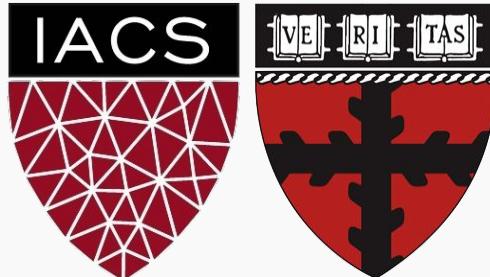


Advanced Section #8: Generative Adversarial Networks (GANs)

CS109B Data Science 2

Vincent Casser

Pavlos Protopapas



Outline

- Concept and Math
- Applications
- Common Problems
- Wasserstein GANs, Conditional GANs and CycleGANs
- Troubleshooting GANs
- **Hands-on:** Building an Image GAN in Keras
- Influential Papers and References



Concept

Generator

Job: Fool discriminator



Real



Generated

“Both are pandas!”

Discriminator

Job: Catch lies of the generator



Confidence: 0.9997



Confidence: 0.1617

“Nope”

Concept

Generator

Job: Fool discriminator



Generated



Real

“Both are pandas!”

Discriminator

Job: Catch lies of the generator



Confidence: 0.3759



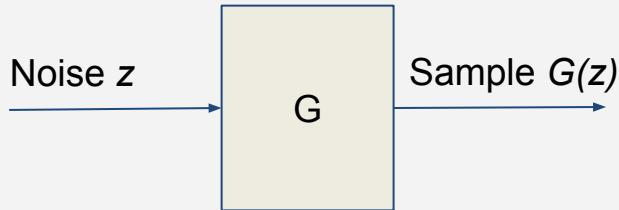
Confidence: 1.0

“Good try...”

GAN Structure

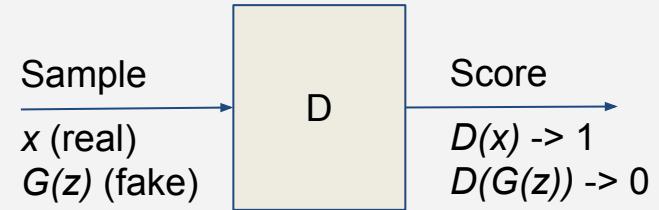
Generator

Job: Fool discriminator



Discriminator

Job: Catch lies of the generator



Math in a nutshell

Generator

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log \left(1 - D \left(G \left(\mathbf{z}^{(i)} \right) \right) \right)$$

m: Number of samples

z: Random noise samples

x: Real samples

How realistic are the generated samples?

G wants to maximize this.

Discriminator

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\underbrace{\log D \left(\mathbf{x}^{(i)} \right)}_{\text{Make sure real samples are classified as being real.}} + \underbrace{\log \left(1 - D \left(G \left(\mathbf{z}^{(i)} \right) \right) \right)}_{\text{Make sure generated samples are classified as unreal.}} \right]$$

Make sure real samples are classified as being real.

D wants to maximize this.

Make sure generated samples are classified as unreal.

D wants to minimize this.



Math in a nutshell

Generator

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log \left(1 - \boxed{D \left(G \left(z^{(i)} \right) \right)} \right)$$

m: Number of samples

z: Random noise samples

x: Real samples

Discriminator

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log \boxed{D \left(x^{(i)} \right)} + \log \left(1 - \boxed{D \left(G \left(z^{(i)} \right) \right)} \right) \right]$$

x



G(z)



