z\sim p_z}[\log\left(1-\mathcal{D}(\mathcal{G}(z))\right)] \tag{1}$$

Where D is the set of parameters of the discriminator and G the one of the generator. In the case of neural network, we aim at learning them using back-propagation.
GAN training can be seen as a two player minimax game, and these types of games are usually unstable, hence, there are little guarantees on the convergence of GANs.

1.2 Deep Convolutional Generative Adversarial Networks

Convolutional Neural Networks (CNNs) have proven to be very effective to tackle supervised computer vision problems such as image classification and object detection. The authors of [16] aim at leveraging the power of convolutions for an unsupervised task that is image generation. They propose several changes in architecture, in order to make GANs perform better.

The main idea is to get rid of most all of the fully connected layers, as well as the pooling ones. In fact, strided convolutions can replace pooling operations whereas transposed convolutions can replace upsampling operations. When the image size is known, correctly adjusting the convolution kernel sizes and padding as well as using a last convolution with only 1 filter allows to get rid of a fully connected head. Convolutions can also be plugged on the input noise when when considering it as a tensor of size $d \times 1 \times 1$.
In addition, the authors leverage batch normalisation [7], an useful modification to improve the convergence of neural networks. Batch normalisation helps the activations stay of reasonable mean and amplitude, and usually fastens the training of CNNs. The authors proposed to add batch normalisation to all layers except for the first layer of the discriminator and the last layer of the generator, for which it was too constraining.
Sigmoid and tanh non-linearities have been shown not to work very well for deeper architectures, because of gradient vanishing issues. Following recent trends in CNNs, the authors use ReLU [14] for the generator and LeakyReLU [11] with a slope of 0.2 for the discriminator. These activations are represented figure 1.

Indeed, the combination Convolution + Batchnorm + ReLU has proven to be really effective in computer vision architectures, for instance in the ResNet [5] one. In this work, we will re-implement the DCGAN architecture as well as the training algorithm.

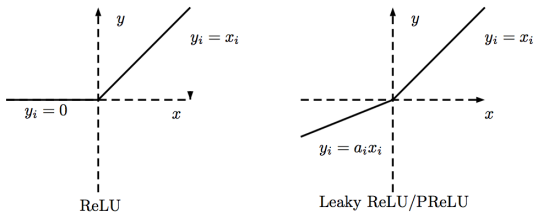


Fig. 1: ReLU [14] and LeakyReLU [11] activations.

1.3 Going Further

After achieving reasonable results with the DCGAN architecture, I will try experimenting with techniques proposed to improve the performance of the DCGAN model.
[18] proposes modifications to improve the convergence of GANs, and I will check the influence of two ideas. The first one is one sided label smoothing, which modifies the positive target of the cross-entropy loss in the positive case, i.e. 1, to a lower value. The next one is feature matching, that modifies the training objective of the generator : it now aims at matching the distributions of features extracted by the discriminator on real and fake images.
I will also take a further look in the learning rate policy, the one proposed by the author is rather simple (batch size 128 and a 2×10^{-4} learning rate). Stability could be improved using a scheduler, and the generator could benefit from using a learning rate higher than the discriminator, for its task is usually harder.

In addition, Conditional GANs [12] will be looked into. They consist of adding the label information in the GAN training, and have proven to be useful to help convergence and diversity. The modification consists of using the label as input for both generator and discriminator, and using this information to generator image of the corresponding class.
Similarly, I will implement the idea behind Auxiliary Classification GANs [15]. The goal is the same as Conditional GANs: help convergence and improve diversity. Compared to Conditional GANs, the generator is unchanged, but the discriminator is no longer given information about the label and now aims at predicting it.

2 Framework

2.1 Data

To train our models, we use a subset of the ImageNet [3] dataset, containing 120 dog breeds [8]. This dataset contains 20,580 images and the associated bounding boxes. Bounding boxes allow to ease a bit the task for GANs as they reduce the variance in the images when used for cropping. I work with 64×64 images, which allows to run experiments on my hardware [1] in approximately 3 hours. The only

[1] a single Nvidia RTX 2080 Ti

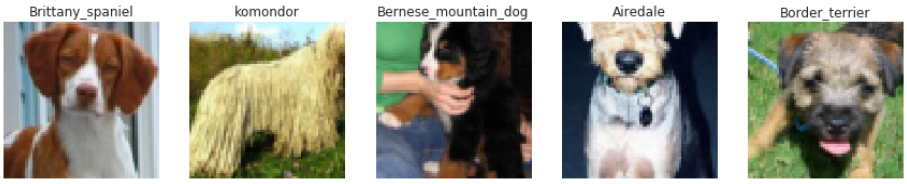


Fig. 2: Pre-processed training data samples.

transformations applied are horizontal flipping and re-scaling the images between -1 and 1. Indeed, the tanh activation on the last layer of the generator allows for images in this range as well. Training samples are represented 2 and we can already notice the difficulty of the task. Images present a much higher variance than those of the CelebA [10] dataset presented in the DCGAN paper [16]. Already, DCGAN struggled to produce high quality results using this dataset, therefore I expect the generated dogs to be of poor quality.

