EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Image Analysis II

Generative Adversarial Networks (GANs)

https://www.whichfaceisreal.com/

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Doprava zdarma

Udržitelný materiál

A.I.Chair

PHILIPPE STARCK

Jídelní židle A.I. Chair od designéra Philippe Starck je prvním počinem Kartellu a umělé inteligence. Více informací

Barva : Vyberte

● 2 varianty skladem

Může být u vás doma do **3 dnů**.

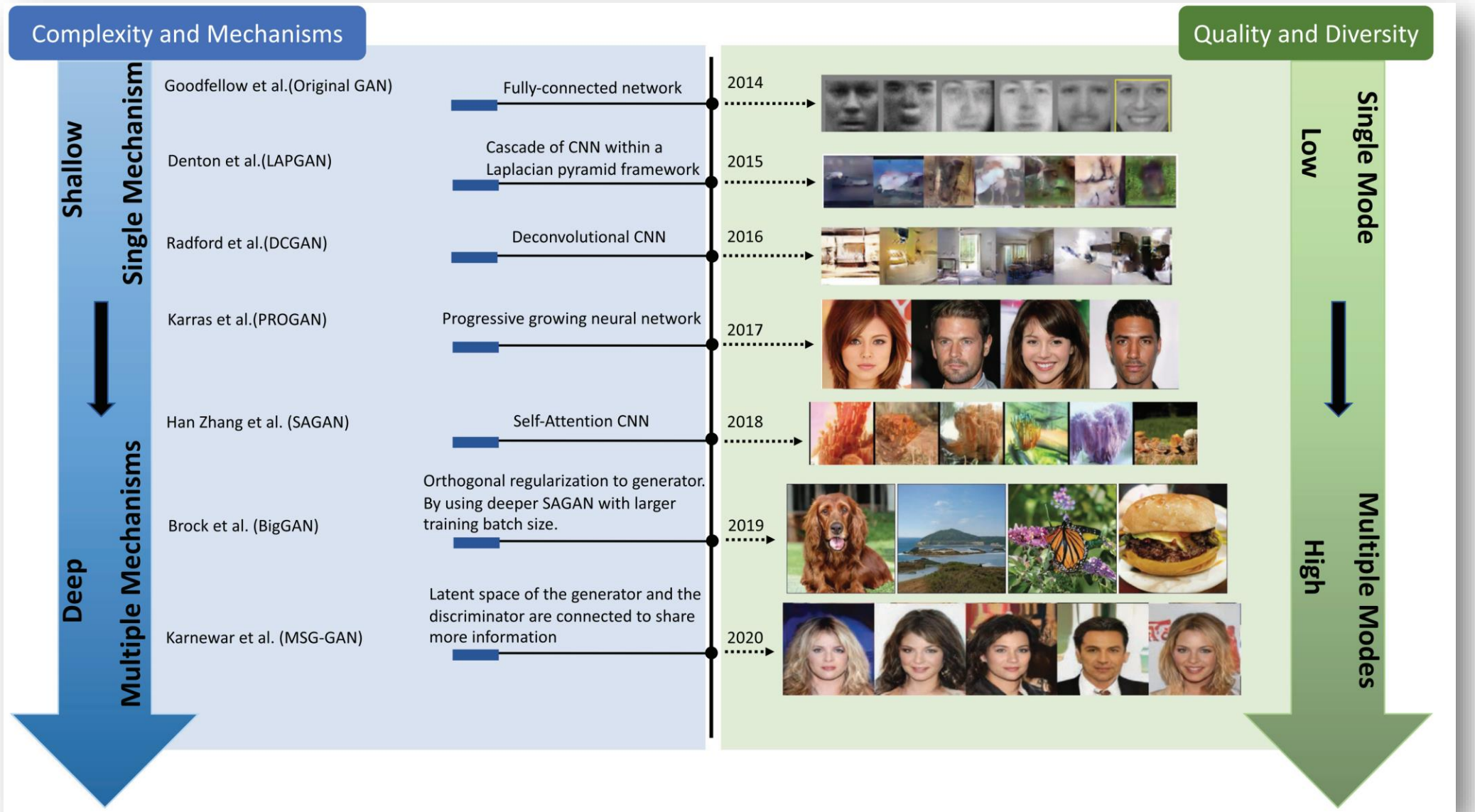
6 230 Kč

5 149 Kč bez DPH

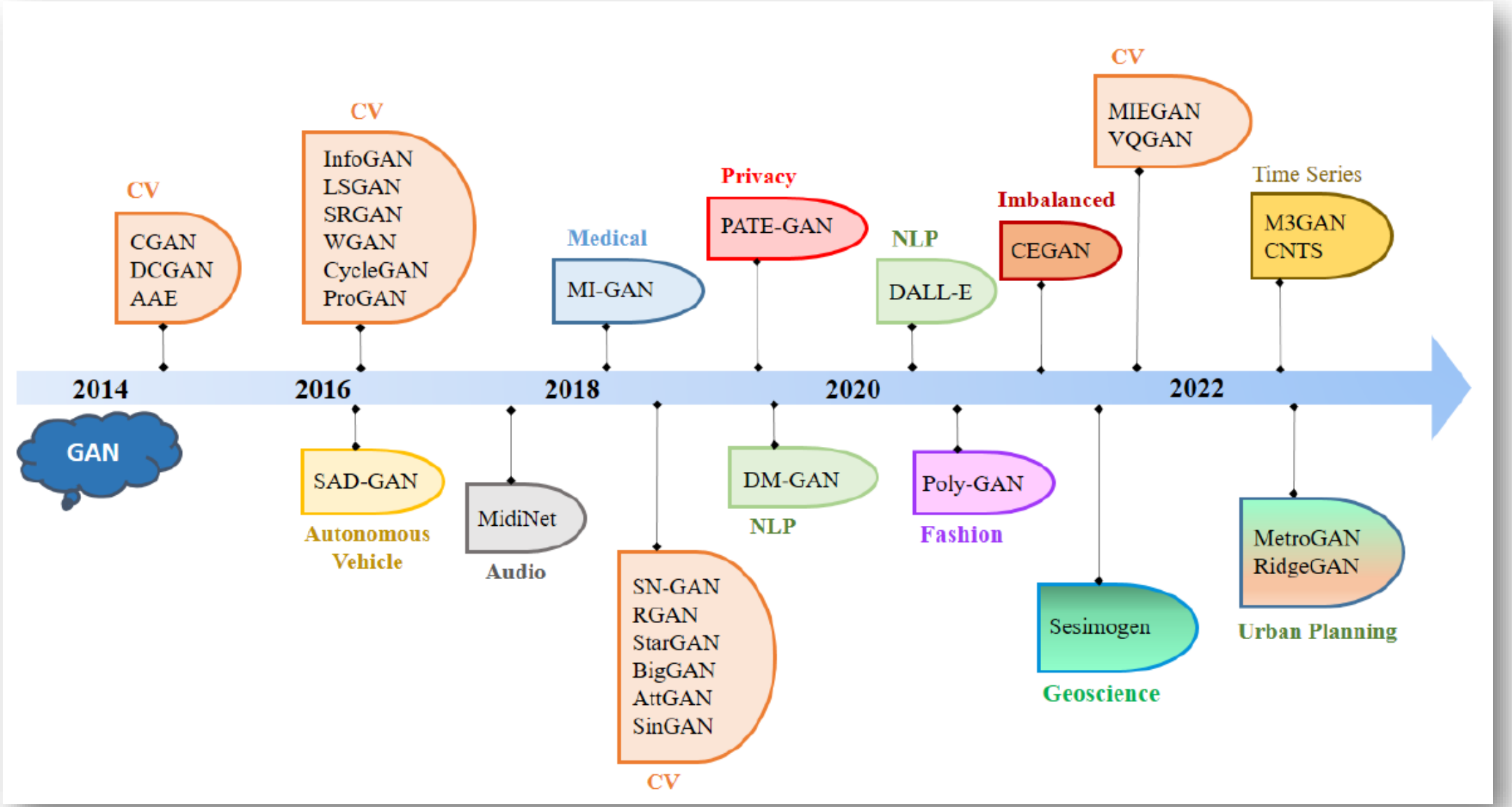
1 ks

DO KOŠÍKU

Naši designéři vám poradí s výběrem: **Kontaktujte nás!**



https://www.researchgate.net/publication/349189619_Generative_Adversarial_Networks_in_Computer_Vision_A_Survey_and_Taxonomy



https://www.researchgate.net/publication/373551906_Ten_Years_of_Generative_Adversarial_Nets_GANs_A_survey_of_the_state-of-the-art

https://www.researchgate.net/publication/349189619_Generative_Adversarial_Networks_in_Computer_Vision_A_Survey_and_Taxonomy

https://github.com/dongb5/GAN-Timeline

https://github.com/hindupuravinash/the-gan-zoo

LoGAN: Generating Logos with a Generative Adversarial Neural Network Conditioned on color

Ajkel Mino
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, Netherlands
ajkimino@gmail.com

Gerasimos Spanakis
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, Netherlands
jerry.spanakis@maastrichtuniversity.nl

Fig. 6. Results from the generation of 64 logos per class after 400 epochs of training. Classes from left to right top to bottom: green, purple, white, brown, blue, cyan, yellow, gray, red, pink, orange, black.

https://www.researchgate.net/publication/373551906_Ten_Years_of_Generative_Adversarial_Nets_GANs_A_survey_of_the_state-of-the-art
https://arxiv.org/pdf/1810.10395.pdf
https://www.researchgate.net/publication/349189619_Generative_Adversarial_Networks_in_Computer_Vision_A_Survey_and_Taxonomy

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

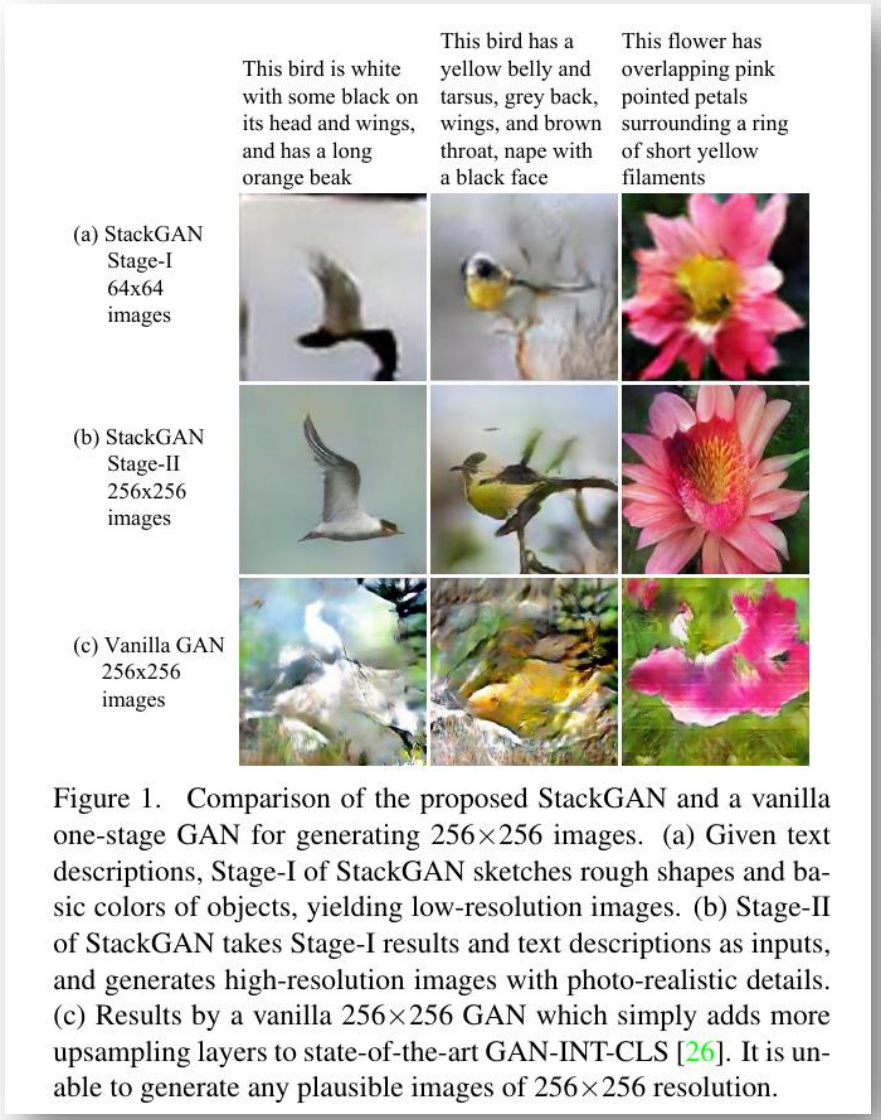


Figure 1. Comparison of the proposed StackGAN and a vanilla one-stage GAN for generating 256×256 images. (a) Given text descriptions, Stage-I of StackGAN sketches rough shapes and basic colors of objects, yielding low-resolution images. (b) Stage-II of StackGAN takes Stage-I results and text descriptions as inputs, and generates high-resolution images with photo-realistic details. (c) Results by a vanilla 256×256 GAN which simply adds more upsampling layers to state-of-the-art GAN-INT-CLS [26]. It is unable to generate any plausible images of 256×256 resolution.

StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks

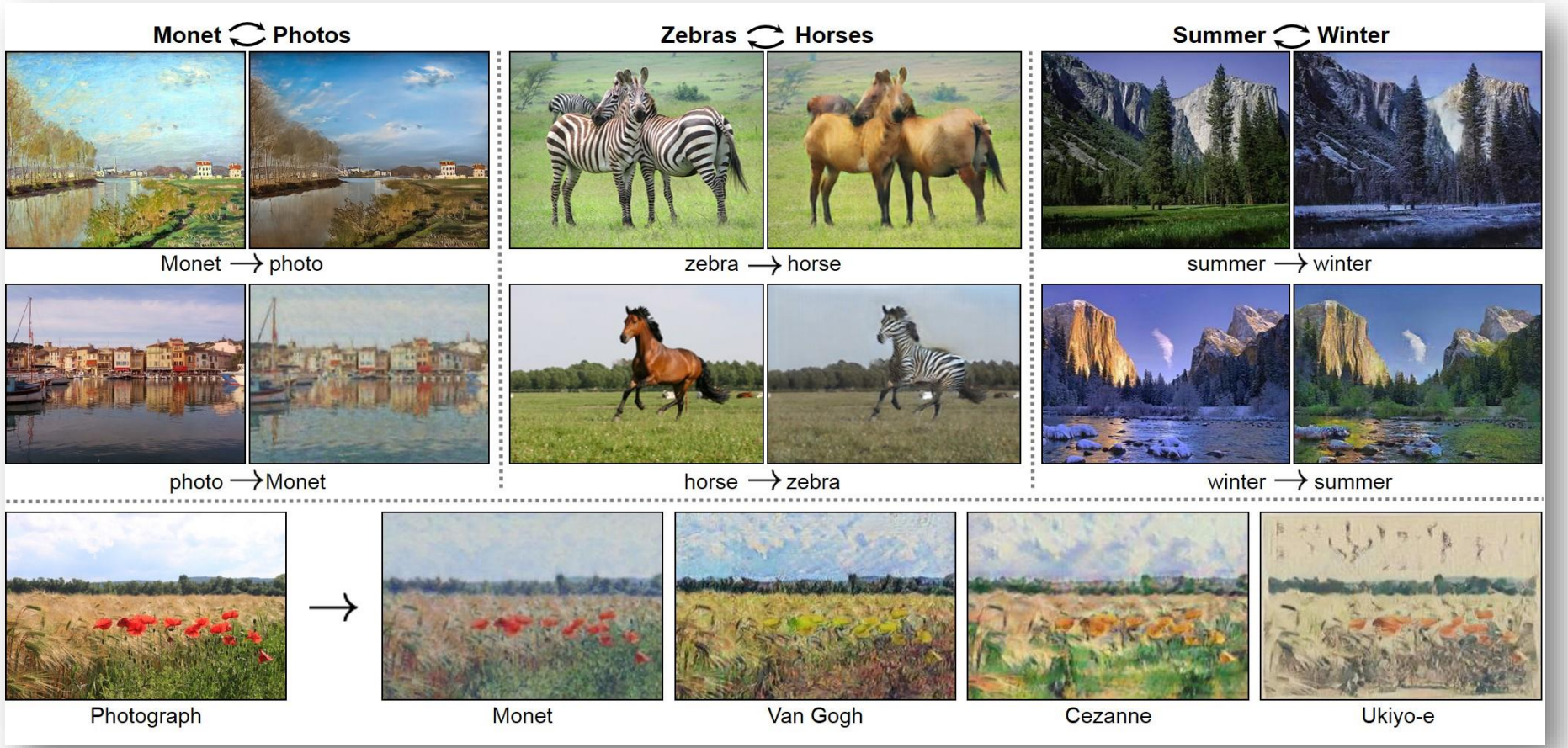
Han Zhang[1], Tao Xu[2], Hongsheng Li[3],
Shaoting Zhang[4], Xiaogang Wang[3], Xiaolei Huang[2], Dimitris Metaxas[1]

[1]Rutgers University [2]Lehigh University [3]The Chinese University of Hong Kong [4]Baidu Research
{han.zhang, dnm}@cs.rutgers.edu, {tax313, xih206}@lehigh.edu
{hsli, xgwang}@ee.cuhk.edu.hk, zhangshaoting@baidu.com

https://github.com/hanzhanggit/StackGAN
https://www.researchgate.net/publication/373551906_Ten_Years_of_Generative_Adversarial_Nets_GANs_A_survey_of_the_state-of-the-art
https://www.researchgate.net/publication/349189619_Generative_Adversarial_Networks_in_Computer_Vision_A_Survey_and_Taxonomy

https://github.com/Sxela/ArcaneGAN/releases/tag/v0.2
https://www.researchgate.net/publication/373551906_Ten_Years_of_Generative_Adversarial_Nets_GANs_A_survey_of_the_state-of-the-art
https://www.researchgate.net/publication/349189619_Generative_Adversarial_Networks_in_Computer_Vision_A_Survey_and_Taxonomy

https://www.youtube.com/watch?v=N7KbfWodXJE
https://github.com/junyanz/CycleGAN



https://github.com/Sxela/ArcaneGAN/releases/tag/v0.2
https://www.researchgate.net/publication/373551906_Ten_Years_of_Generative_Adversarial_Nets_GANs_A_survey_of_the_state-of-the-art
https://www.researchgate.net/publication/349189619_Generative_Adversarial_Networks_in_Computer_Vision_A_Survey_and_Taxonomy

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi
Twitter

{cledig,ltheis,fhuszar,jcaballero,aacostadiaz,aaitken,atejani,jtotz,zehanw,wshi}@twitter.com

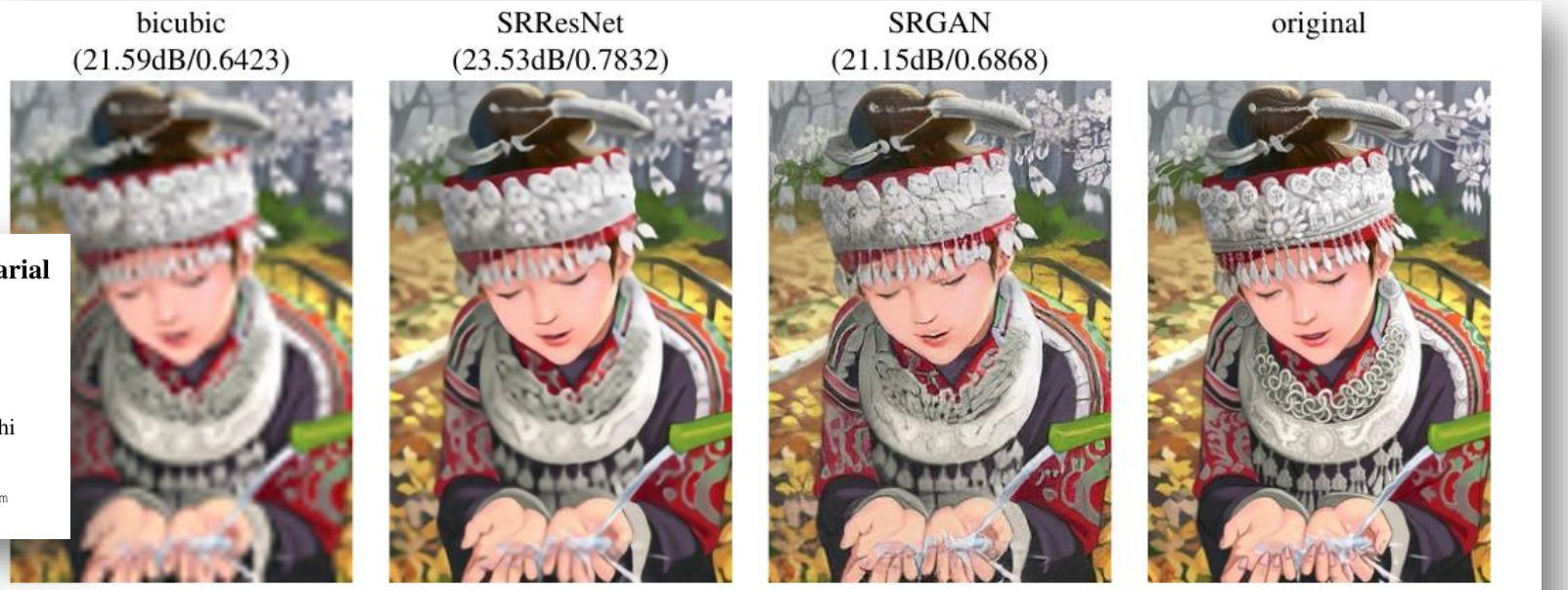


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative mized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and kets. [4× upscaling]