$$D(x) = 0.9997$$

$$D(G(z)) = 0.1617$$

Generator	-	1.0
Discriminator	1.0	0.0

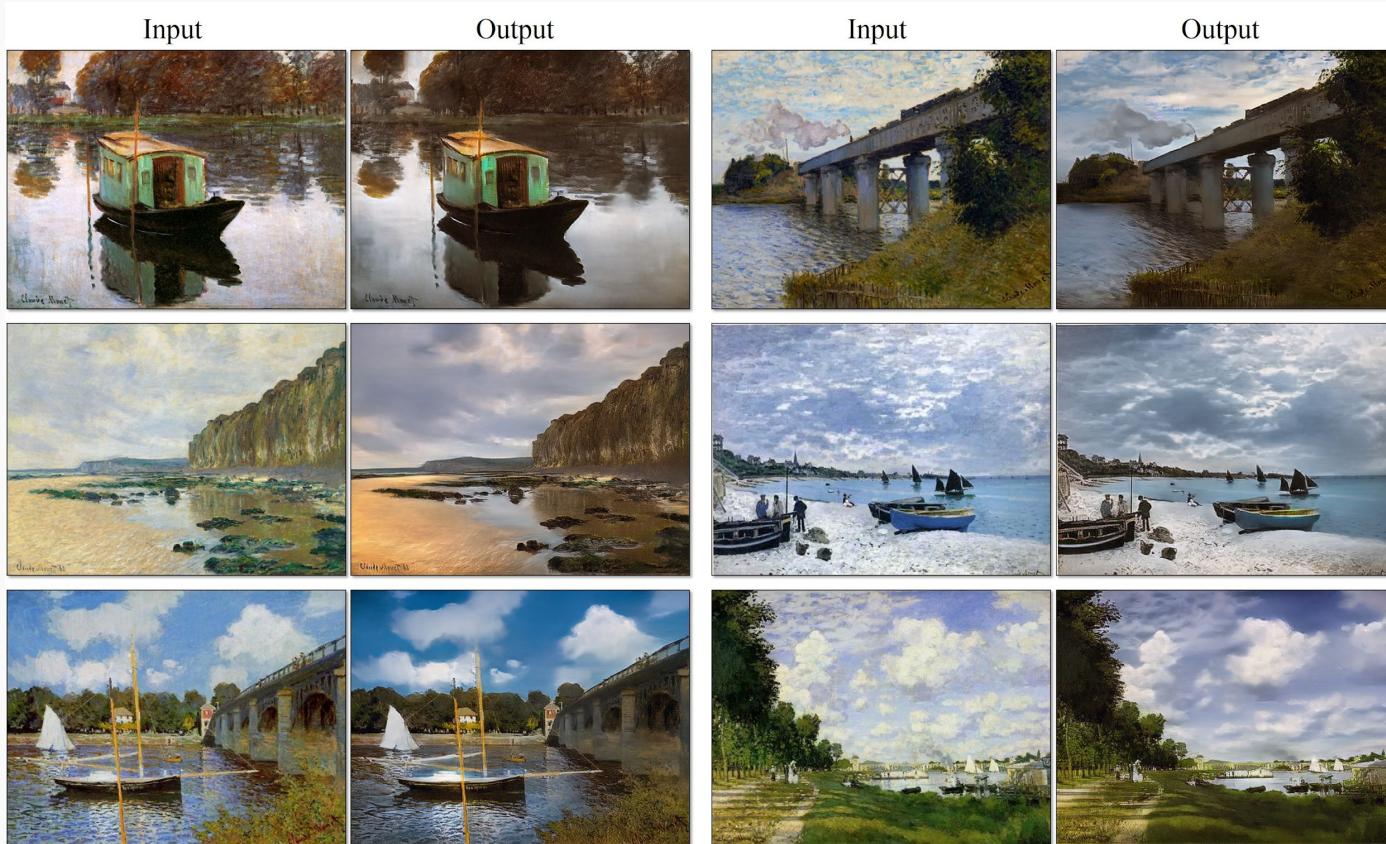
Targets



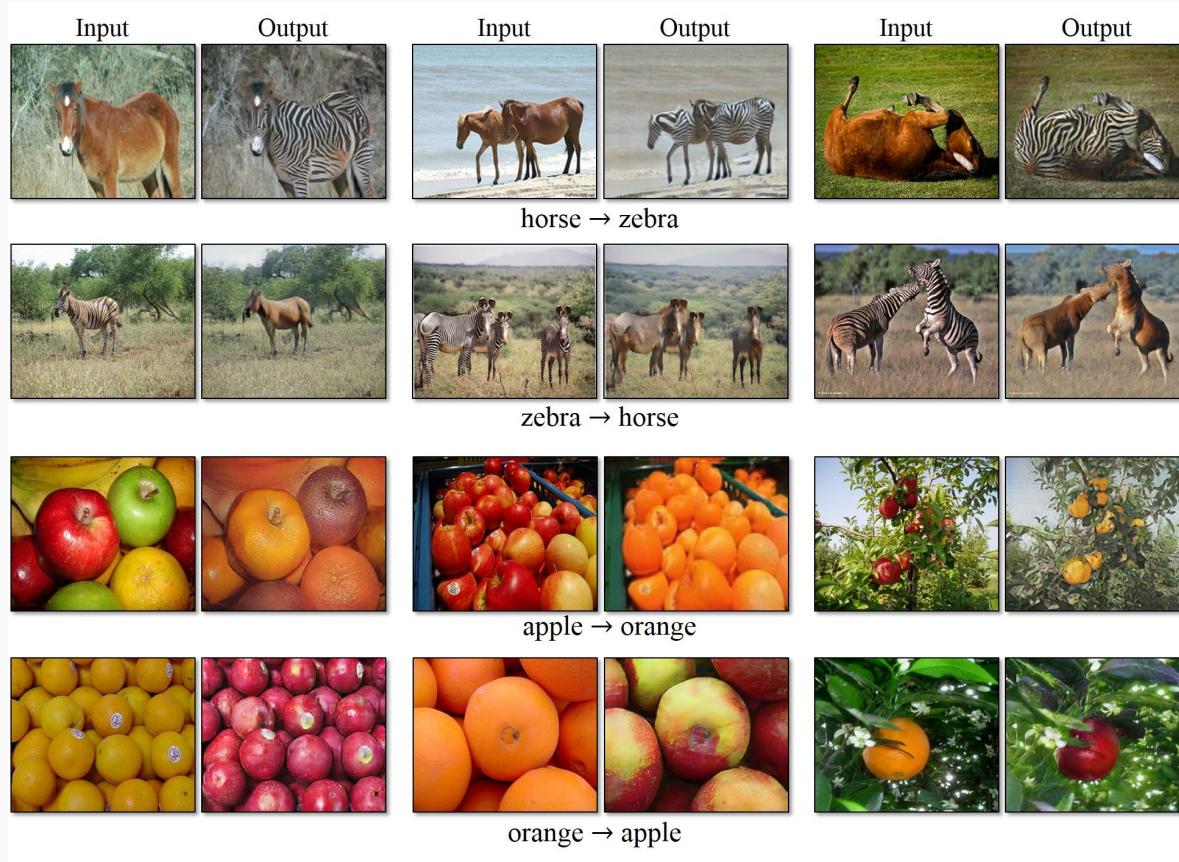
Applications

- (Conditional) synthesis
 - Font generation
 - Text2Image
 - 3D Object generation
- Data augmentation
 - Aiming to reduce need for labeled data
 - GAN is only used as a tool enhancing the training process of another model
- Style transfer and manipulation
 - Face Aging
 - Painting
 - Pose estimation and manipulation
 - Inpainting
 - Blending
- Signal super resolution