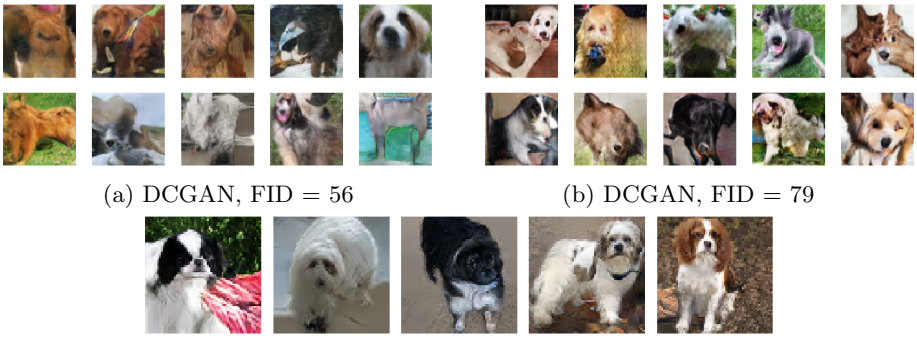
2.2 Fréchet Inception Distance (FID)

Introduced in [6], the FID aims at assessing the quality of artificially generated samples. In fact, it is very difficult to compare the performance of generative models, and the FID is one of the most successful attempts.
It relies on the Fréchet distance between two multivariate Gaussians $X_1 \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $X_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$, which is defined by :

$$d^2(X_1, X_2) = \|\mu_1 - \mu_2\|^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2\sqrt{\Sigma_1 \Sigma_2}).$$

In the case of the FID, we use the activations of an hidden layer (layer `pool_3`) of the Inception [19] network pretrained on ImageNet [3]. X_1 will then be the activations on the real data and X_2 on the fake data. Its authors have shown that this metric correlates well with human observation.
However, this metric has some drawbacks : it is slow to compute, complicated to implement and is really far from being a perfectly accurate indicator of GAN performance. Still, we will be using its official implementation [2] to report model performances. Following guidelines, we generate 10,000 images to evaluate them. Indeed, using fewer will result in a performance drop.
In order to further assess the viability of the FID, we show figure 3 dogs generated with two DCGANs with different FID scores. The 56 scoring model (fig. 3a) seems to have higher quality images overall but it is quite hard to tell. One default of the 79 scoring model (fig 3b) is poor looking textures on the middle 3 images of the first line for instance. Currently, state of the art models such as BigGAN [1] achieve a FID of ≈ 15 on the dog generation task, with dogs presenting a great variety (see fig. 3c) , and being much more realistic than those I hope to generate with a DCGAN.

[2] See `https://github.com/bioinf-jku/TTUR`



(a) DCGAN, FID = 56 (b) DCGAN, FID = 79

(c) BigGAN, FID = 13

Fig. 3: Example of generated dogs for models with different FIDs

3 Implementing DCGAN

3.1 DCGAN Discriminator

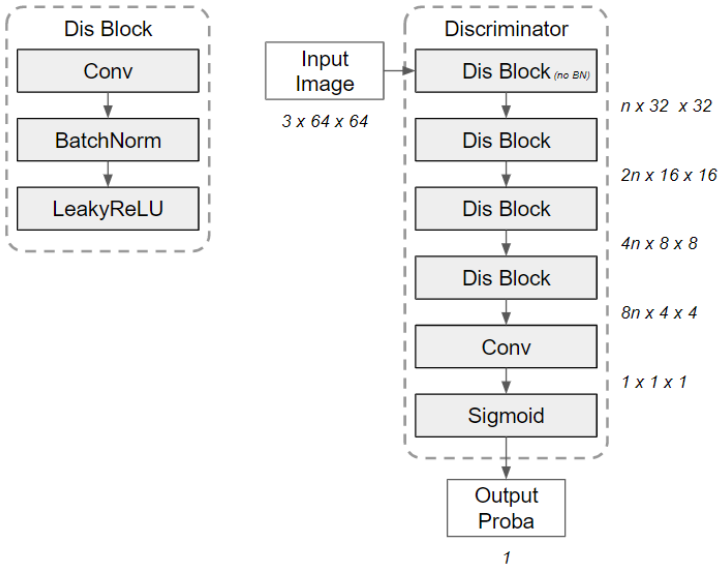


Fig. 4: Architecture of the DCGAN discriminator.

The discriminator is based on Convolution + Batchnorm + LeakyReLu blocks, its architecture is reported figure 4. In this block, the convolution has a kernel size of 4, a stride of 2 and a padding of 1 which results is the width and height being divided by two. The first convolution is chosen to be have n filters and we

double this number each time we go deeper in the architecture. The authors also mentioned that using batch normalisation in the first block resulted in instability, so we remove it. Finally, the last convolution is chosen to have a kernel size of 4, a stride of 1 and no padding, in order to obtain a $1 \times 1 \times 1$ output corresponding to the probability of the sample being real.