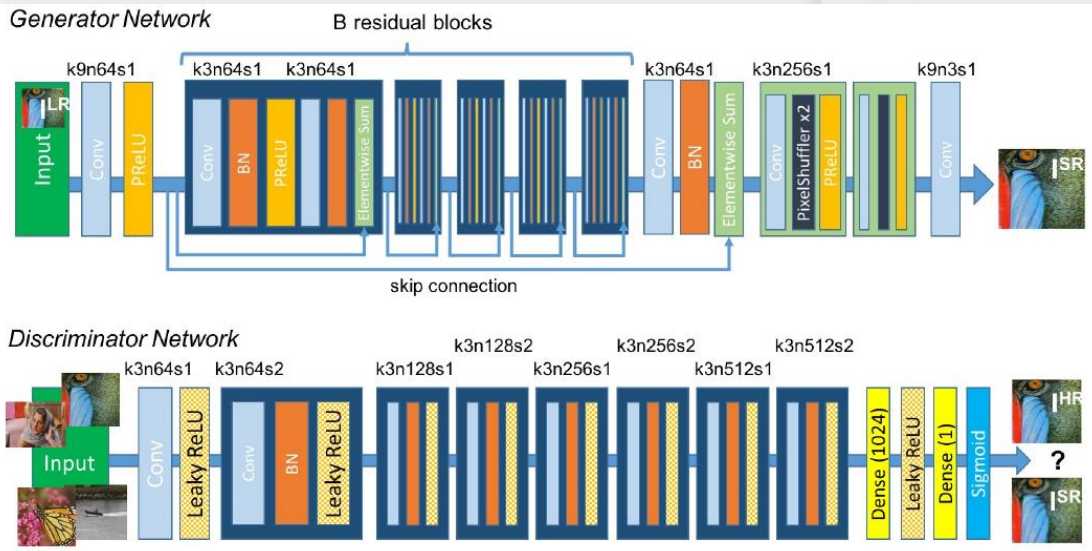


Figure 4: Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

https://arxiv.org/pdf/1609.04802.pdf

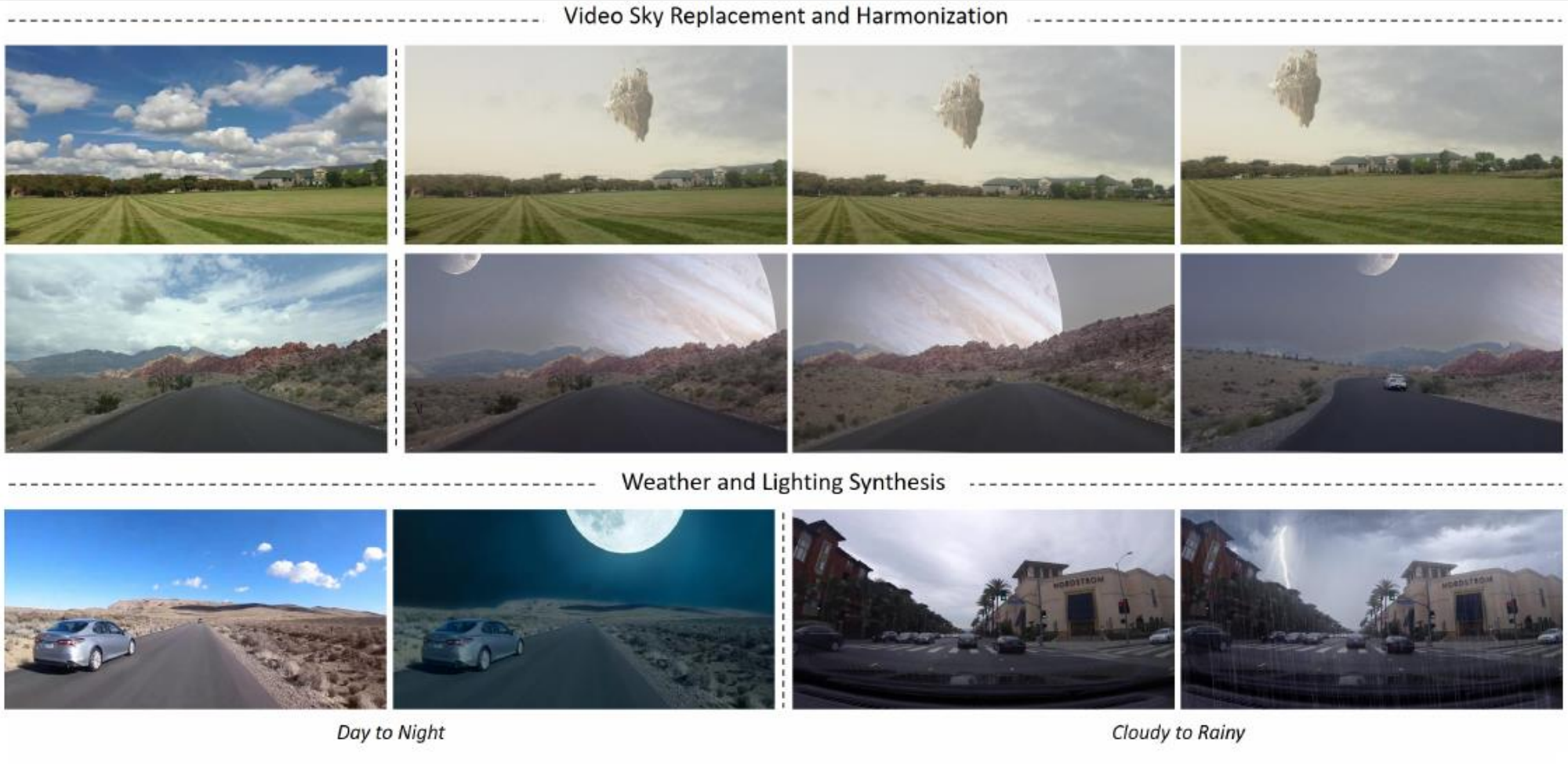


Figure 1: We propose a vision-based method for generating videos with controllable sky backgrounds and realistic weather/lighting conditions. First and second row: rendered sky videos - flying castle and Jupiter sky (leftmost shows a frame from input video and the rest are the output frames). Third row: weather and lighting synthesis (day to night, cloudy to rainy).

https://arxiv.org/pdf/2010.11800v1.pdf
https://www.youtube.com/watch?v=zal9Ues0aOQ

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Generative Adversarial Nets

Ian J. Goodfellow[*], **Jean Pouget-Abadie**[†], **Mehdi Mirza**, **Bing Xu**, **David Warde-Farley**, **Sherjil Ozair**[‡], **Aaron Courville**, **Yoshua Bengio**[§]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

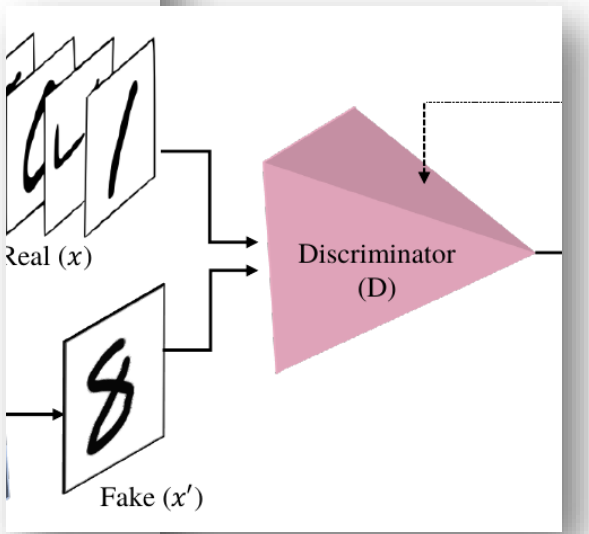
5 Experiments

We trained adversarial nets an a range of datasets including MNIST[21], the Toronto Face Database (TFD) [27], and CIFAR-10 [19]. The generator nets used a mixture of rectifier linear activations [17, 8] and sigmoid activations, while the discriminator net used maxout [9] activations. Dropout [16] was applied in training the discriminator net. While our theoretical framework permits the use of dropout and other noise at intermediate layers of the generator, we used noise as the input to only the bottommost layer of the generator network.

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

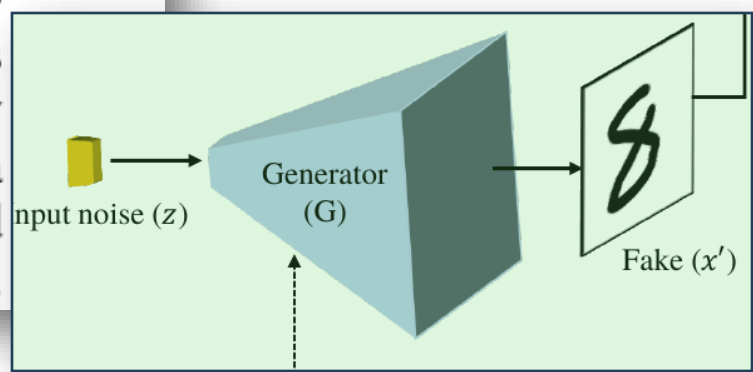
Ian J. Goodfellow, Jean Pouget-Abadie,* Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,† Aaron Courville, Yoshua Bengio‡

Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7



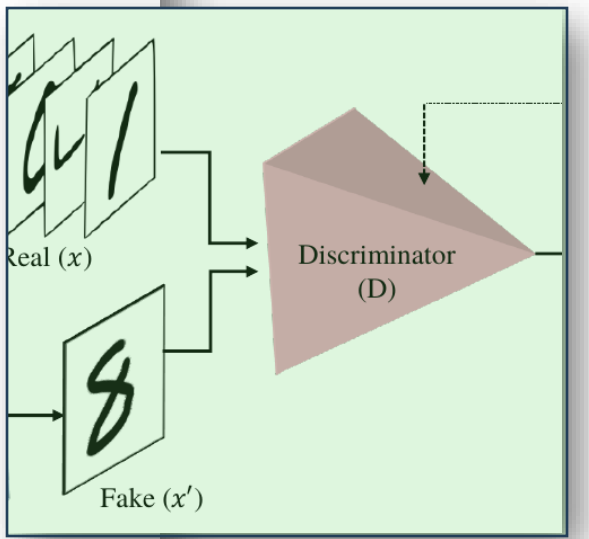
Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation.



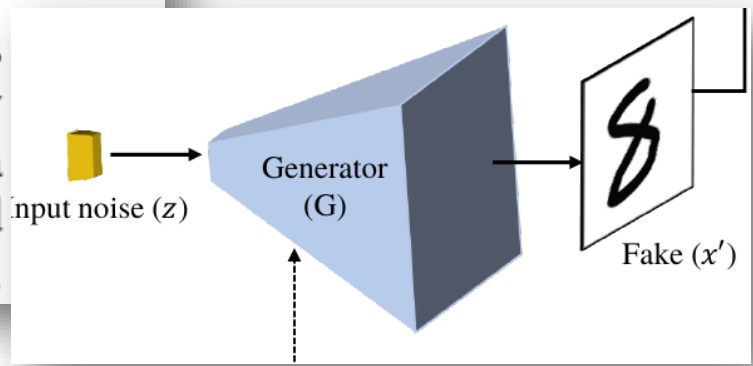
https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

Ian J. Goodfellow, Jean Pouget-Abadie,* Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,† Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7



Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation.



https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

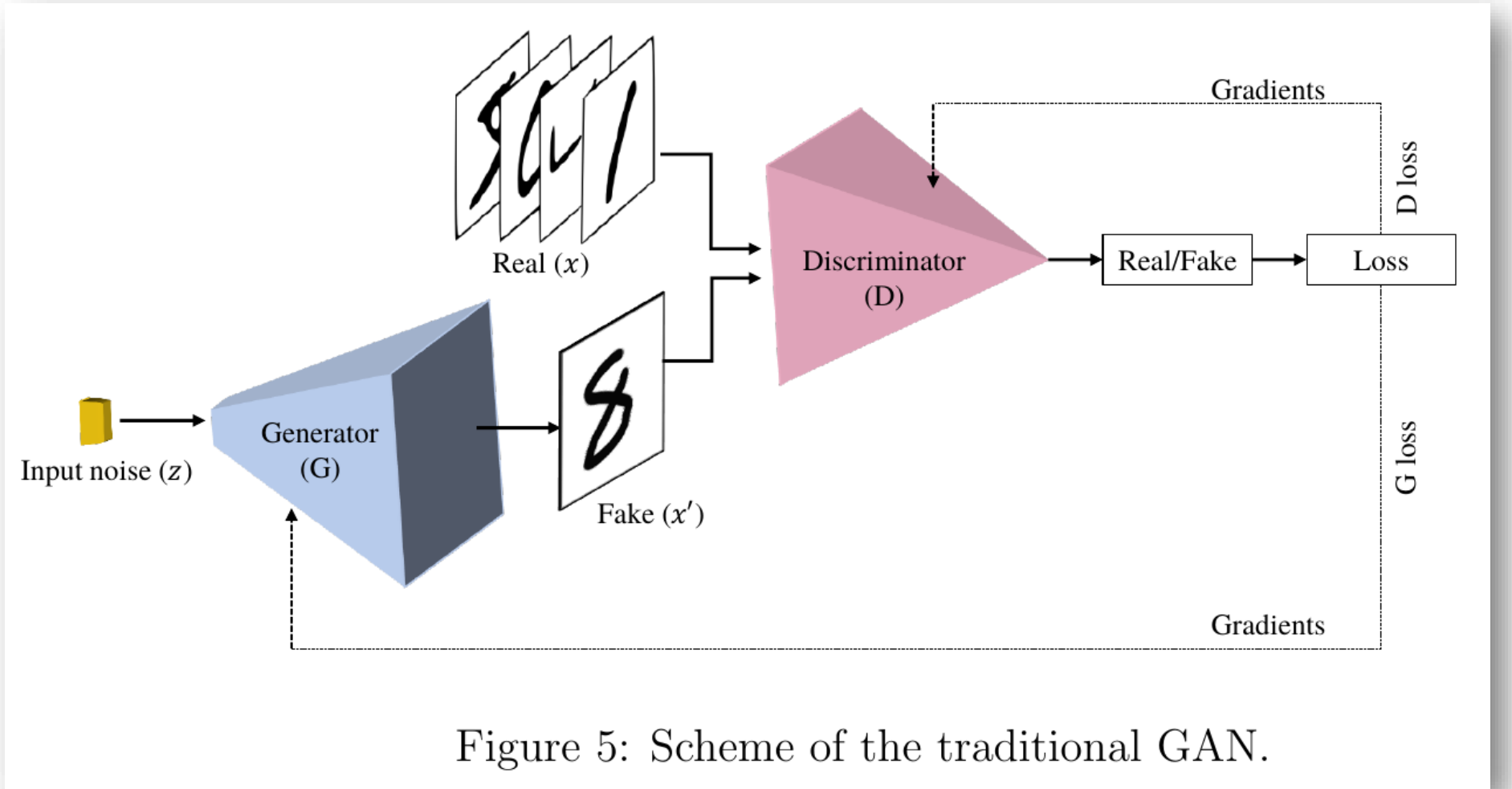


Figure 5: Scheme of the traditional GAN.

The discriminator is trying to be better and better and at the same time the generator is trying to be better and better to outsmart the discriminator.