Applications: Style Transfer and Manipulation



Applications: Style Transfer and Manipulation



Applications: Style Transfer and Manipulation

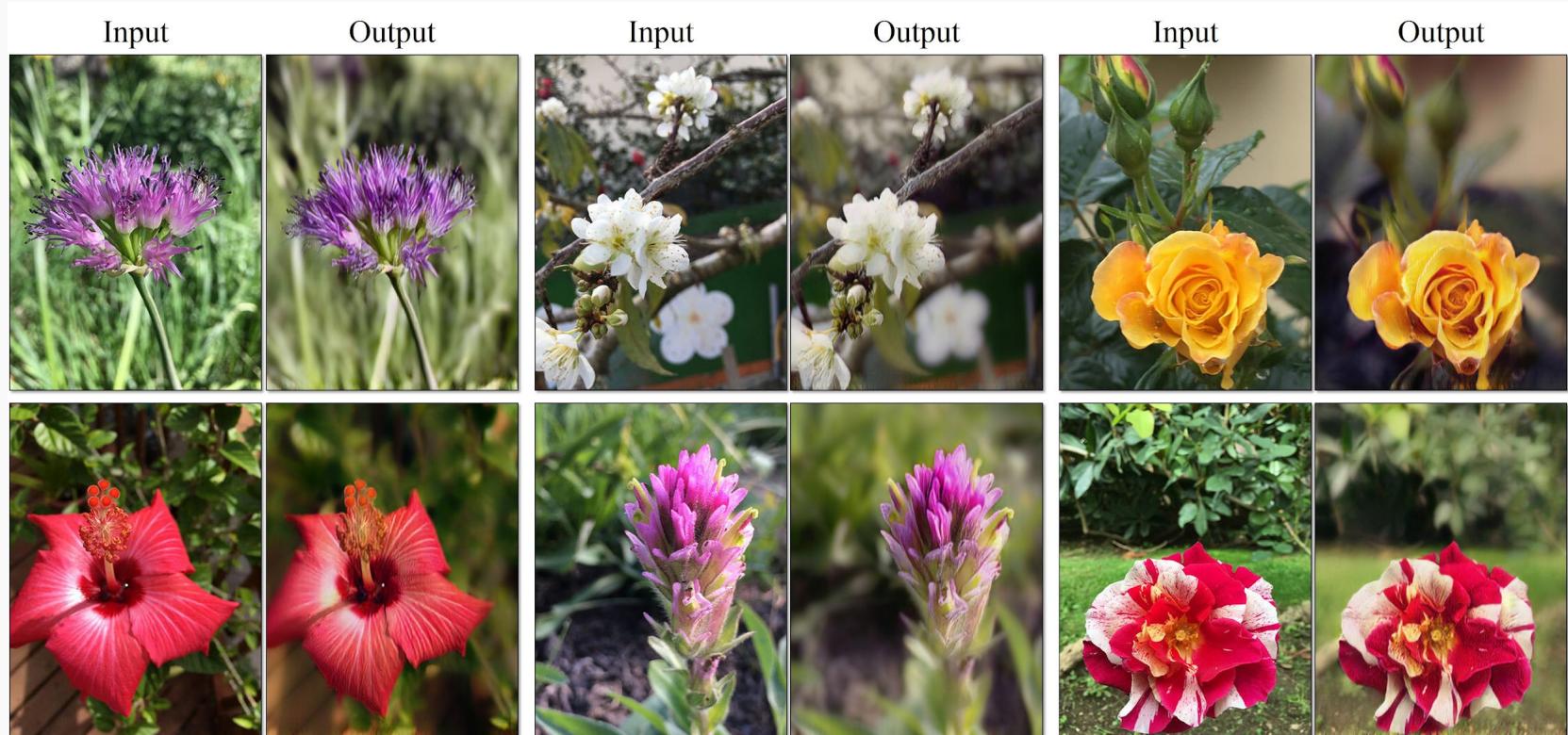


winter Yosemite → summer Yosemite



summer Yosemite → winter Yosemite

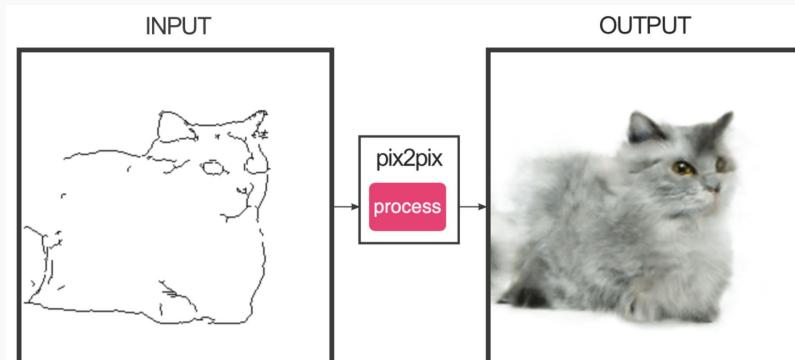
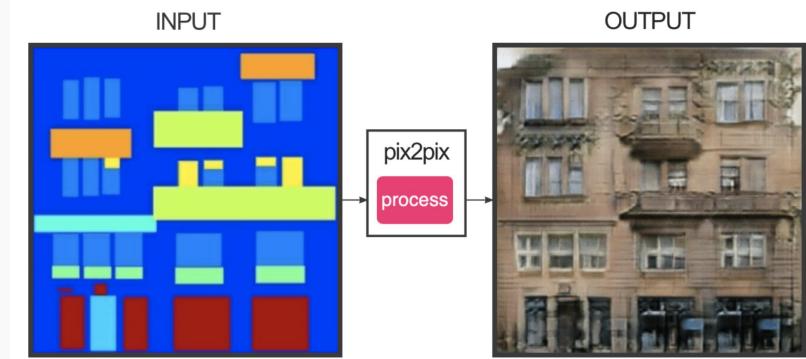
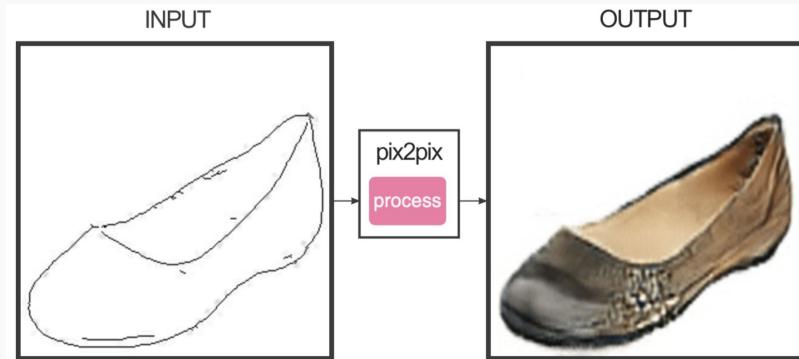
Applications: Style Transfer and Manipulation