3.2 DCGAN Generator

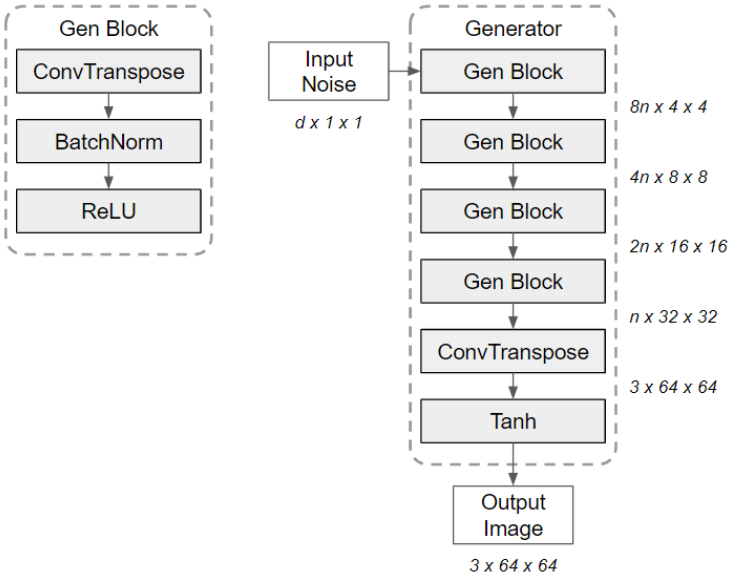


Fig. 5: Architecture of the DCGAN generator.

The generator looks similar to the discriminator, and is based on Transposed Convolution + Batchnorm + ReLu blocks, its architecture is reported figure 5. In this block, the transposed convolution has a kernel size of 4, a stride of 2 and a padding of 1 (and 0 for the first one) which results is the width and height being multiplied by two. The last transposed convolution is chosen to be have n filters and we double this number each time we go close to the input in the architecture. Finally, the last transposed convolution is chosen to have a kernel size of 4, a stride of 2 a padding of 1 as well, but only uses 3 channels in order to obtain a $3 \times 64 \times 64$ output image. A tanh activation is used in order to make sure the image pixel values are in $[-1, 1]$.
Here again, the weights are initialised by following a gaussian law $\mathcal{N}(0, 0.02)$. The authors report using $n = 128$ in one of their experiments but this parameter will be quickly looked into I will also take a look at the influence of removing batch normalisation, as well as changing the activation functions.

3.3 Training

We report in algorithm 1 the key ideas to implement the training of a DCGAN. The goal is to optimize the objective (1), and we use the binary cross-entropy (BCE) to do so :

$$\text{BCE}(x, y) = -y \log(x) - (1 - y) \log(1 - x)$$

Where $y \in \{0, 1\}$ is the label and $x \in [0, 1]$ the probability. Actually, the objective is slightly different in practice, for stability issues. The objective of the discriminator is :

$$\min_{D} \quad - \mathbb{E}_{x \sim p_{\text{data}}}[\log(\mathcal{D}(x)] \; - \; \mathbb{E}_{z \sim p_z}[\log\left(1 - \mathcal{D}(\mathcal{G}(z))\right)] \tag{2}$$

And the objective of the generator is :

$$\min_{G} \quad - \mathbb{E}_{z \sim p_z}[\log \mathcal{D}(\mathcal{G}(z))] \tag{3}$$

Indeed, the term in $-\log(1 - x)$ corresponds to the BCE loss when the label is 0, and the term $-\log(x)$ when the label is 1.
Classically, we use stochastic gradient descent to train the neural network. As in [16], the optimizer Adam [9] with $\beta = (0.5, 0.999)$ is used.

Algorithm 1 DCGAN training

for epoch in epochs **do**
  for $\mathcal{I}_{\text{real}}$ in batch(data) **do**
    # Update discriminator
    $p_{\text{real}} = \mathcal{D}(\mathcal{I}_{\text{real}})$
    $\text{Loss}_{\text{real}} = \text{BCE}(p_{\text{real}}, 1)$
    sample $z \sim \mathcal{N}(0, 1)$
    $\mathcal{I}_{\text{fake}} = \mathcal{G}(z)$
    $p_{\text{fake}} = \mathcal{D}(\mathcal{I}_{\text{fake}})$
    $\text{Loss}_{\text{fake}} = \text{BCE}(p_{\text{fake}}, 0)$
    Back-propagate $\text{Loss}_{\text{fake}} + \text{Loss}_{\text{real}}$

    # Update generator
    sample $z \sim \mathcal{N}(0, 1)$
    $\mathcal{I}_{\text{fake}} = \mathcal{G}(z)$
    $p_{\text{fake}} = \mathcal{D}(\mathcal{I}_{\text{fake}})$
    $\text{Loss} = \text{BCE}(p_{\text{fake}}, 1)$
    Back-propagate Loss
  end for
end for

3.4 First Experiments

In this subsection, we review the influence of the main parameters of the training of the GAN and of the architecture. Results are reported after 200 epochs, and we provide appendix A an overview of the generated dogs.
We report table 1 the performances for different learning rates. In this case, we used $d = 100$ and $n = 128$. As the task of the generator is harder than the task of the discriminator, using a higher learning rate for the generator proved to be effective. This setup achieves a FID on 56.5, which is 10 points better than the score reached using the parameters described in the original paper [16].
The influence of n and d is reported table 2. Best results were achieved using $d = 100$ and $n = 128$. $n = 128$ takes 4 times longer than the $n = 64$ model, resulting in a training of ≈ 3 hours. Therefore, I did not experiment with $n = 256$ although performances might benefit from it, as the training would take 12 hours. Experiments were conducted with the previously obtained learning rates.