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

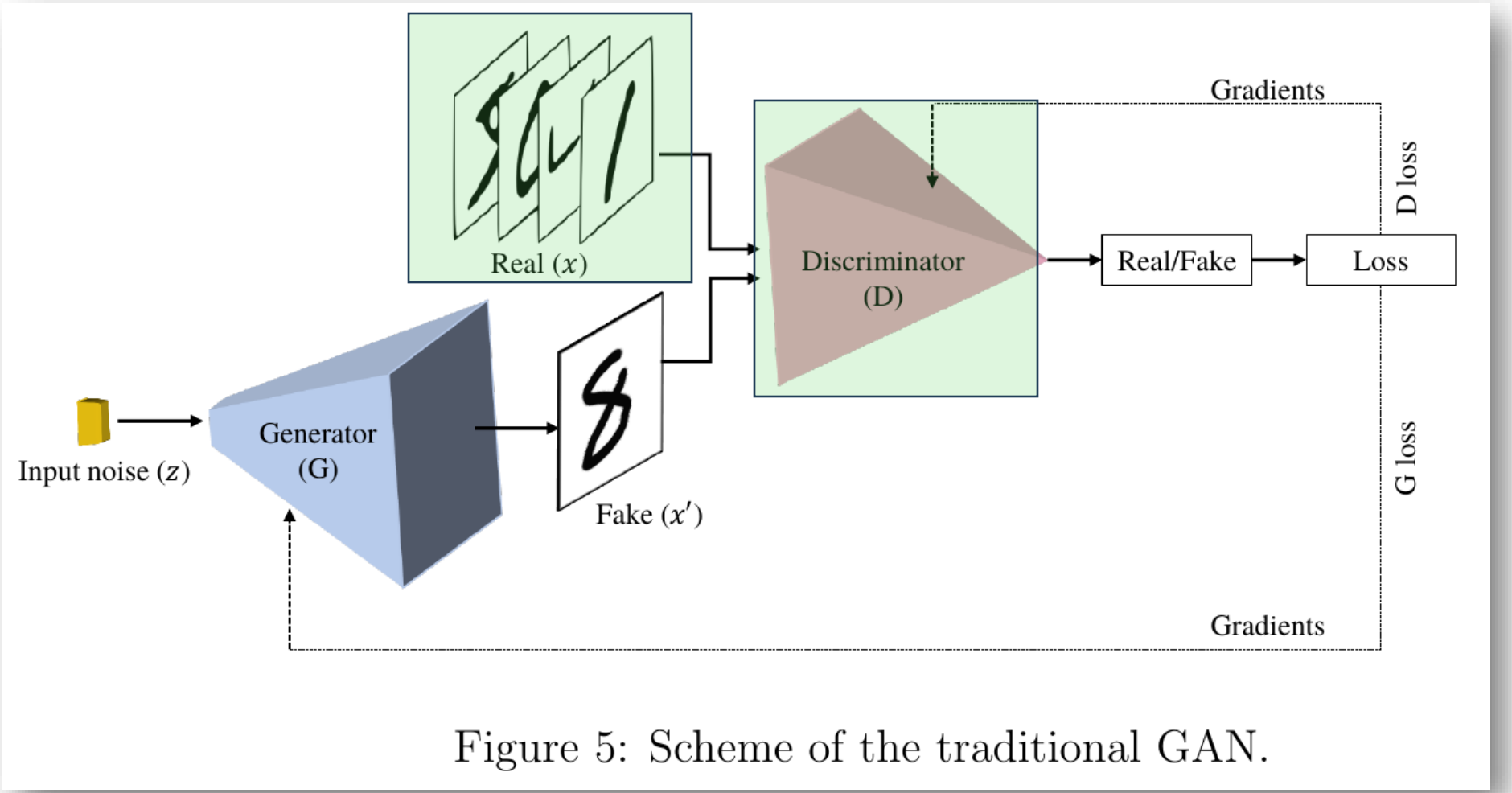


Figure 5: Scheme of the traditional GAN.

Discriminator Training Phase:

- supervised training
- real data with labels 1

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

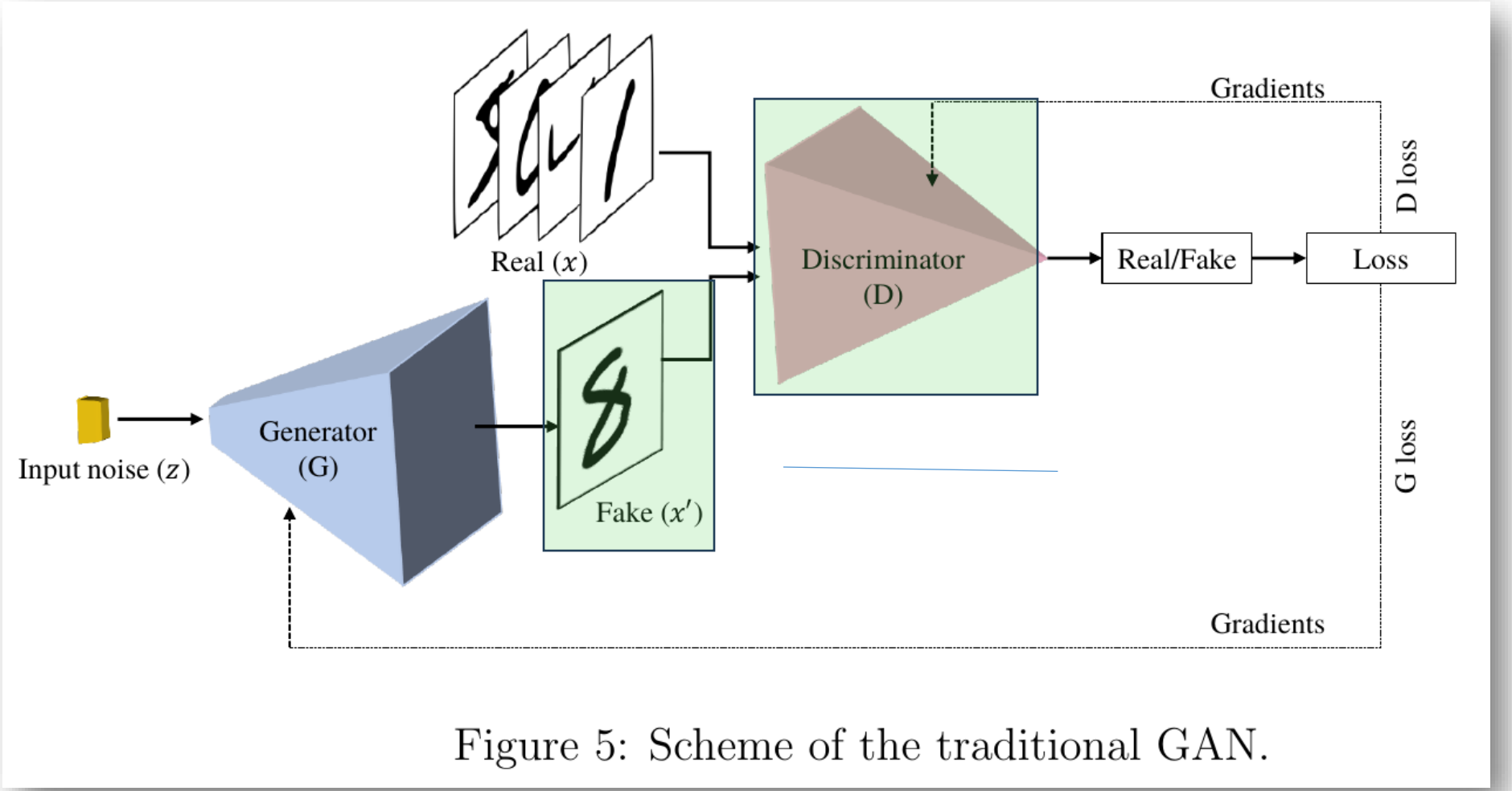


Figure 5: Scheme of the traditional GAN.

Discriminator Training Phase:

- supervised training
- real data with labels 1

- supervised training
- fake data with labels 0

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

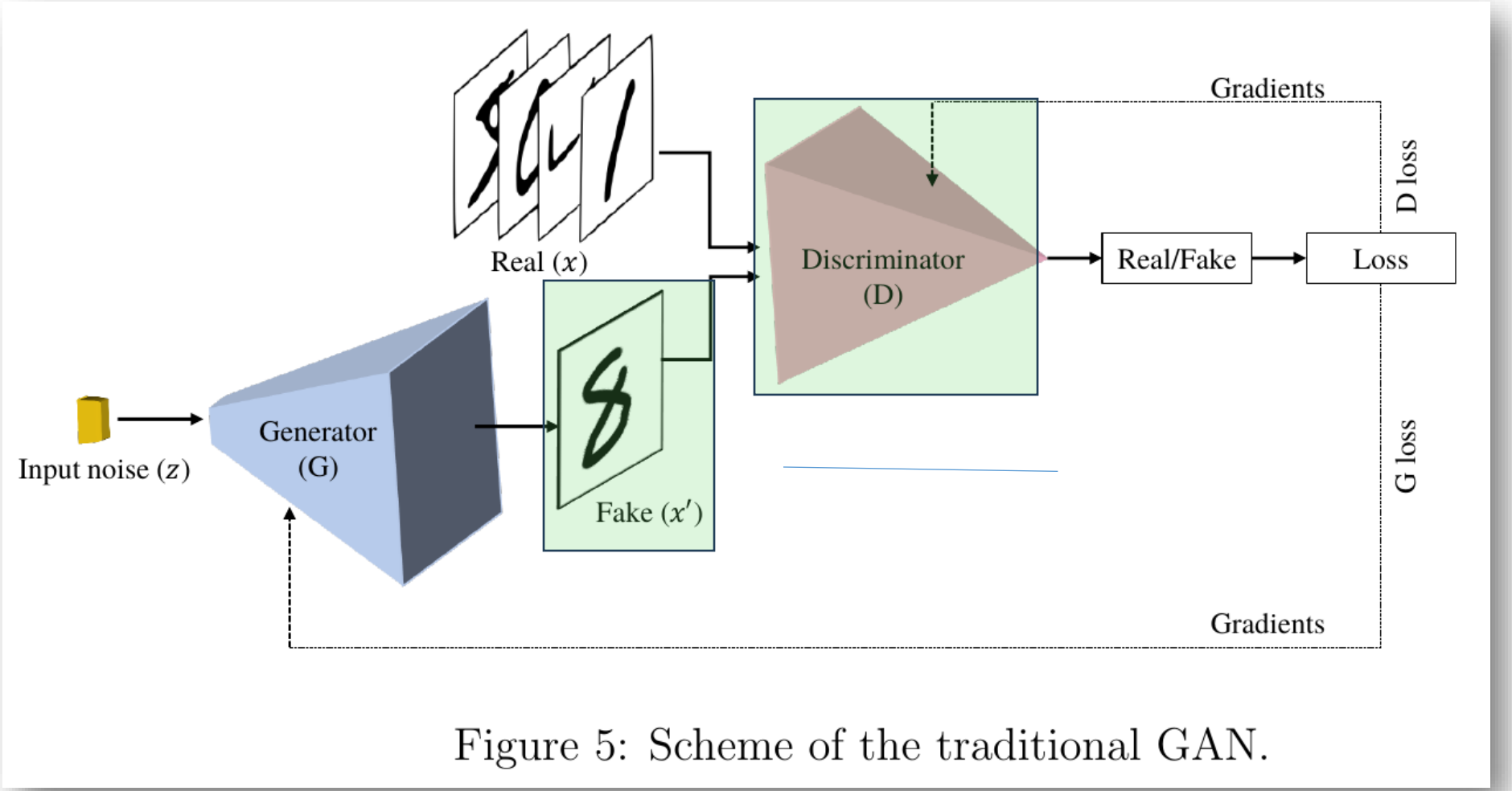


Figure 5: Scheme of the traditional GAN.

Discriminator Training Phase:

- supervised training
- real data with labels 1

- supervised training
- fake data with labels 0

The goal is (for the discriminator) to learn to distinguish between real and fake images.

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

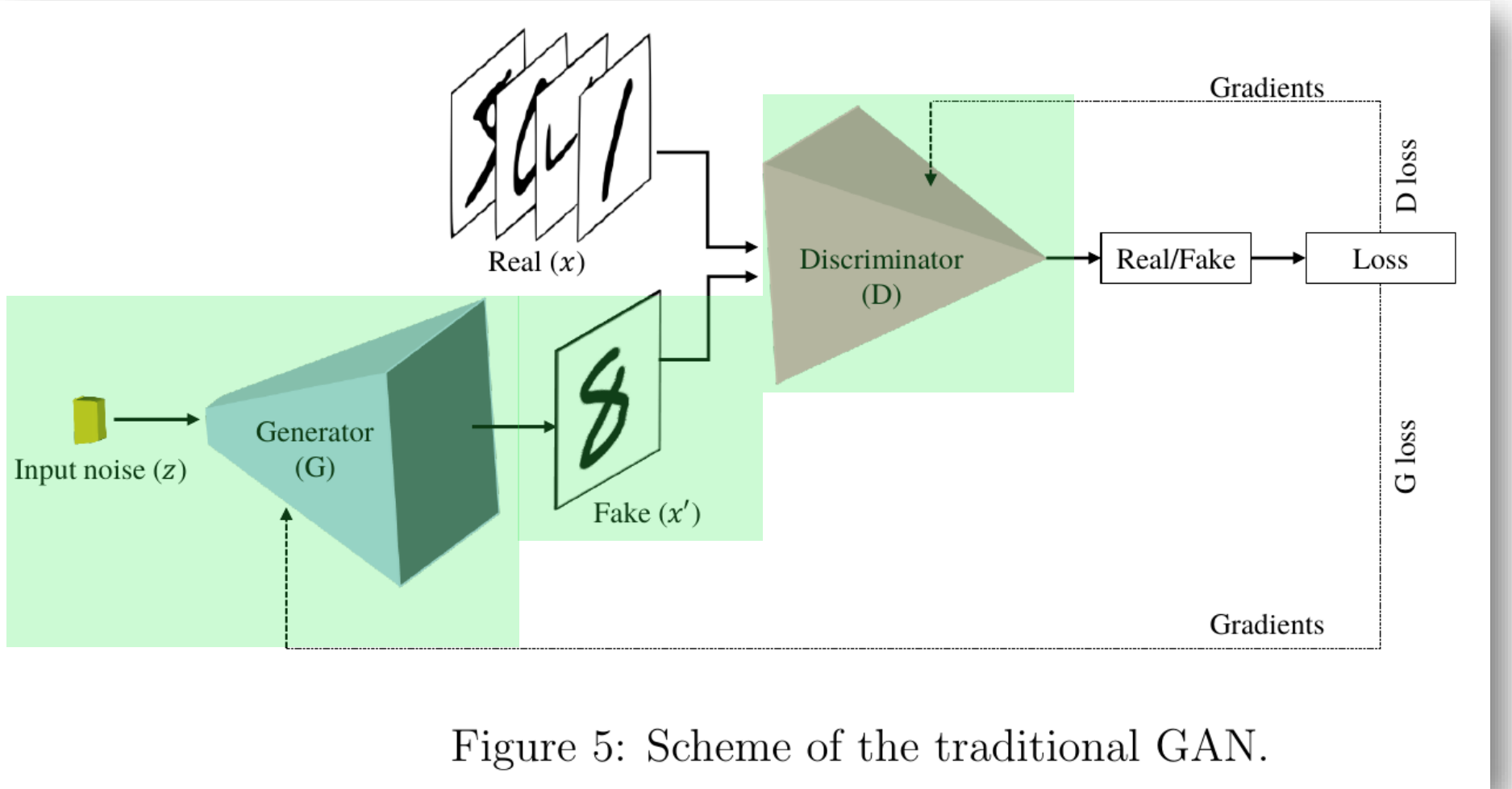


Figure 5: Scheme of the traditional GAN.