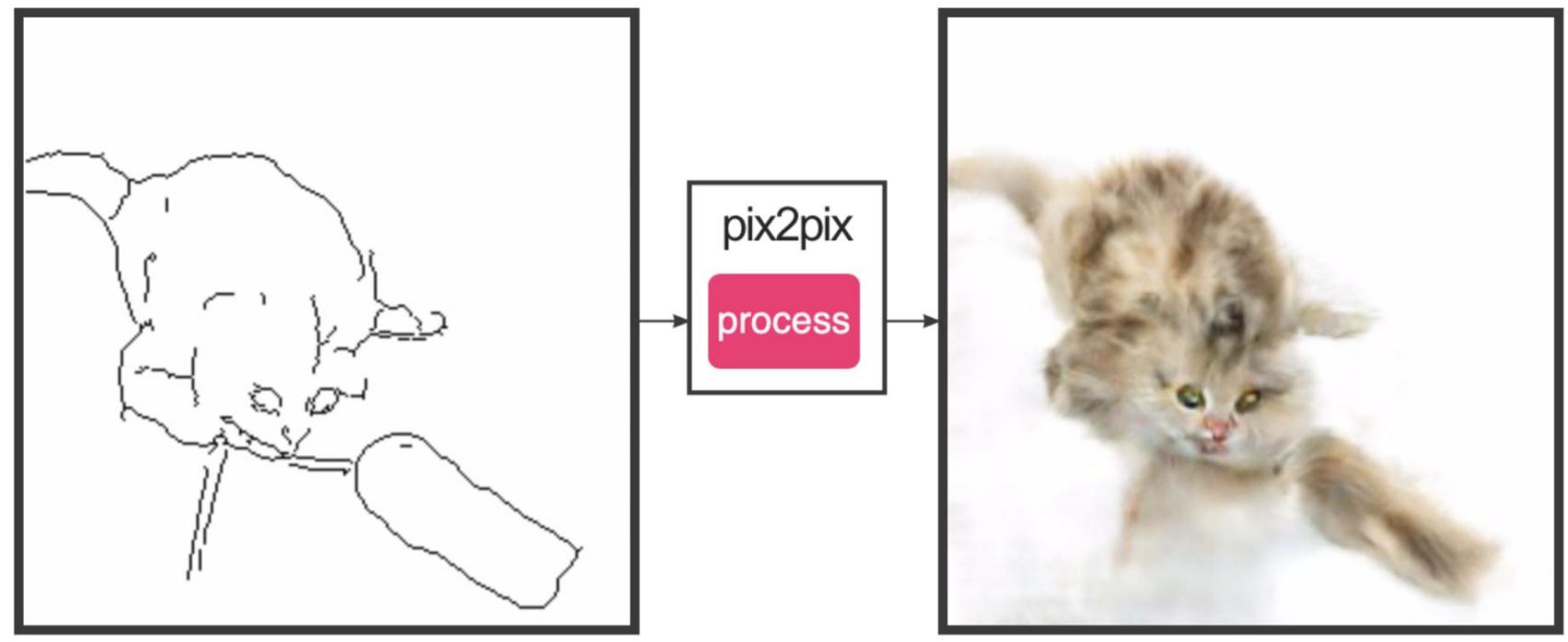
Applications: Style Transfer and Manipulation



Applications: Style Transfer and Manipulation



Applications: Style Transfer and Manipulation



Applications: Style Transfer and Manipulation



Applications: Style Transfer and Manipulation



Applications: Style Transfer and Manipulation



Applications: Signal Super Resolution



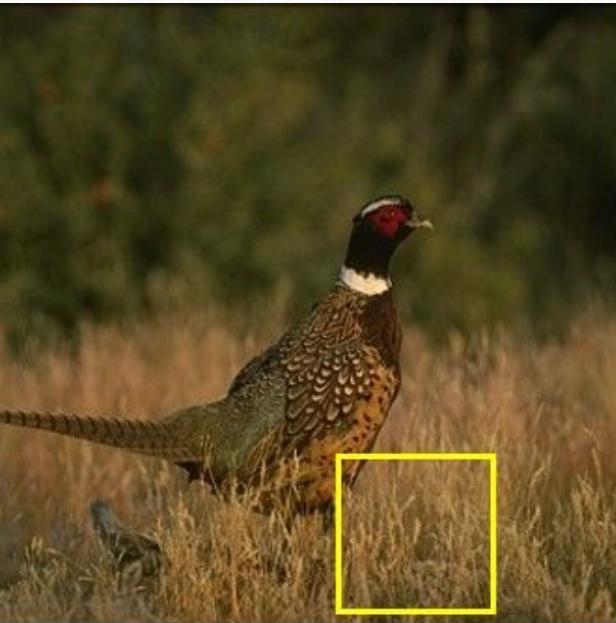
LG Image

SRGAN



Generated Image

Applications: Signal Super Resolution



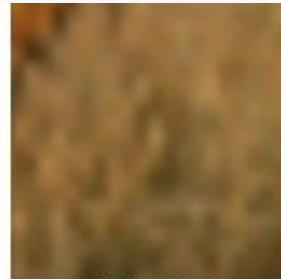
43074 from BSD100
(PSNR / Perceptual Index)



HR
(∞ / 2.31)



Bicubic
(29.29 / 7.35)



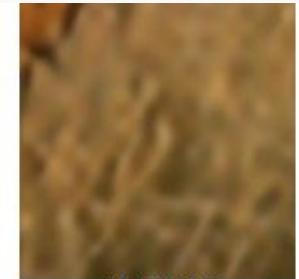
SRCCNN
(29.62 / 6.46)



EDSR
(29.76 / 6.25)



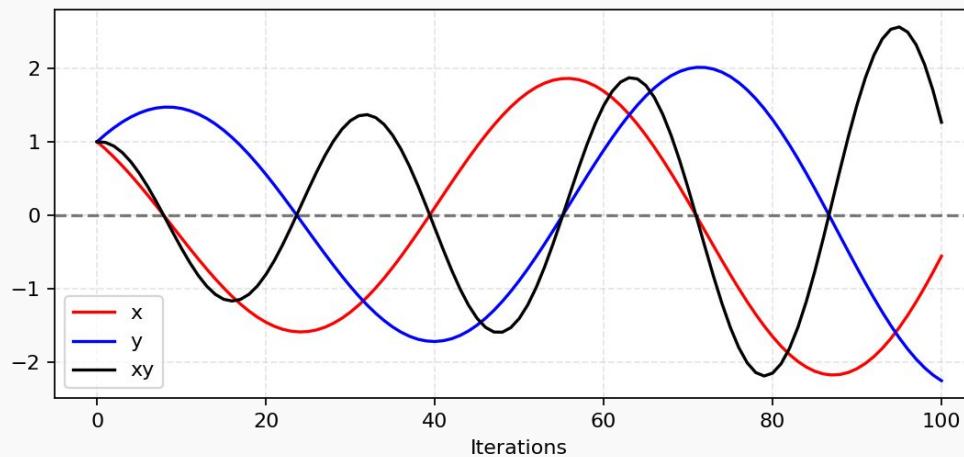
SRGAN
(27.29 / 2.74)



ESRGAN(ours)
(27.69 / 2.76)