Table 1: Influence of the learning rate and batch size.

Batch size	Discriminator lr	Generator lr	**FID**
128	2×10^{-4}	2×10^{-4}	67.9
64	1×10^{-3}	1×10^{-3}	60.2
64	5×10^{-4}	5×10^{-4}	66
64	5×10^{-4}	1×10^{-3}	**56.5**

Table 2: Influence of the latent dimension d and model size n.

d	n	epoch length	**FID**
100	128	44s	**56.5**
256	128	44s	62.2
100	64	11s	78

3.5 Scheduling

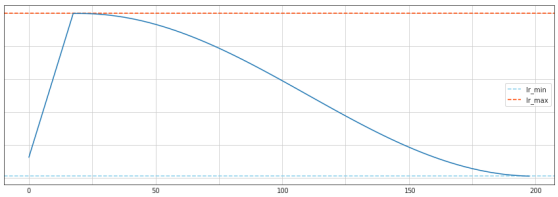


Fig. 6: Cosine learning rate scheduling.

Finally, we improve the learning rate policy by adding a cosine scheduling. This scheduling is illustrated figure 6, with the first epochs being used for warm-up. In our case, warm-up didn't prove to be useful so the learning rate directly starts at *lr_max*. Results are reported table 3, where the scheduling is described on the form $lr\_max \rightarrow lr\_min$. Best results were achieved with $lr\_min = 10^{-5}$, and with $lr\_max = 2 \times 10^{-5}$ for the generator and $lr\_max = 1 \times 10^{-5}$ for the discriminator, hence keeping the differentiated learning rate. The influence on the training stability of adding the scheduling is reported appendix A.

Table 3: Adding a learning rate scheduler

Discriminator lr	Generator lr	**FID**
$5 \times 10^{-4} \rightarrow 1 \times 10^{-5}$	$1 \times 10^{-3} \rightarrow 1 \times 10^{-5}$	58
$1 \times 10^{-3} \rightarrow 1 \times 10^{-5}$	$2 \times 10^{-3} \rightarrow 1 \times 10^{-5}$	**55.5**
$1 \times 10^{-3} \rightarrow 1 \times 10^{-4}$	$2 \times 10^{-3} \rightarrow 2 \times 10^{-4}$	57.8
$2 \times 10^{-3} \rightarrow 2 \times 10^{-5}$	$4 \times 10^{-3} \rightarrow 4 \times 10^{-5}$	58.7

4 Feature matching

4.1 Algorithm

Proposed in [18], feature matching is a new objective for the generator that is similar to the idea behind the FID: we want the generator to generate images that has features matching those of the real data. This time, the idea is to use the discriminator to compute those features. Let us note $f_{\mathcal{D}}$ the fact of using the discriminator to output hidden features, the new objective is :

$$\min_{G} \quad \|\mathbb{E}_{z \sim p_z}[f_{\mathcal{D}}(\mathcal{G}(z))] - \mathbb{E}_{x \sim p_{\text{data}}}[f_{\mathcal{D}}(x)]\|_2^2 \tag{4}$$

And we can adapt the training algorithm 1 to use feature matching, by using the mean squared error (MSE), see algorithm 2.

Algorithm 2 DCGAN training with Feature Matching

for epoch in epochs **do**
  for $\mathcal{I}_{\text{real}}$ in batch(data) **do**
    # Update discriminator
    [...] (Same as algo. 1)

    # Update generator
    sample $z \sim \mathcal{N}(0, 1)$
    $\mathcal{I}_{\text{fake}} = \mathcal{G}(z)$
    $x_{\text{fake}} = f_{\mathcal{D}}(\mathcal{I}_{\text{fake}})$
    $x_{\text{real}} = f_{\mathcal{D}}(\mathcal{I}_{\text{real}})$
    Average x_{fake} and x_{real} over the batch axis
    Loss = MSE($\bar{x}_{\text{fake}}, \bar{x}_{\text{real}}$)
    Back-propagate Loss
  end for
end for

4.2 Implementation

We represent figure 7 the chosen approach to implement feature matching. It consists of modifying the last convolution to be of *nb_ft* channels. The feature

output $f_{\mathcal{D}}$ will then be the flattened output of this convolution layer. As we still need the discriminator to output probabilities for the training of the discriminator, we plug a dense layer of size 1 with a sigmoid activation to retrieve it. Note that is possible to keep this architecture fully convolutional by using a convolution of 1 filter of kernel size 1, this approach being equivalent.

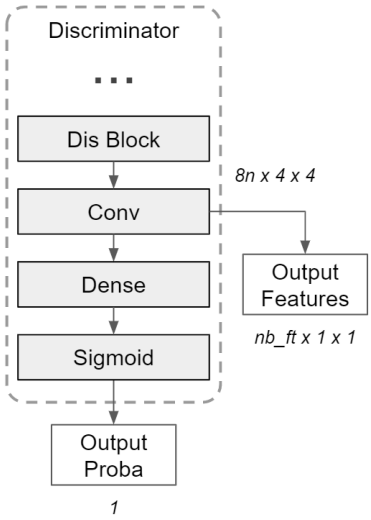


Fig. 7: Adding feature matching to the DCGAN discriminator.