Generator Training Phase:

- We feed the generated data to the input of the discriminator.
- Tell the discriminator that they are real even though they are generated.
- What does the discriminator tell us, are they true or false?
- The discriminator tells us they are false -> that is an error signal that is propagated back to the generator.

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

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

From the discriminator's point of view:
The best possible result of D(G(z)) is close to 0.
This means that the discriminator won't be fooled.

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

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

From the discriminator's point of view:
The best possible result of D(G(z)) is close to 0.
This means that the discriminator won't be fooled.
What about D(x)?

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

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

From the discriminator's point of view:
The best possible result of D(G(z)) is close to 0.
This means that the discriminator won't be fooled.
What about D(x)?

The best possible result of D(x) is close to 1, from the discriminator's point of view.

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

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

From the discriminator's point of view:
The best possible result of D(G(z)) is close to 0.
This means that the discriminator won't be fooled.
What about D(x)?
The best possible result of D(x) is close to 1, From the discriminator's point of view.

From the generator's point of view:
The best possible result of D(G(z)) is 1.

This means that the generator outsmarted the discriminator.

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

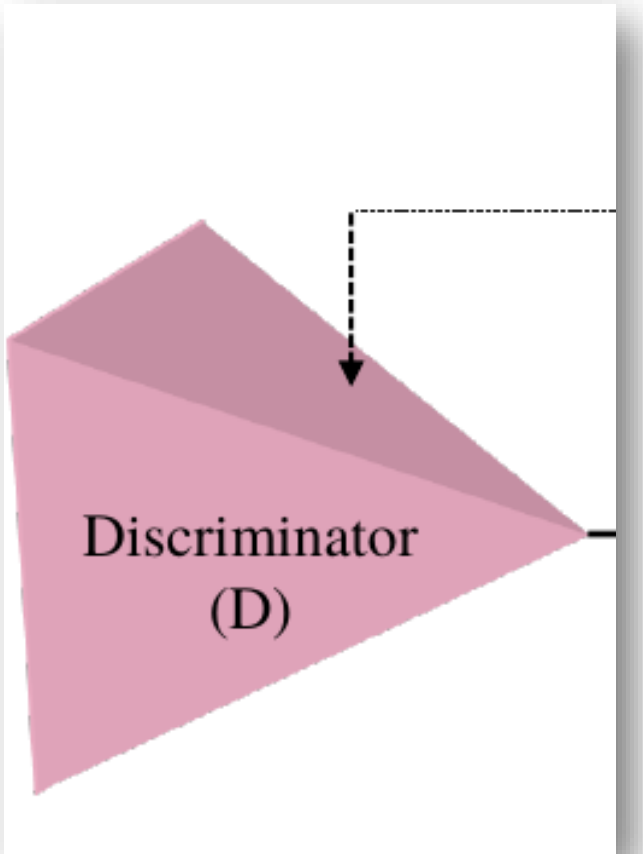
$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

From the discriminator's point of view:
The best possible result of D(G(z)) is close to 0.
This means that the discriminator won't be fooled.
What about D(x)?
The best possible result of D(x) is close to 1, From the discriminator's point of view.

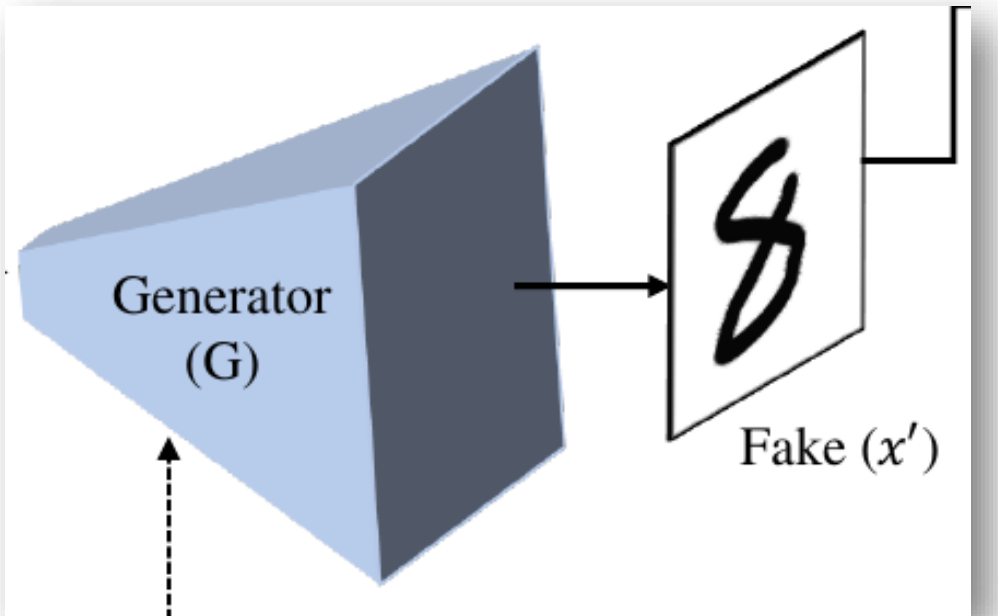
From the generator's point of view:
The best possible result of D(G(z)) is 1.
This means that the generator outsmarted the discriminator.

https://www.researchgate.net/publication/341425292_Throwing_light_on_black_boxes_emergence_of_visual_categories_from_deep_learning
https://arxiv.org/abs/1406.2661

```python
class Discriminator_0(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(IMG_SIZE*IMG_SIZE, 2048),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3),
            nn.Linear(2048, 1024),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        output = self.model(x)
        return output
```



```python
class Generator_0(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, IMG_SIZE*IMG_SIZE),
            nn.Tanh(),
        )

    def forward(self, x):
        output = self.model(x)
        return output
```



EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Training phase

```
real_img, labels = data[0].to(device), data[1].to(device)
fake_img = g_net(torch.randn(BATCH_SIZE, 100).to(device))

real_labels = torch.ones(BATCH_SIZE,1).to(device)
fake_labels = torch.zeros(BATCH_SIZE,1).to(device)
```

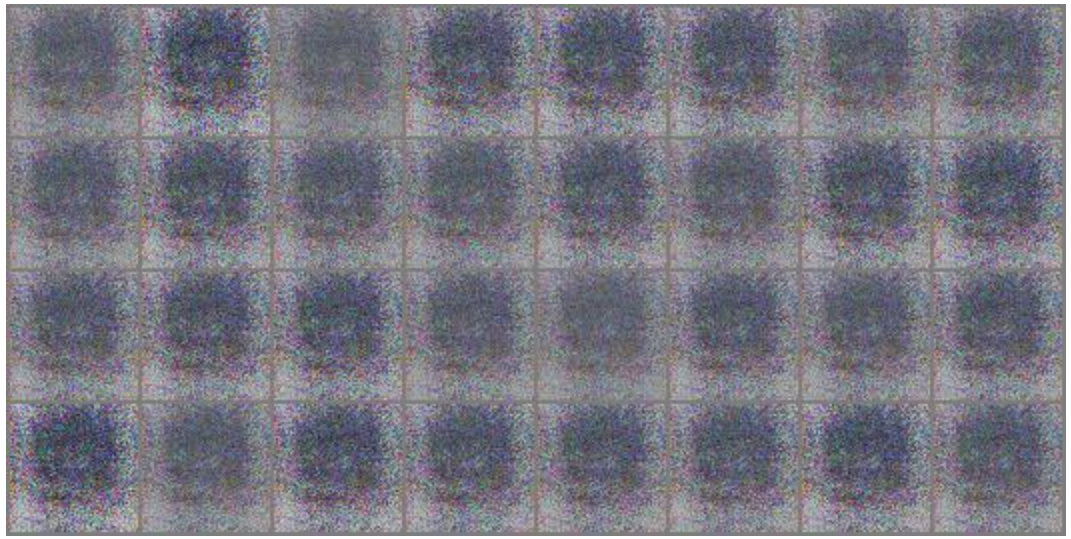
```
# REAL IMG - all labels are 1
pred_real   = d_net(real_img)
d_loss_real = lossfun(pred_real,real_labels)

# FAKE IMG - all labels are 0
pred_fake   = d_net(fake_img)
d_loss_fake = lossfun(pred_fake,fake_labels)

# combined losses
d_loss = d_loss_real + d_loss_fake

# backprop
d_optimizer.zero_grad()
d_loss.backward()
d_optimizer.step()
```

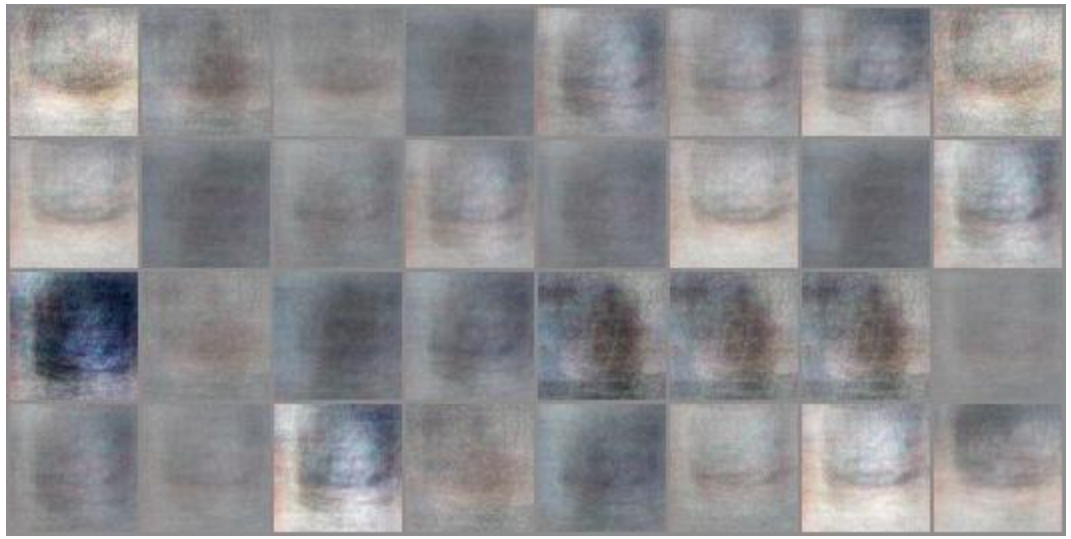
Training phase

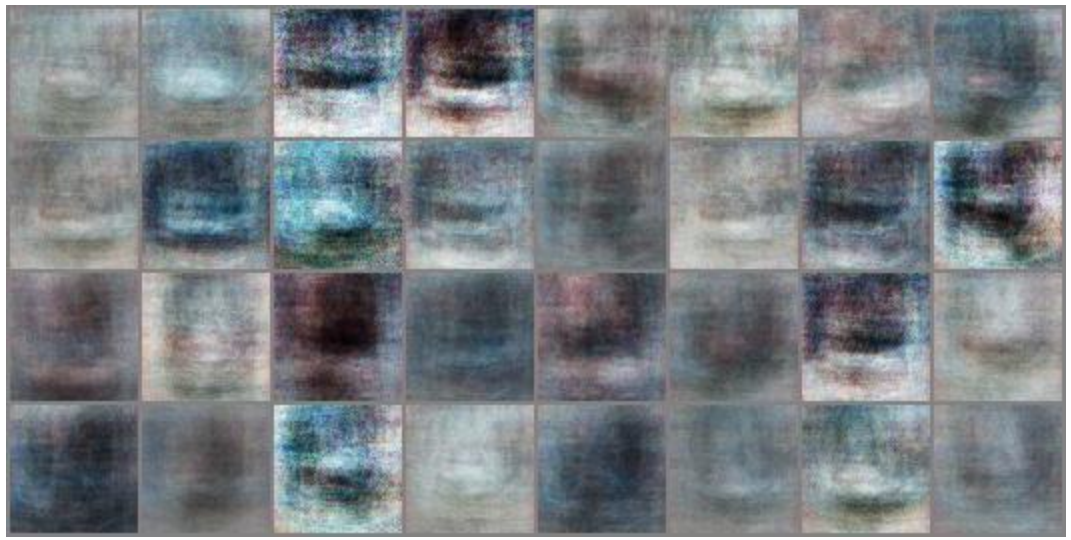
```
# create fake images
fake_images = g_net( torch.randn(BATCH_SIZE, 100).to(device) )
pred_fake   = d_net(fake_images)

# compute loss
g_loss = lossfun(pred_fake,real_labels)

# backprop
g_optimizer.zero_grad()
g_loss.backward()
g_optimizer.step()
```