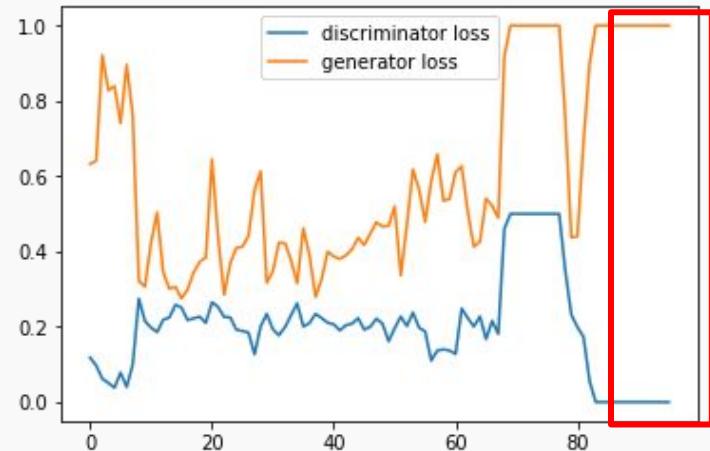
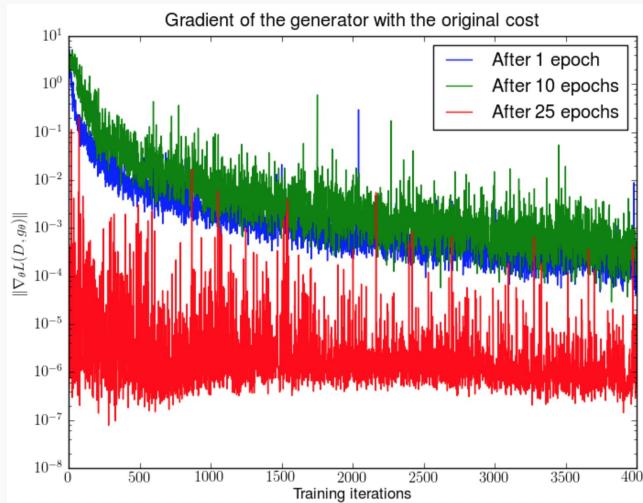
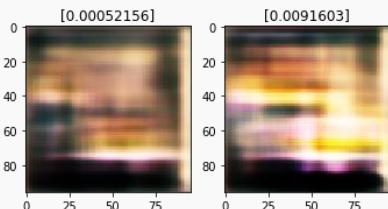
Common Problems: Oscillation

- Both generator and discriminator jointly searching for equilibrium, but model updates are independent.
- No theoretical convergence guarantees.
- Solution: Extensive hyperparameter-search, sometimes manual intervention.



Common Problems: Vanishing gradient

- Discriminator can become too strong to provide signal for the generator.
- Generator can learn to fool the discriminator consistently.
- Solution: Do (not) pretrain discriminator, or lower its learning rate relatively to the generator. Change the number of updates for generator/discriminator per iteration.



GANs and Game Theory

- Original GAN formulation based on zero-sum non-cooperative game.
- If one wins, the other one loses (minimax).
- GANs converge when G and D reach a Nash equilibrium: the optimal point of

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

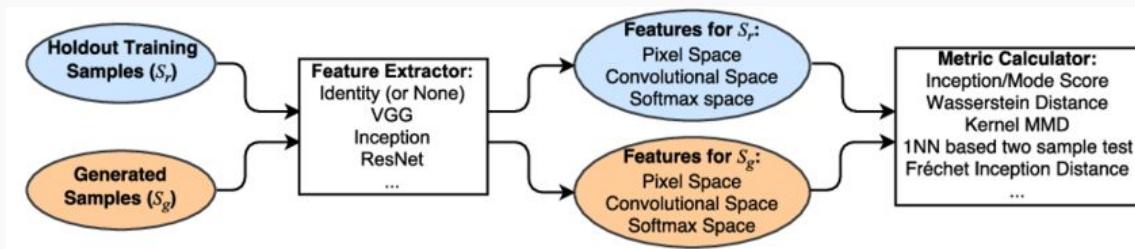
Common Problems: Mode collapse

- Generator can collapse so that it always produces the same samples.
- Generator restrained to small subspace generating samples of low diversity.
- Solution: Encourage diversity through minibatch discrimination (presenting the whole batch to the discriminator for review) or feature matching (add generator penalty for low diversity), or use multiple GANs



Common Problems: Evaluation metrics

- GANs are still evaluated on a very qualitative basis.
- Defining proper metrics is challenging. How does a “good” generator look like?
- Solution: Active research field and domain specific. Strong classification models are commonly used to judge the quality of generated samples



Inception score

TSTR score

Wasserstein GAN

- Using the standard GAN formulation, training is extremely unstable.
- Discriminator often improves too quickly for the generator to catch up.
- Careful balancing is needed.
- Mode collapse is frequent.

WGAN (Wasserstein GAN):

Arjovsky, M., Chintala, S. and Bottou, L., 2017. Wasserstein GAN.
arXiv preprint arXiv:1701.07875.

Wasserstein GAN

Distance is everything:

In general, generative models seek to minimize the distance between
real and learned distribution.

Wasserstein (also EM, Earth-Mover) distance:

“Informally, if the distributions are interpreted as two different ways of piling up a certain amount of dirt over the region D, the **Wasserstein distance** is the **minimum cost** of turning one pile into the other; where the cost is assumed to be amount of dirt moved times the distance by which it is moved.”



Wasserstein GAN

- Exact computation is intractable.
- Idea: Use a CNN to **approximate** Wasserstein distance.
- Here, we re-use the discriminator, whose outputs are now **unbounded**
- We define a custom loss function, in Keras:

```
K.mean(y_true * y_pred)
```

y_true here is chosen from $\{-1, 1\}$ according to real/fake

Idea: make predictions for one type as large as possible, for others as small as possible

Wasserstein GAN

The authors claim:

- Higher stability during training, less need for carefully balancing generator and discriminator.
- Meaningful loss metric, correlating well with sample quality.
- Mode collapse is rare.

Wasserstein GAN

Tips for implementing Wasserstein GAN in Keras.