4.3 Results

We reuse the previously obtained setup that achieved the best performance, i.e. $d = 100$, $n = 128$, a learning rate of 10^{-3} for the discriminator and 2×10^{-3} for the generator, with cosine scheduling. We report table 4 the change in performance and notice that feature matching does not really impact our results. Furthermore, since feature matching requires one additional forward pass in the discriminator, run times are longer hence this modification will not be kept afterwards.

Table 4: Influence of adding feature matching to our DCGAN.

Feature Matching	**FID**
Without	**55.5**
With	55.9

5 Conditional GANs

5.1 Algorithm

Proposed in [12], Conditional GANs improve GANs by adding label information y to the classical GAN objective (eq .1) :

$$\min_G \max_D \quad \mathbb{E}_{x\sim p_{\text{data}}}[\log \mathcal{D}(x|y)] \; + \; \mathbb{E}_{z\sim p_z}[\log\left(1-\mathcal{D}(\mathcal{G}(z|y))\right)] \tag{5}$$

In our case, we use the dog breed for conditioning, and both the generator and discriminator will take as additional input a label y. When using real data, y is the label associated to the image. When using fake data, the label is sampled uniformly in $[\![0, c]\!]$ where c is the number of classes. We report 3 the modifications to the training algorithm made to implement the training of a Conditional GAN.

Algorithm 3 Conditional DCGAN training

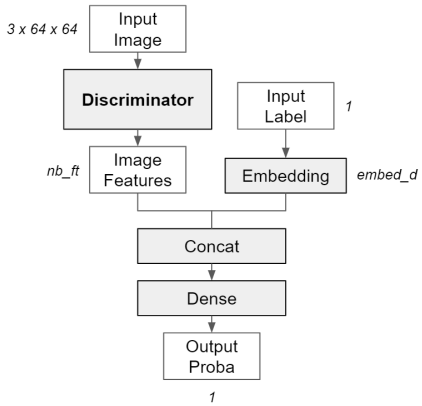
for epoch in epochs **do**
  for $\mathcal{I}_{\text{real}}, y_{\text{real}}$ in batch(data) **do**
    # Update discriminator
    $p_{\text{real}} = \mathcal{D}(\mathcal{I}_{\text{real}}, y_{\text{real}})$
    $\text{Loss}_{\text{real}} = \text{BCE}(p_{\text{real}}, 1)$
    sample $z \sim \mathcal{N}(0, 1)$
    sample $y_{\text{fake}} \sim \mathcal{U}([\![0, c]\!])$
    $\mathcal{I}_{\text{fake}} = \mathcal{G}(z, y_{\text{fake}})$
    $p_{\text{fake}} = \mathcal{D}(\mathcal{I}_{\text{fake}}, y_{\text{fake}})$
    $\text{Loss}_{\text{fake}} = \text{BCE}(p_{\text{fake}}, 0)$
    Back-propagate $\text{Loss}_{\text{fake}} + \text{Loss}_{\text{real}}$

    # Update generator
    sample $z \sim \mathcal{N}(0, 1)$
    sample $y_{\text{fake}} \sim \mathcal{U}([\![0, c]\!])$
    $\mathcal{I}_{\text{fake}} = \mathcal{G}(z, y_{\text{fake}})$
    $p_{\text{fake}} = \mathcal{D}(\mathcal{I}_{\text{fake}}, y_{\text{fake}})$
    $\text{Loss} = \text{BCE}(p_{\text{fake}}, 1)$
    Back-propagate Loss
  end for
end for

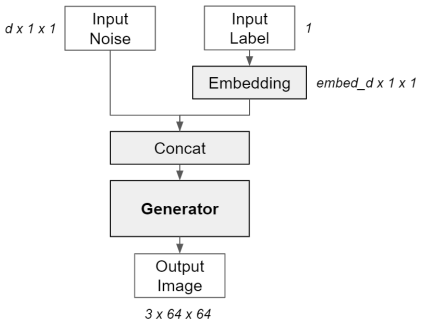
5.2 Implementation

We represent figure 8 the modifications made to the DCGAN architecture in order to introduce conditioning. I went for an approach considering of using the class information after the convolutional network, such as in [17]. Originally, the authors of [12] feed the class information through the CNN but I was not successful with this approach. Both discriminator and generator then take as

additional input the class associated to the image we want to generate or score. In both cases, it is fed to an embedding layer of size *embed_d*.
In the case of the discriminator (fig. 8a), it is used to extract features of size *nb_ft*, similarly to the feature matching approach (fig. 7). Those features are concatenated with the embedding output and fed to a final dense layer used to obtain the probability of the image being real.
In the case of the generator (fig. 8a), the output of the embedding layer is concatenated to the latent noise, and the rest of the architecture does not change. Another option is to concatenate the one hot encoding of the class label to the input noise, therefore removing the need of the embedding layer.



(a) Conditional Discriminator



(b) Conditional Generator

Fig. 8: Adding conditioning to the DCGAN architecture.