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY







EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

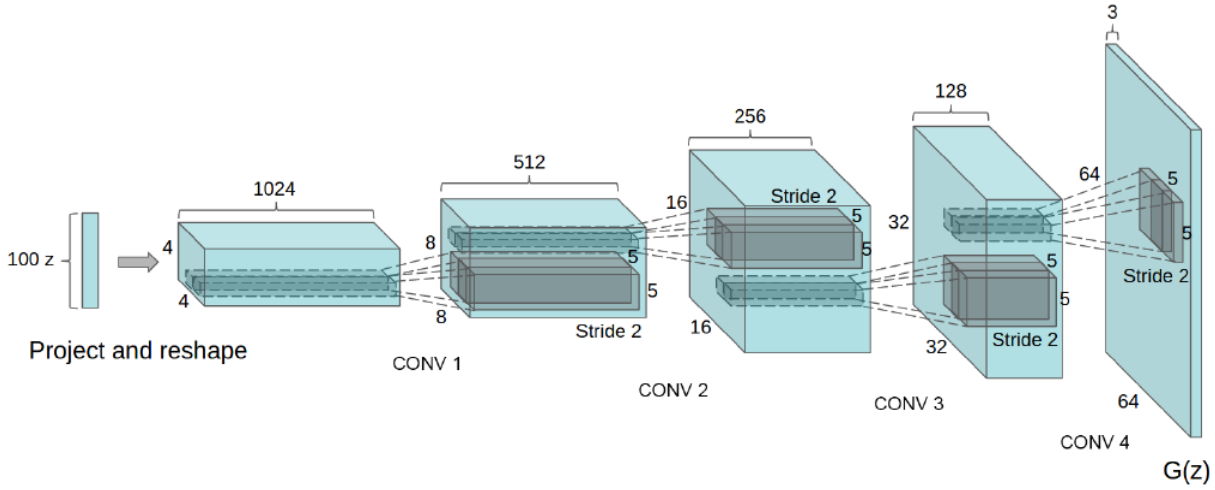


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

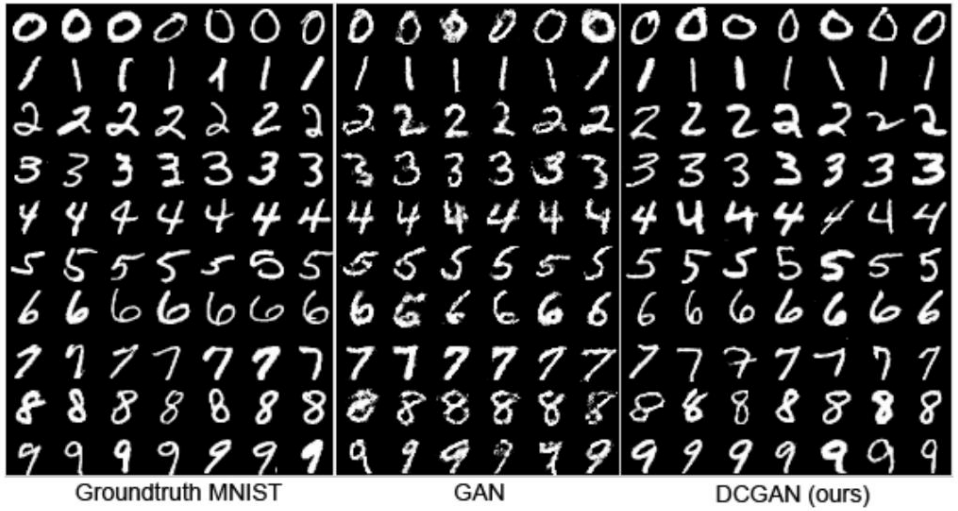
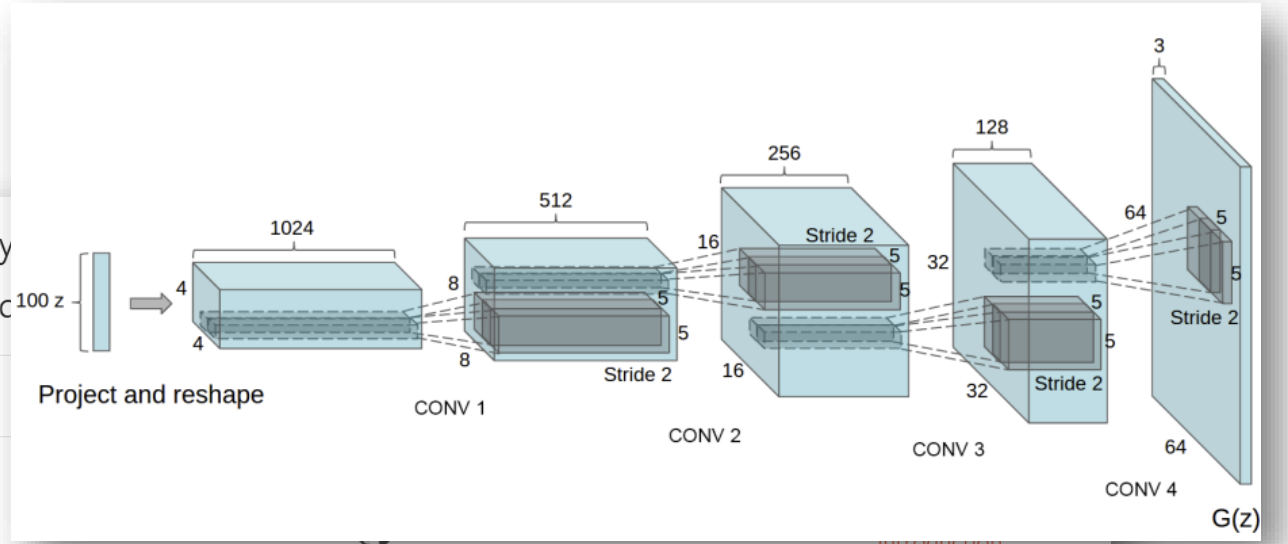


Figure 9: Side-by-side illustration of (from left-to-right) the MNIST dataset, generations from a baseline GAN, and generations from our DCGAN .

https://arxiv.org/abs/1511.06434



PyTorch

Get Started | Ecosystem | Py Ec

2.1.1+cu121

Search Tutorials

PyTorch Recipes [+]

Introduction to PyTorch [-]

- Learn the Basics
- Quickstart
- Tensors
- Datasets & DataLoaders
- Transforms
- Build the Neural Network
- Automatic Differentiation with `torch.autograd`
- Optimizing Model Parameters
- Save and Load the Model

Introduction to PyTorch on YouTube [-]

Tutorials > DCGAN Tutorial

Run in Google Colab

DCGAN TUTORIAL

Author: Nathan Inkawhich

Introduction

This tutorial will give an introduction to DCGANs through an example. We will train a generative adversarial network (GAN) to generate new celebrities after showing it pictures of many real celebrities. Most of the code here is from the DCGAN implementation in pytorch/examples, and this document will give a thorough explanation of the implementation and shed light on how and why this model works. But don't worry, no prior knowledge of GANs is required, but it may require a first-timer to spend some time reasoning about what is actually happening under the hood. Also, for the sake of time it will help to have a GPU, or two. Lets start from the beginning.

Generative Adversarial Networks

Introduction
+ Generative Adversarial Networks
Inputs
Data
+ Implementation
Results
Where to Go Next

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

What is a DCGAN?

A DCGAN is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. It was first described by Radford et. al. in the paper Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. 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. 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 the paper, the authors also give some tips about how to setup the optimizers, how to calculate the loss functions, and how to initialize the model weights, all of which will be explained in the coming sections.

Tutorials > DCGAN Tutorial

What is a DCGAN?

A DCGAN is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. It was first described by Radford et. al. in the paper Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. The discriminator is made up of strided convolution layers, batch norm layers, and

Ignoring channels for now, let's begin with the basic transposed convolution operation with stride of 1 and no padding. Suppose that we are given a $n_h \times n_w$ input tensor and a $k_h \times k_w$ kernel. Sliding the kernel window with stride of 1 for n_w times in each row and n_h times in each column yields a total of $n_h n_w$ intermediate results. Each intermediate result is a $(n_h + k_h - 1) \times (n_w + k_w - 1)$ tensor that are initialized as zeros. To compute each intermediate tensor, each element in the input tensor is multiplied by the kernel so that the resulting $k_h \times k_w$ tensor replaces a portion in each intermediate tensor. Note that the position of the replaced portion in each intermediate tensor corresponds to the position of the element in the input tensor used for the computation. In the end, all the intermediate results are summed over to produce the output.

As an example, Fig. 14.10.1 illustrates how transposed convolution with a 2×2 kernel is computed for a 2×2 input tensor.

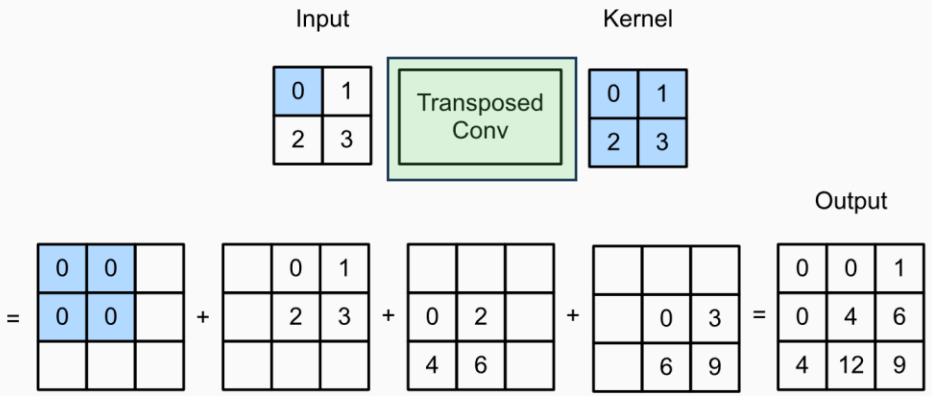


Fig. 14.10.1 Transposed convolution with a 2×2 kernel. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
https://d2l.ai/chapter_computer-vision/transposed-conv.html

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

What is a DCGAN?

A DCGAN is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. It was first described by Radford et. al. in the paper Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. The discriminator is made up of strided convolution layers, batch norm layers, and

In the transposed convolution, strides are specified for intermediate results (thus output), not for input. Using the same input and kernel tensors from Fig. 14.10.1, changing the stride from 1 to 2 increases both the height and weight of intermediate tensors, hence the output tensor in Fig. 14.10.2.

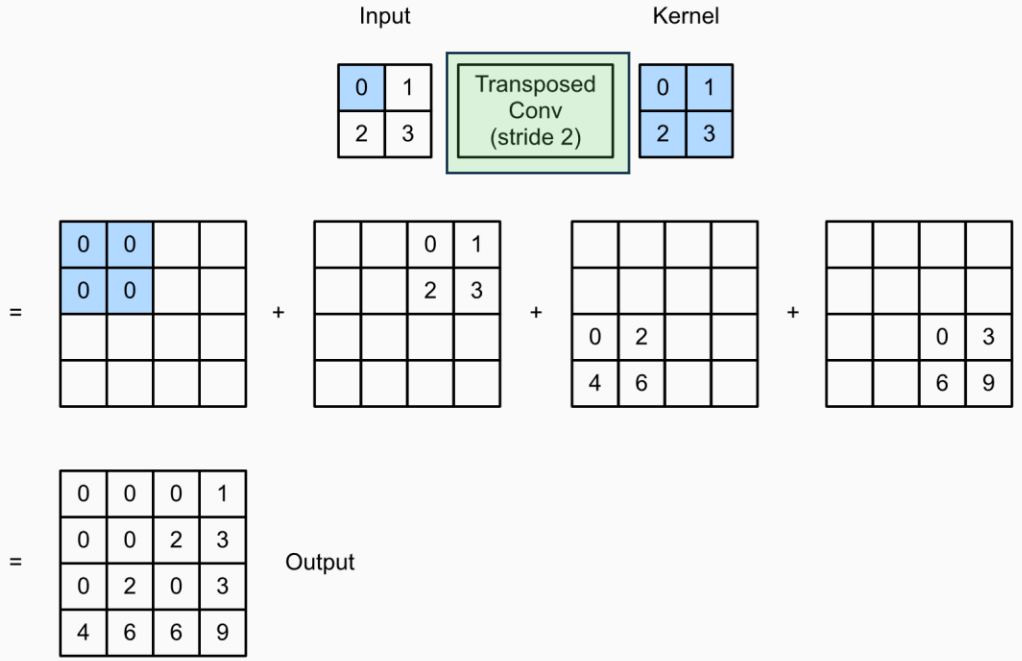


Fig. 14.10.2 Transposed convolution with a 2×2 kernel with stride of 2. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
https://d2l.ai/chapter_computer-vision/transposed-conv.html

Docs > torch.nn > ConvTranspose2d

ConvTranspose2d

CLASS torch.nn.ConvTranspose2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D transposed convolution operator over an input image composed of several input planes.