- Leave the discriminator output unbounded, i.e. apply linear activation.
- Initialize with small weights to not run into clipping issues from the start.
- Remember to run sufficient discriminator updates. This is crucial in the WGAN setup.
- You can use the wasserstein surrogate loss implementation below.
- Clip discriminator weights by implementing your own keras constraint.

```
def wasserstein_loss(y_true, y_pred):
    return K.mean(y_true * y_pred)
```

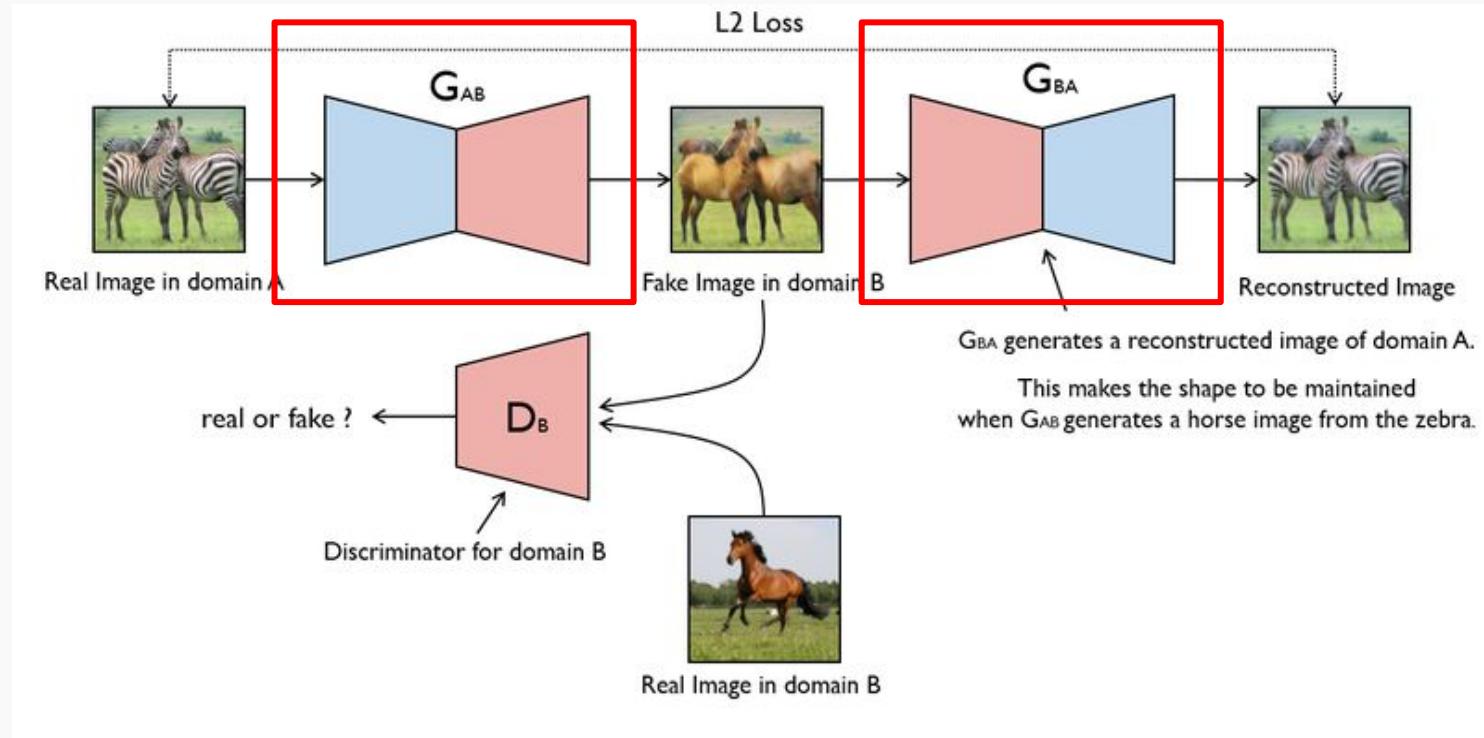
```
class WeightClip(keras.constraints.Constraint):
    def __init__(self, c):
        self.c = c

    def __call__(self, p):
        return K.clip(p, -self.c, self.c)

    def get_config(self):
        return {'name': self.__class__.__name__, 'c': self.c}
```



CycleGAN



CycleGAN

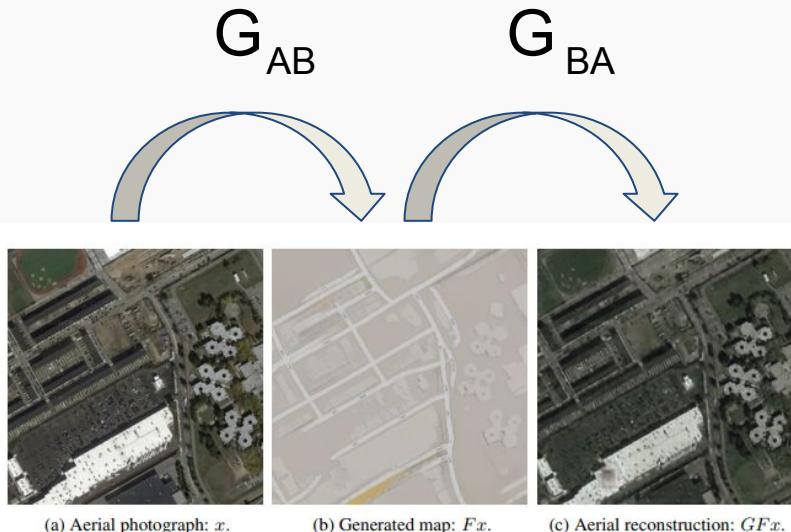
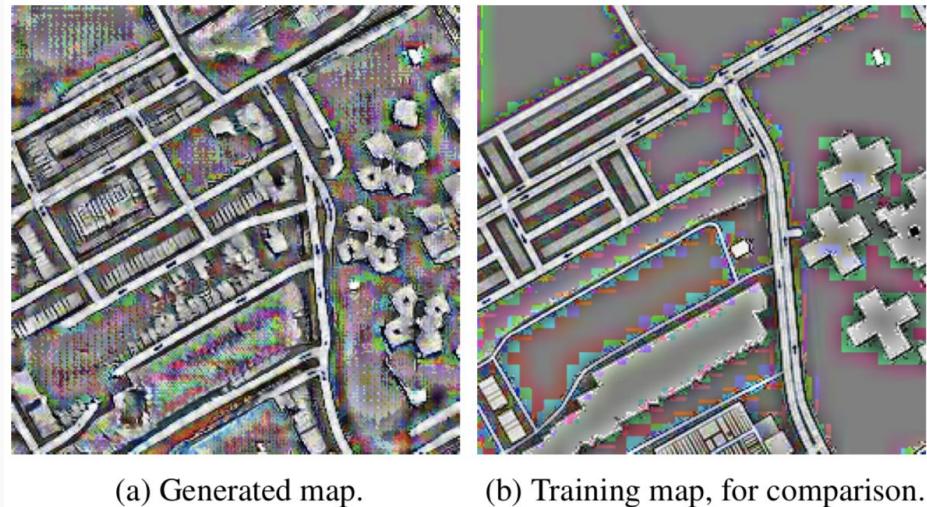


Figure 1: Details in x are reconstructed in GFx , despite not appearing in the intermediate map Fx .