5.3 Projection Discriminator

The approach used to add conditioning in the state of the art BigGAN [1] is taken from [13] and differs from the one I used. In order to improve the results, I

implemented their approach. The generator is unchanged and they directly use the concatenation of the noise and of the one-hot encoded class label.
The discriminator (fig. 9) differs quite a bit. The idea is to project the class features (extracted by an embedding layer) on the image feature (extracted by the DCGAN discriminator), by using a scalar product. We then obtain a similarity score between the class and the image. This measure is added to a score based purely on the image, extracted by a convolution with a kernel size 1 and 1 filter.

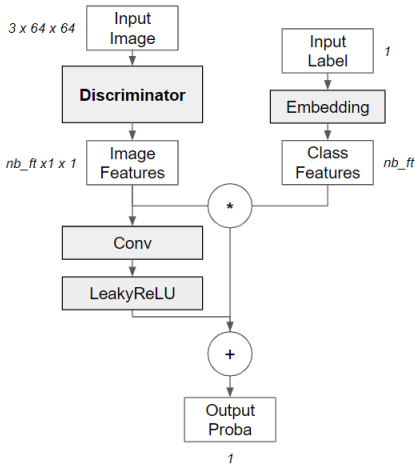


Fig. 9: Architecture of the projection discriminator.

5.4 Results

Once again, we reuse the previously obtained setup that achieved the best performance. We report table 5 the results, and do not notice an increase in performance. In fact, neither of the models managed to correctly learn the dog races. It appears that the loss of the generator reaches 0 too fast, the generator does not seem to be powerful enough for the task. Also, batch normalisation could also be too constraining in this case, and switching to conditional batch normalisation [2] could help here.

Table 5: Influence of adding conditioning to our DCGAN.

Conditioning	*nb_ft*	**FID**
Without		**55.5**
With	8	60.3
Projected	1024	62.8

6 Auxiliary Classifier GANs

6.1 Algorithm

Proposed in [15], Auxiliary Classifier GANs (ACGANs) lay on the same idea as Conditional GANs, but differ on how the discriminator handles labels. The label is not fed to the discriminator, but aims at predicting it using and auxiliary head to do classification. ACGANs uses the combination two objectives, the usual GAN one with conditioning on the generator:

$$\min_G \max_D \quad \mathbb{E}_{x\sim p_{\text{data}}}[\log \mathcal{D}(x)] \; + \; \mathbb{E}_{z\sim p_z}[\log\left(1-\mathcal{D}(\mathcal{G}(z|y))\right)] \tag{6}$$

And the classification one, where we note $\mathcal{D}_c$ the use of the discriminator as a classifier :

$$\max_G \max_D \mathbb{E}_{x\sim p_{\text{data}}}[\log \mathcal{D}_c(x)=y] \; + \; \mathbb{E}_{z\sim p_z}[\log \mathcal{D}_c(\mathcal{G}(z|y))=y] \tag{7}$$

This time, only the generator is conditioned on y. Once again, we use the dog breed for conditioning. Classically, we use the cross-entropy loss to model the multi-class problem. Algorithm 4 reports the modifications.

Algorithm 4 Auxiliary Classifier DCGAN training

for epoch in epochs **do**
  for $\mathcal{I}_{\text{real}}, y_{\text{real}}$ in batch(data) **do**
    # Update discriminator
    $p_{\text{real}}, p^c_{\text{real}} = \mathcal{D}(\mathcal{I}_{\text{real}})$
    $\text{Loss}_{\text{real}} = \text{BCE}(p_{\text{real}}, 1) + \text{CE}(p^c_{\text{real}}, y_{\text{real}})$
    sample $z \sim \mathcal{N}(0,1)$
    sample $y_{\text{fake}} \sim \mathcal{U}([\![0, c]\!])$
    $\mathcal{I}_{\text{fake}} = \mathcal{G}(z, y_{\text{fake}})$
    $p_{\text{fake}}, p^c_{\text{fake}} = \mathcal{D}(\mathcal{I}_{\text{fake}})$
    $\text{Loss}_{\text{fake}} = \text{BCE}(p_{\text{fake}}, 0) + \text{CE}(p^c_{\text{fake}}, y_{\text{fake}})$
    Back-propagate $\text{Loss}_{\text{fake}} + \text{Loss}_{\text{real}}$

    # Update generator
    sample $z \sim \mathcal{N}(0,1)$
    sample $y_{\text{fake}} \sim \mathcal{U}([\![0, c]\!])$
    $\mathcal{I}_{\text{fake}} = \mathcal{G}(z, y_{\text{fake}})$
    $p_{\text{fake}}, p^c_{\text{fake}} = \mathcal{D}(\mathcal{I}_{\text{fake}}$
    $\text{Loss} = \text{BCE}(p_{\text{fake}}, 1) + \text{CE}(p^c_{\text{fake}}, y_{\text{fake}})$
    Back-propagate Loss
  end for
end for

6.2 Implementation

We represent figure 10 the modifications made to the DCGAN architecture in order to introduce the auxiliary classification task. The generator is the same as for the Conditional GAN (fig. 8a),
Once again, we use the discriminator to extract features of size *nb_ft*, to which we plug a classification head for the class probability in addition to the one used for the probability of the image being real.