This module can be seen as the gradient of Conv2d with respect to its input. It is also known as a fractionally-strided convolution or a deconvolution (although it is not an actual deconvolution operation as it does not compute a true inverse of convolution). For more information, see the visualizations here and the Deconvolutional Networks paper.

This module supports TensorFloat32.

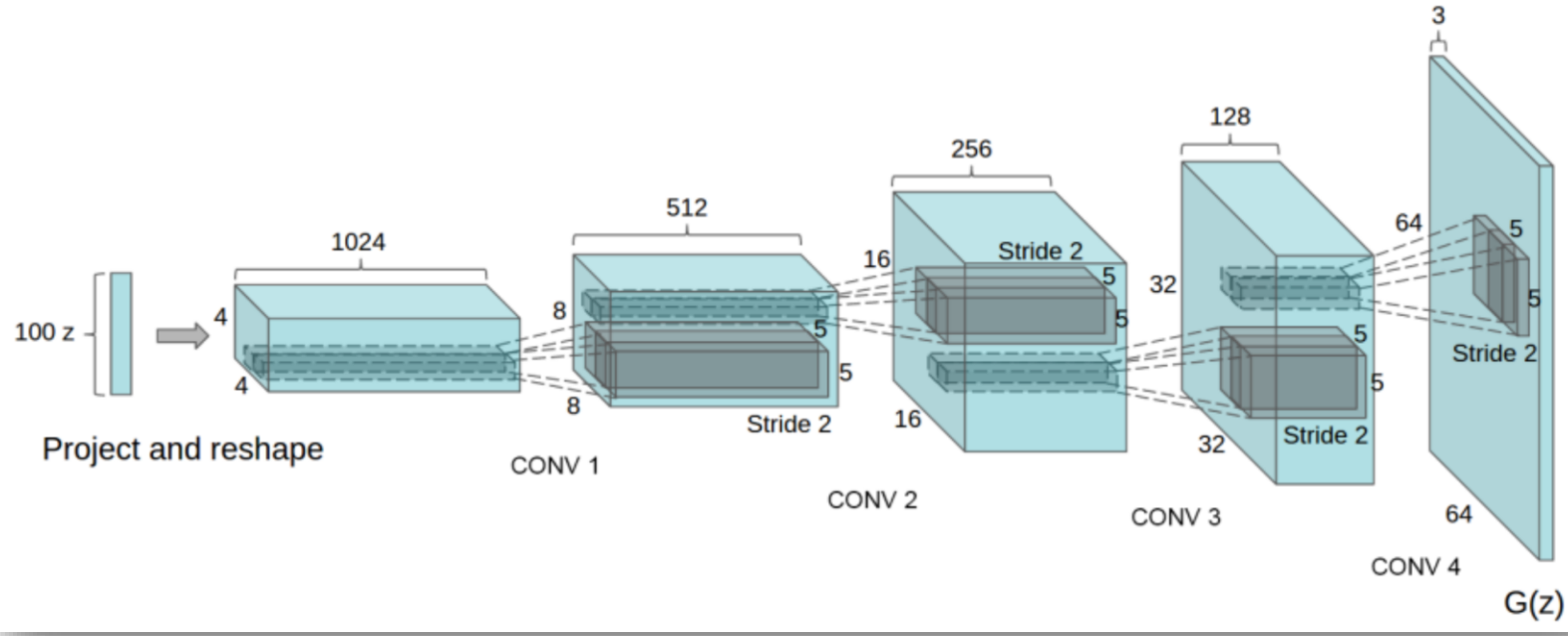
On certain ROCm devices, when using float16 inputs this module will use different precision for backward.

- `stride` controls the stride for the cross-correlation.

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
https://d2l.ai/chapter_computer-vision/transposed-conv.html
https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html

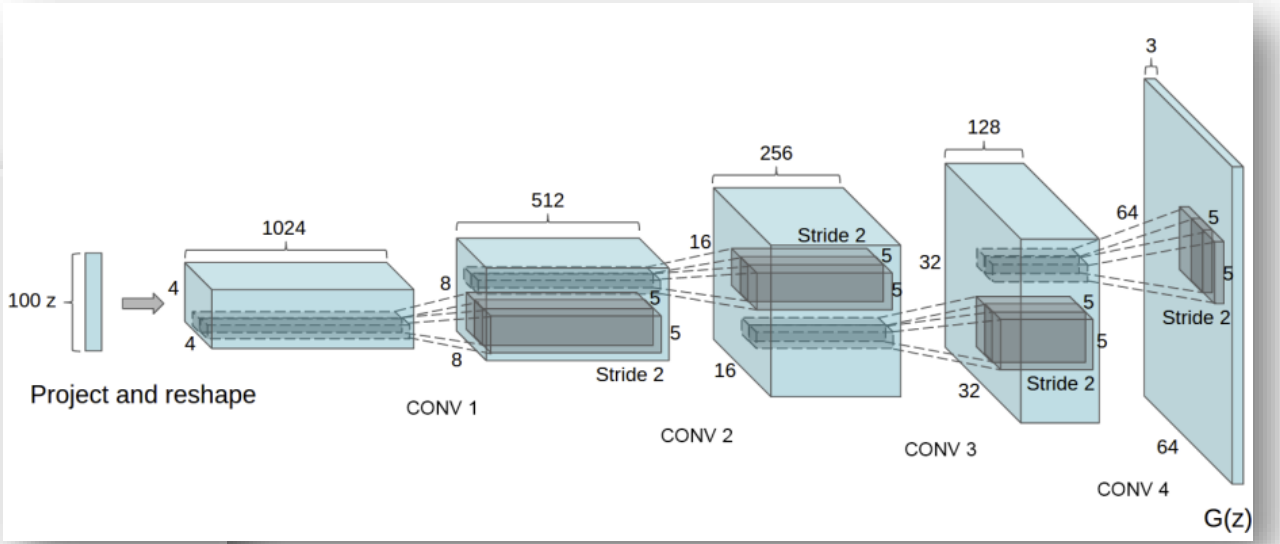
Generator

The generator, G, is designed to map the latent space vector (z) to data-space. Since our data are images, converting z to data-space means ultimately creating a RGB image with the same size as the training images (i.e. 3x64x64). In practice, this is accomplished through a series of strided two dimensional convolutional transpose layers, each paired with a 2d batch norm layer and a relu activation. The output of the generator is fed through a tanh function to return it to the input data range of $[-1, 1]$. It is worth noting the existence of the batch norm functions after the conv-transpose layers, as this is a critical contribution of the DCGAN paper. These layers help with the flow of gradients during training. An image of the generator from the DCGAN paper is shown below.



https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

```python
# Generator Code

class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. ``(ngf*8) x 4 x 4``
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. ``(ngf*4) x 8 x 8``
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. ``(ngf*2) x 16 x 16``
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. ``(ngf) x 32 x 32``
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. ``(nc) x 64 x 64``
        )

    def forward(self, input):
        return self.main(input)
```



https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

Discriminator

As mentioned, the discriminator, D, is a binary classification network that takes an image as input and outputs a scalar probability that the input image is real (as opposed to fake). Here, D takes a 3x64x64 input image, processes it through a series of Conv2d, BatchNorm2d, and LeakyReLU layers, and outputs the final probability through a Sigmoid activation function. This architecture can be extended with more layers if necessary for the problem, but there is significance to the use of the strided convolution, BatchNorm, and LeakyReLUs. The DCGAN paper mentions it is a good practice to use strided convolution rather than pooling to downsample because it lets the network learn its own pooling function. Also batch norm and leaky relu functions promote healthy gradient flow which is critical for the learning process of both G and D.

Discriminator Code

```
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is ``(nc) x 64 x 64``
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf) x 32 x 32``
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*2) x 16 x 16``
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*4) x 8 x 8``
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*8) x 4 x 4``
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

Loss Functions and Optimizers

With D and G setup, we can specify how they learn through the loss functions and optimizers. We will use the Binary Cross Entropy loss (BCELoss) function which is defined in PyTorch as:

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

Notice how this function provides the calculation of both log components in the objective function (i.e. $log(D(x))$ and $log(1 - D(G(z)))$). We can specify what part of the BCE equation to use with the y input. This is accomplished in the training loop which is coming up soon, but it is important to understand how we can choose which component we wish to calculate just by changing y (i.e. GT labels).

Next, we define our real label as 1 and the fake label as 0. These labels will be used when calculating the losses of D and G, and this is also the convention used in the original GAN paper. Finally, we set up two separate optimizers, one for D and one for G. As specified in the DCGAN paper, both are Adam optimizers with learning rate 0.0002 and Beta1 = 0.5. For keeping track of the generator's learning progression, we will generate a fixed batch of latent vectors that are drawn from a Gaussian distribution (i.e. fixed_noise) . In the training loop, we will periodically input this fixed_noise into G, and over the iterations we will see images form out of the noise.