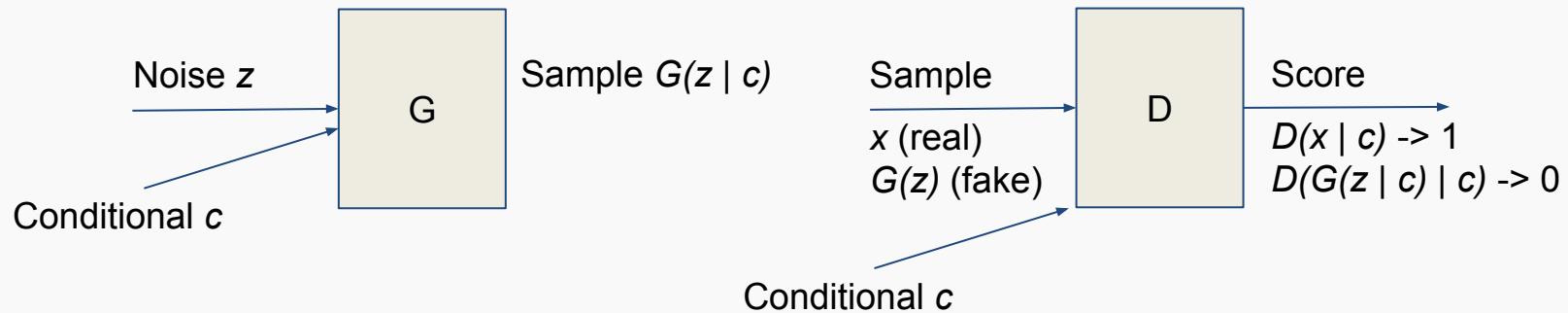
(a) Generated map.

(b) Training map, for comparison.

Generator G_{AB} learns to sneak in information for G_{BA}

Conditional GAN

- As in VAEs, GANs can simply be conditioned to generate a certain mode of data.



Troubleshooting GANs

GANs can be frustrating to work with. Here are some tips for your reference:

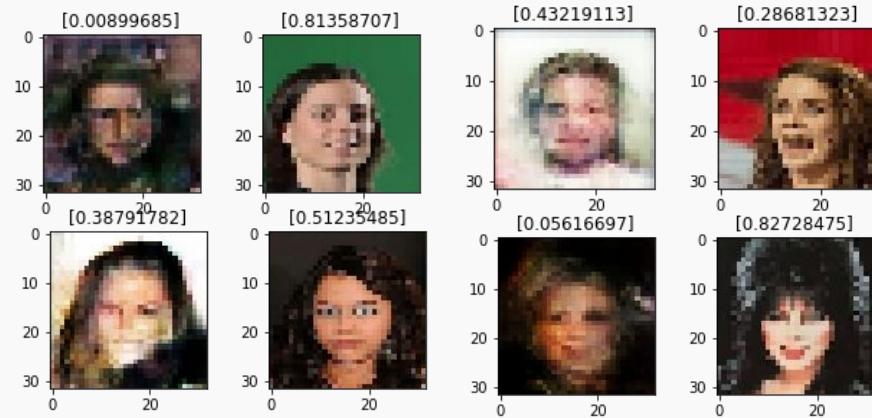
- **Models.** Make sure models are correctly defined. You can debug the discriminator alone by training on a vanilla image-classification task.
- **Data.** Normalize inputs properly to $[-1, 1]$. Make sure to use tanh as final activation for the generator in this case.
- **Noise.** Try sampling the noise vector from a normal distribution (not uniform).
- **Normalization.** Apply BatchNorm when possible, and send the real and fake samples in separate mini-batches.
- **Activations.** Use LeakyRelu instead of Relu.
- **Smoothing.** Apply label smoothing to avoid overconfidence when updating the discriminator, i.e. set targets for real images to less than 1.
- **Diagnostics.** Monitor the magnitude of gradients constantly.
- **Vanishing gradients.** If the discriminator becomes too strong (discriminator loss = 0), try decreasing its learning rate or update the generator more often.



Building an Image GAN

- Training a GAN can be frustrating and time-intensive.
- We will walk through a clean minimal example in Keras.
- Results are only on proof-of-concept level to enhance understanding. For state-of-the-art GANs, see references.

In the code example, if you don't tune parameters carefully, you won't surpass this level by much:

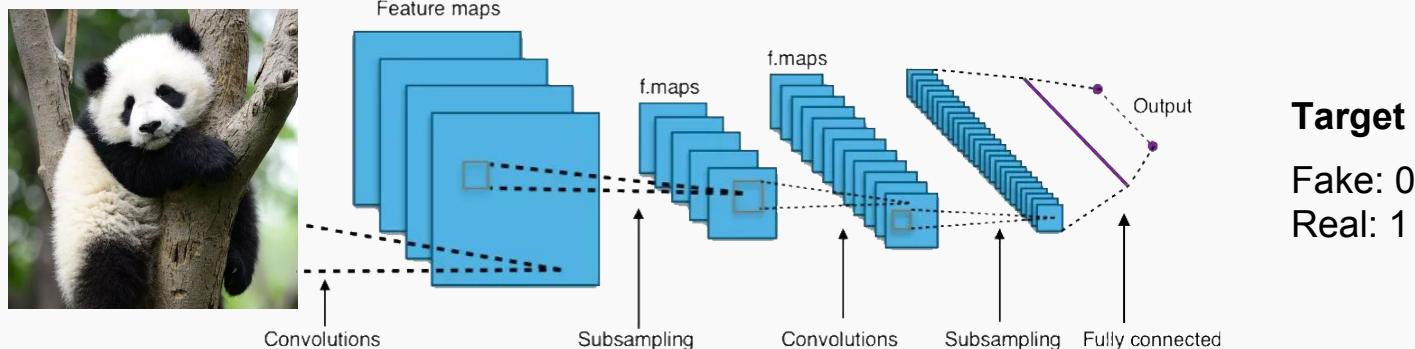


Building an Image GAN: Discriminator

Takes an **image** $[H, W, C]$ and outputs a **vector of** $[M]$, either class scores (classification) or single score quantifying photorealism.