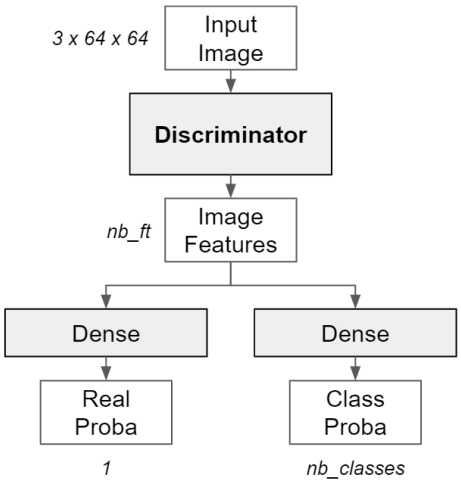


Fig. 10: Adding an auxiliary classifier to the DCGAN discriminator.

6.3 Results

Once again, we reuse the previously obtained setup that achieved the best performance. We report table 6 the results. Even though the results are better than for the Conditional GAN, the standard architecture still performs better. However, this setup appeared to be more robust. For the reported results, I used $nb\_ft = embed\_d = 64$.

Table 6: Influence of adding an auxiliary classifier to our DCGAN.

Auxiliary classifier	**FID**
Without	**55.5**
With	57.9

7 Label smoothing

7.1 Idea

The authors of [18] also proposed to use label smoothing for GANs. This quite old technique was first applied to deep learning in [20], and consists of replacing the 0 and 1 targets of a classifier with smoothed values (e.g. 0.1 and 0.9) to stabilise the training. Indeed, the $y = \log(x)$ function diverges when x is close to 0, which can be a problem for GANs as well.
The authors of [4] state and demonstrate that, when labels are not smoothed, the optimal discriminator is :

$$\mathcal{D}(x) = \frac{p_{\text{real}}(x)}{p_{\text{real}}(x) + p_{\text{fake}}(x)}$$

When smoothing 1 to α and 0 to β, it becomes :

$$\mathcal{D}(x) = \frac{\alpha p_{\text{real}}(x) + \beta p_{\text{fake}}(x)}{p_{\text{real}}(x) + p_{\text{fake}}(x)}$$

However, having such an optimal discriminator is a problem since when $p_{\text{real}} \approx 0$, $D(x) \approx \beta$ and p_{fake} is not meant to go closer to p_{real}, which does not make sense since this is what we are trying to achieve. Therefore, β has to be kept at 0 for label smoothing to work. Hence, the authors recommend the use of one-sided label smoothing.

7.2 Results

The influence of adding label smoothing is reported table 7. I was not able to improve the FID using this technique, probably because I kept the parameters found previously and it requires some more tuning, for instance on the learning rates, as label smoothing changes the amplitude of the loss. The paper mentions the use of $\alpha = 0.9$ instead of 1, and I also experimented with $\alpha = 0.5$ which gave slightly better results.

Table 7: Influence of using label smoothing.

Positive label value	**FID**
1	**55.5**
0.9	63
0.5	60.4

8 Conclusion

After introducing the DCGAN framework, I presented the results of my DCGAN implementation applied to the task of generating dogs. After tweaking some parameters, I managed to achieve a FID of 56.5. I studied the influence of some techniques designed to improve the performance of the model, and added conditioning.

On the one hand, adding a cosine scheduling gave a 1 point boost and improved stability, and seemed like something worth including in the training of my DCGAN. On the other hand, techniques such as label smoothing and feature matching did not improve the models. These results might be to take with a pinch of salt for they are assessed with the FID, a metric that is not a perfect indicator of the performance. Furthermore, GAN instability can result in big variations and changing the seed could in fact lead to a different conclusion.
Regarding the addition of dog breed information, I had no success with both conditioning and auxiliary classifier. It appears that both of these ideas require more parameter tuning than expected and I could not make it work on my setup.

As mentioned earlier, state of the art model now achieve much better performances. The field of image generation evolved very rapidly recently and a lot of progress has been made. However, a lot of ideas introduced in the DCGAN architecture were kept as this architecture outperformed previously existing ones by a substantial margin. The fact of using a fully convolutional architecture was mostly kept, and the normalisation idea is still fundamental although architectures tend to prefer spectral normalisation to batch normalisation now.

References

1. Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018.
2. Harm de Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C. Courville. Modulating early visual processing by language. *CoRR*, abs/1707.00683, 2017.
3. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
4. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.
5. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
6. Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.

7. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
8. Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.
9. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
10. Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
11. Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.
12. Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.
13. Takeru Miyato and Masanori Koyama. cgans with projection discriminator. *CoRR*, abs/1802.05637, 2018.
14. Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.
15. Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.
16. A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *Intl Conf. on Learning Representation*, 2016.
17. Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *CoRR*, abs/1605.05396, 2016.
18. Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.
19. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.
20. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