```
# Initialize the ``BCELoss`` function
criterion = nn.BCELoss()
```

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

the concept is similar to the previous slides where the GAN principle was explained

Training

Finally, now that we have all of the parts of the GAN framework defined, we can train it. Be mindful that training GANs is somewhat of an art form, as incorrect hyperparameter settings lead to mode collapse with little explanation of what went wrong. Here, we will closely follow Algorithm 1 from the Goodfellow's paper, while abiding by some of the best practices shown in ganhacks. Namely, we will "construct different mini-batches for real and fake" images, and also adjust G's objective function to maximize $log(D(G(z)))$. Training is split up into two main parts. Part 1 updates the Discriminator and Part 2 updates the Generator.

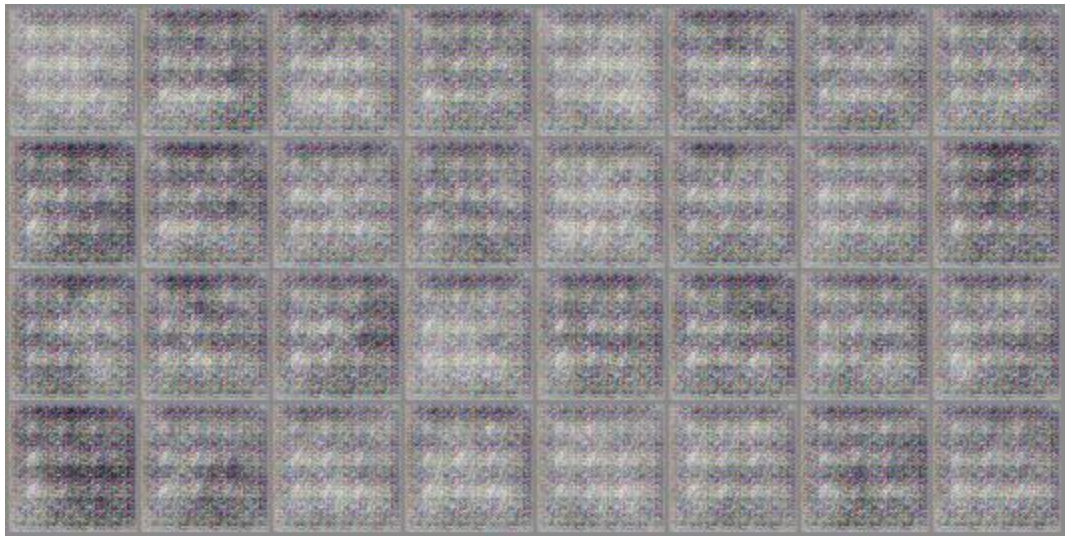
Part 1 - Train the Discriminator

Recall, the goal of training the discriminator is to maximize the probability of correctly classifying a given input as real or fake. In terms of Goodfellow, we wish to "update the discriminator by ascending its stochastic gradient". Practically, we want to maximize $log(D(x)) + log(1 - D(G(z)))$. Due to the separate mini-batch suggestion from ganhacks, we will calculate this in two steps. First, we will construct a batch of real samples from the training set, forward pass through D, calculate the loss ($log(D(x))$), then calculate the gradients in a backward pass. Secondly, we will construct a batch of fake samples with the current generator, forward pass this batch through D, calculate the loss ($log(1 - D(G(z)))$), and *accumulate* the gradients with a backward pass. Now, with the gradients accumulated from both the all-real and all-fake batches, we call a step of the Discriminator's optimizer.

Part 2 - Train the Generator

As stated in the original paper, we want to train the Generator by minimizing $log(1 - D(G(z)))$ in an effort to generate better fakes. As mentioned, this was shown by Goodfellow to not provide sufficient gradients, especially early in the learning process. As a fix, we instead wish to maximize $log(D(G(z)))$. In the code we accomplish this by: classifying the Generator output from Part 1 with the Discriminator, computing G's loss *using real labels as GT*, computing G's gradients in a backward pass, and finally updating G's parameters with an optimizer step. It may seem counter-intuitive to use the real labels as GT labels for the loss function, but this allows us to use the $log(x)$ part of the `BCELoss` (rather than the $log(1 - x)$ part) which is exactly what we want.

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html



https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

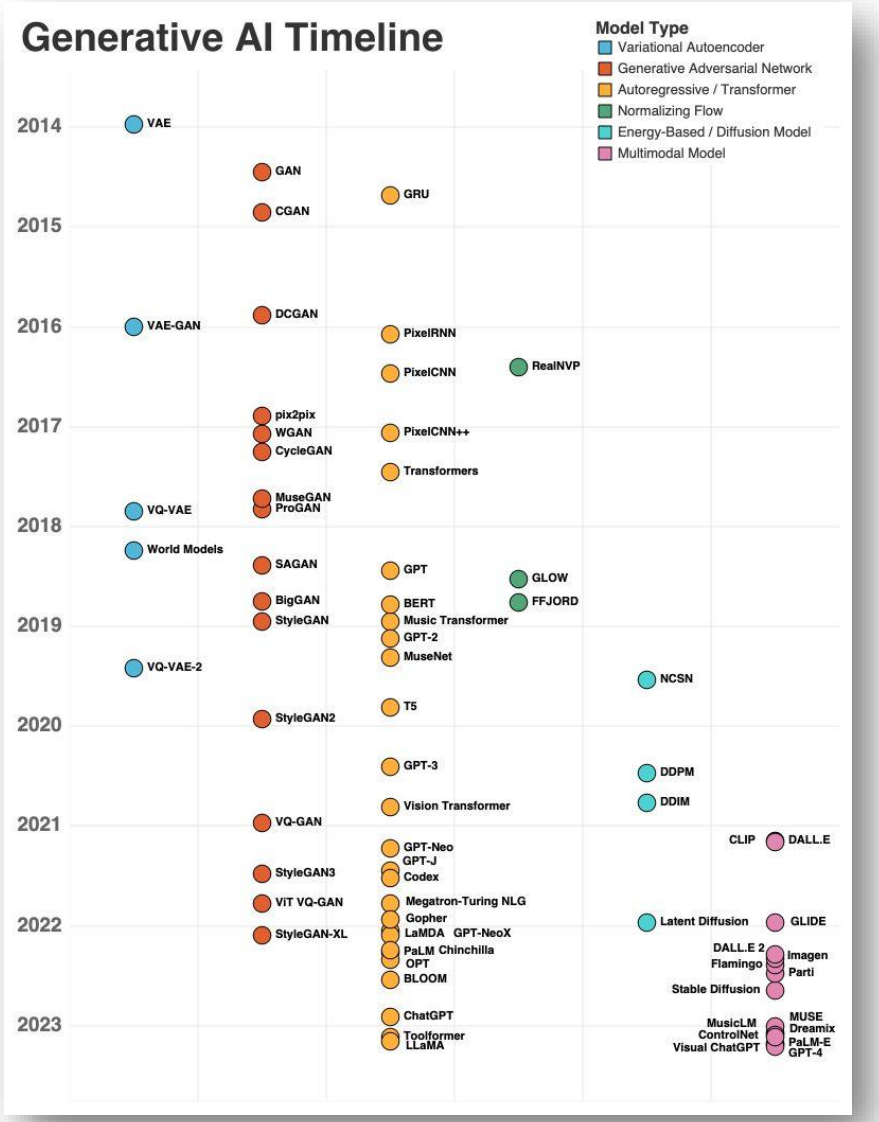
Real Images

Fake Images

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

Where to go from here?

https://www.kaggle.com/discussions/getting-started/397345



https://arxiv.org/pdf/2206.02262

Where to go from here?

Wasserstein Generative Adversarial Networks

Martin Arjovsky [1] **Soumith Chintala** [2] **Léon Bottou** [1 2]

Abstract

We introduce a new algorithm named WGAN, an alternative to traditional GAN training. In this new model, we show that we can improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyperparameter searches. Furthermore, we show that the corresponding optimization problem is sound, and provide extensive theoretical work highlighting the deep connections to different distances between distributions.

1. Introduction

The problem this paper is concerned with is that of unsupervised learning. Mainly, what does it mean to learn a probability distribution? The classical answer to this is to learn a probability density. This is often done by defining a parametric family of densities $(P_\theta)_{\theta \in \mathbb{R}^d}$ and finding the one that maximized the likelihood on our data: if we have real data examples $\{x^{(i)}\}_{i=1}^m$, we would solve the problem

The typical remedy is to add a noise term to the model distribution. This is why virtually all generative models described in the classical machine learning literature include a noise component. In the simplest case, one assumes a Gaussian noise with relatively high bandwidth in order to cover all the examples. It is well known, for instance, that in the case of image generation models, this noise degrades the quality of the samples and makes them blurry. For example, we can see in the recent paper (Wu et al., 2016) that the optimal standard deviation of the noise added to the model when maximizing likelihood is around 0.1 to each pixel in a generated image, when the pixels were already normalized to be in the range $[0, 1]$. This is a very high amount of noise, so much that when papers report the samples of their models, they don't add the noise term on which they report likelihood numbers. In other words, the added noise term is clearly incorrect for the problem, but is needed to make the maximum likelihood approach work.

Rather than estimating the density of $\mathbb{P}_r$ which may not exist, we can define a random variable Z with a fixed distribution $p(z)$ and pass it through a parametric function $g_\theta : \mathcal{Z} \to \mathcal{X}$ (typically a neural network of some kind)

https://proceedings.mlr.press/v70/arjovsky17a.html

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Where to go from here?

Published as a conference paper at ICLR 2018

PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION

Tero Karras NVIDIA
Timo Aila NVIDIA
Samuli Laine NVIDIA
Jaakko Lehtinen NVIDIA and Aalto University
{tkarras,taila,slaine,jlehtinen}@nvidia.com

ABSTRACT

We describe a new training methodology for generative adversarial networks. The key idea is to grow both the generator and discriminator progressively: starting from a low resolution, we add new layers that model increasingly fine details as training progresses. This both speeds the training up and greatly stabilizes it, allowing us to produce images of unprecedented quality, e.g., CELEBA images at 1024^2. We also propose a simple way to increase the variation in generated images, and achieve a record inception score of 8.80 in unsupervised CIFAR10. Additionally, we describe several implementation details that are important for discouraging unhealthy competition between the generator and discriminator. Finally, we suggest a new metric for evaluating GAN results, both in terms of image quality and variation. As an additional contribution, we construct a higher-quality version of the CELEBA dataset.

https://arxiv.org/abs/1710.10196

Where to go from here?

Published as a conference paper at ICLR 2023

DIFFUSION-GAN: TRAINING GANS WITH DIFFUSION

Zhendong Wang[1,2], **Huangjie Zheng**[1,2], **Pengcheng He**[2], **Weizhu Chen**[2], **Mingyuan Zhou**[1]
[1]The University of Texas at Austin, [2]Microsoft Azure AI
{*zhendong.wang, huangjie.zheng*}*@utexas.edu,* {*penhe,wzchen*}*@microsoft.com*
mingyuan.zhou@mccombs.utexas.edu

ABSTRACT

Generative adversarial networks (GANs) are challenging to train stably, and a promising remedy of injecting instance noise into the discriminator input has not been very effective in practice. In this paper, we propose Diffusion-GAN, a novel GAN framework that leverages a forward diffusion chain to generate Gaussian-mixture distributed instance noise. Diffusion-GAN consists of three components, including an adaptive diffusion process, a diffusion timestep-dependent discriminator, and a generator. Both the observed and generated data are diffused by the same adaptive diffusion process. At each diffusion timestep, there is a different noise-to-data ratio and the timestep-dependent discriminator learns to distinguish the diffused real data from the diffused generated data. The generator learns from the discriminator's feedback by backpropagating through the forward diffusion chain, whose length is adaptively adjusted to balance the noise and data levels. We theoretically show that the discriminator's timestep-dependent strategy gives consistent and helpful guidance to the generator, enabling it to match the true data distribution. We demonstrate the advantages of Diffusion-GAN over strong GAN baselines on various datasets, showing that it can produce more realistic images with higher stability and data efficiency than state-of-the-art GANs.

https://arxiv.org/pdf/2206.02262

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Where to go from here?

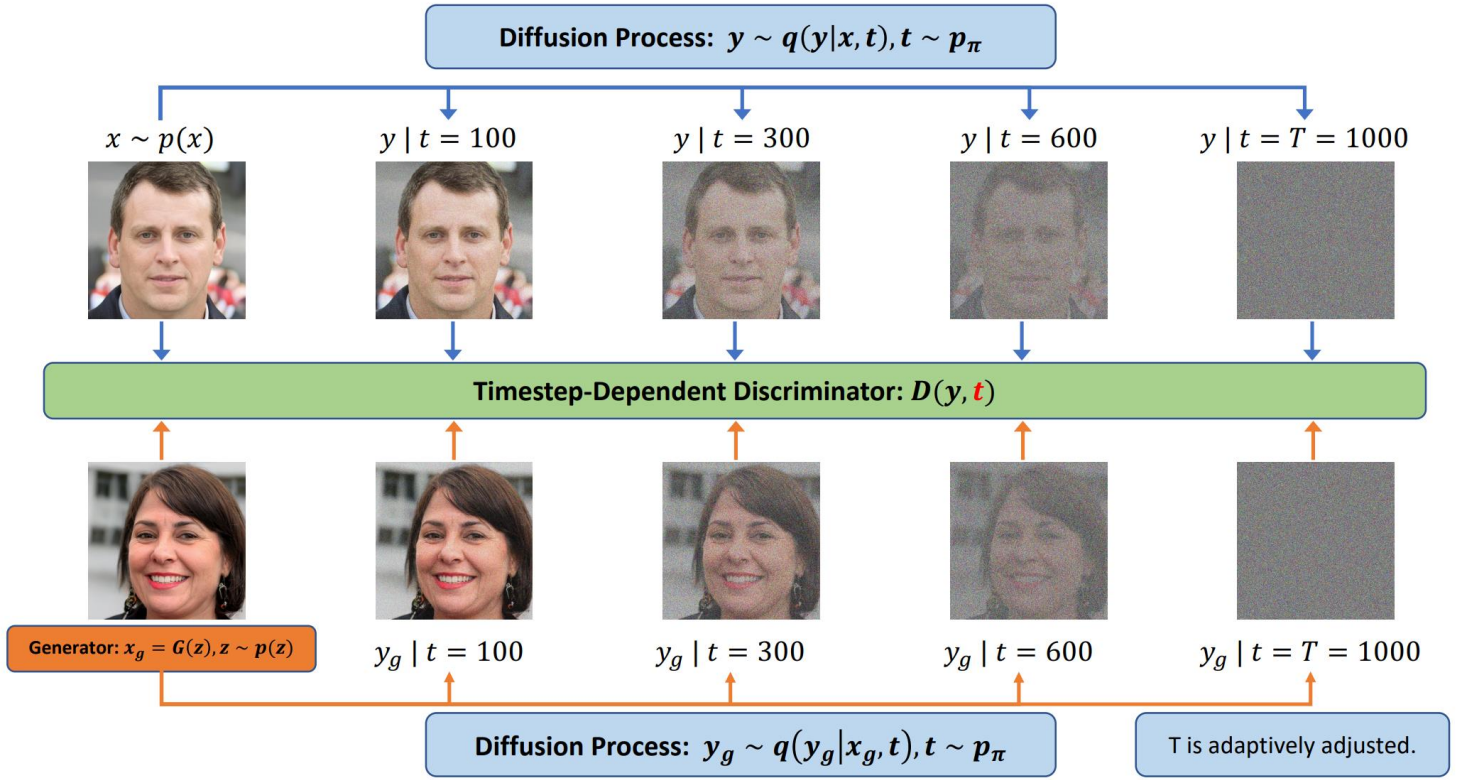


Figure 1: Flowchart for Diffusion-GAN. The top-row images represent the forward diffusion process of a real image, while the bottom-row images represent the forward diffusion process of a generated fake image. The discriminator learns to distinguish a diffused real image from a diffused fake image at all diffusion steps.

https://arxiv.org/pdf/2206.02262