Can be any image classification network, e.g. ResNet or DenseNet.

We use a minimalistic custom architecture.

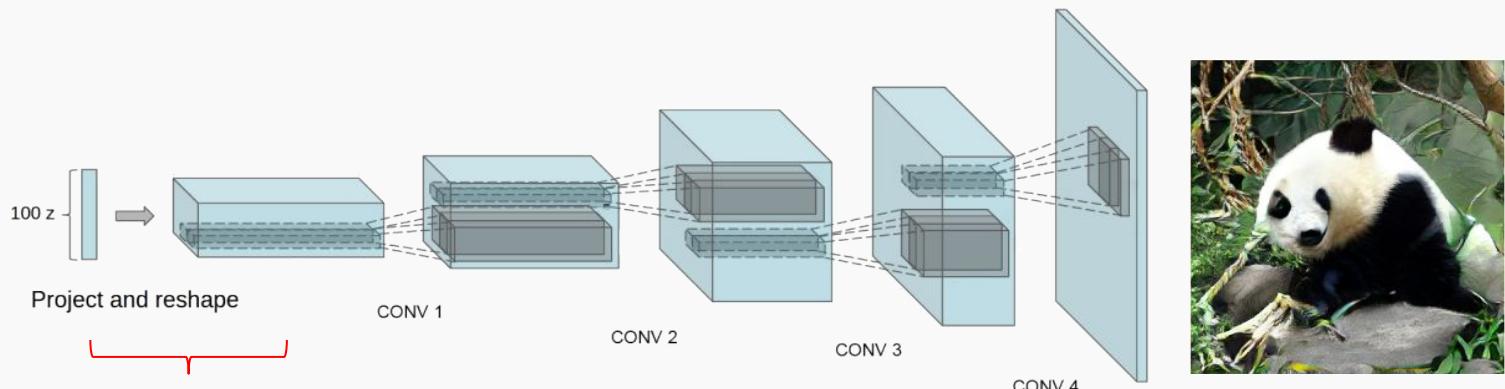


Building an Image GAN: Generator

Takes a vector of noise $[N]$ and outputs an image of $[H, W, C]$.

Network has to perform synthesis. Again, we use a very minimalistic custom architecture.

Source
Noise vector



In practice, the projection is usually done using a dense of $H \times W \times C$ units, followed by a reshape operation. You might want to regularize this part well.

Building an Image GAN: Full Setup

It is important to define the models properly in Keras, so that the weights of the respective models are **fixed** at the right time.

1. Define the discriminator model, and compile it.
2. Define the generator model, no need to compile.
3. Define an overall model comprised of these two, setting the discriminator to not trainable before the compilation:

```
model = keras.Sequential()  
model.add(generator)  
model.add(discriminator)  
discriminator.trainable = False  
model.compile(...)
```



Building an Image GAN: Training Loop

The training loop has to be executed manually:

- 
1. **Select** R real images from the training set.
 2. **Generate** F fake images by sampling random vectors of size N, and predicting images from them using the generator.
 3. **Train the discriminator** using `train_on_batch`: call it separately for the batch of R real images and F fake images, with the groundtruth being 1 and 0, respectively.
 4. **Sample** new random vectors of size N.
 5. **Train the full model** on the new vectors using `train_on_batch` with targets of 1. This will update the generator.

Building an Image GAN: Training Progress



Influential GAN-Papers (in order)

DCGAN	2015	https://arxiv.org/pdf/1511.06434v2.pdf
Wasserstein GAN (WGAN)	2017	https://arxiv.org/pdf/1701.07875.pdf
Conditional Generative Adversarial Nets (CGAN)	2014	https://arxiv.org/pdf/1411.1784v1.pdf
Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks (LAPGAN)	2015	https://arxiv.org/pdf/1506.05751.pdf
Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (SRGAN)	2016	https://arxiv.org/pdf/1609.04802.pdf
Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (CycleGAN)	2017	https://arxiv.org/pdf/1703.10593.pdf
InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets	2016	https://arxiv.org/pdf/1606.03657

Influential GAN-Papers (in order)

DCGAN	2017	https://arxiv.org/pdf/1704.00028.pdf
Improved Training of Wasserstein GANs (WGAN-GP)	2017	https://arxiv.org/pdf/1701.07875.pdf
Energy-based Generative Adversarial Network (EBGAN)	2016	https://arxiv.org/pdf/1609.03126.pdf
Autoencoding beyond pixels using a learned similarity metric (VAE-GAN)	2015	https://arxiv.org/pdf/1512.09300.pdf
Adversarial Feature Learning (BiGAN)	2016	https://arxiv.org/pdf/1605.09782v6.pdf
Stacked Generative Adversarial Networks (SGAN)	2016	https://arxiv.org/pdf/1612.04357.pdf
StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks	2016	https://arxiv.org/pdf/1710.10916.pdf
Learning from Simulated and Unsupervised Images through Adversarial Training (SimGAN)	2016	https://arxiv.org/pdf/1612.07828v1.pdf

Some Available GANs...

DCGAN	SGAN	AffGAN	RTT-GAN	SL-GAN
WGAN	SimGAN	TP-GAN	GANCS	Context-RNN-GAN
CGAN	VGAN	IcGAN	SSL-GAN	SketchGAN
LAPGAN	iGAN	ID-CGAN	MAD-GAN	GoGAN
SRGAN	3D-GAN	AnoGAN	PrGAN	RWGAN
CycleGAN	CoGAN	LS-GAN	AL-CGAN	MPM-GAN
WGAN-GP	CatGAN	Triple-GAN	ORGAN	MV-BiGAN
EBGAN	MGAN	TGAN	SD-GAN	
VAE-GAN	S^2GAN	BS-GAN	MedGAN	
BiGAN	LSGAN	MaIGAN	SGAN	



Sources and References

Run BigGAN in COLAB:

https://colab.research.google.com/github/tensorflow/hub/blob/master/examples/colab/big_gan_generation_with_tf_hub.ipynb

<https://www.jessicayung.com/explaining-tensorflow-code-for-a-convolutional-neural-network/>

<https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html>

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

<https://github.com/tensorlayer/srgan>

<https://junyanz.github.io/CycleGAN/>

<https://affinelayer.com/pixsrv/>

<https://tcwang0509.github.io/pix2pixHD/>