Appendix A

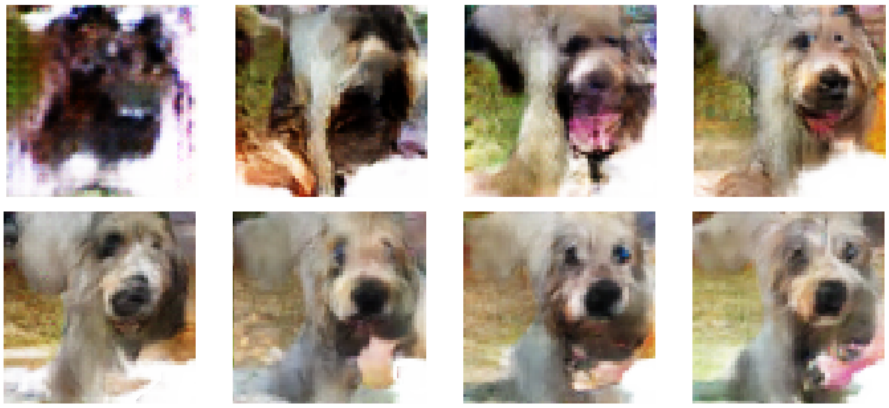


Fig. 11: Evolution of the output for a fixed noise during the training. The used model achieves a FID of 56.5 and does not use scheduling.

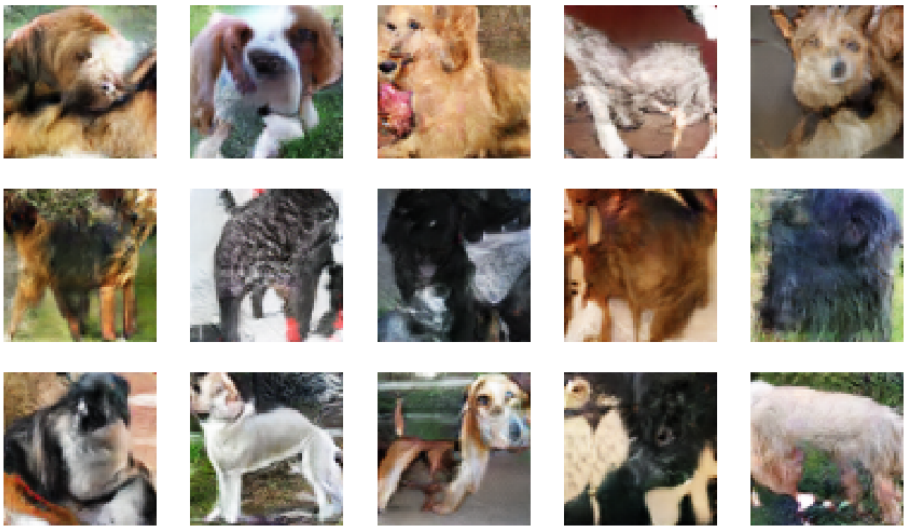
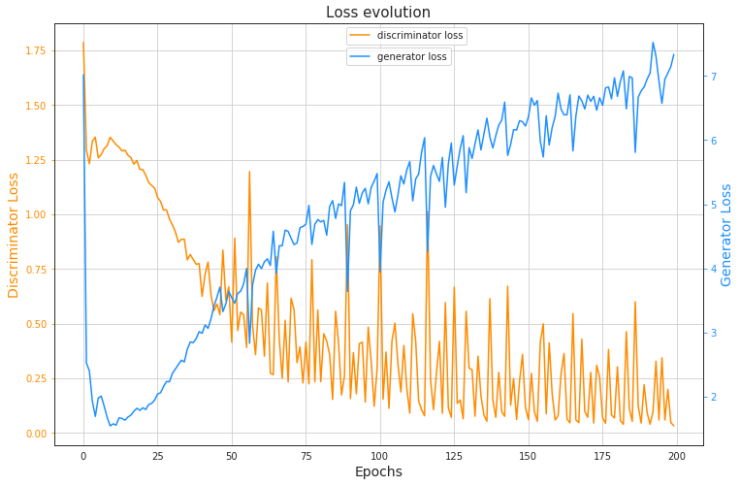
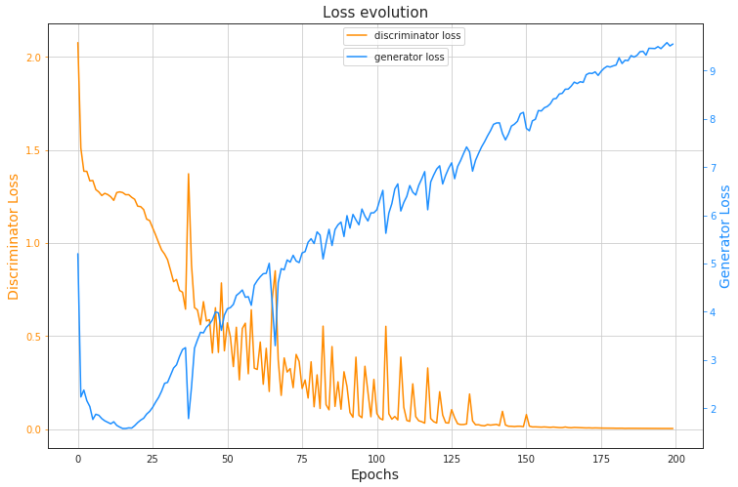


Fig. 12: Generated dogs for the best model, achieving an FID of 55.5.

Appendix B



(a) Without Scheduling



(b) With Scheduling

Fig. 13: Influence of adding a cosine learning rate scheduling on the generator and discriminator losses. The training is much more stable with the scheduler. However, the scheduler helps the discriminator loss reach 0 which highlights that the task is indeed too hard for the DCGAN generator.