CIS 6930 Topics in Computing for Data Science
Week 4: Generative Adversarial Networks

9/21/2021

Yoshihiko (Yoshi) Suhara

2pm-3:20pm & 3:30pm-4:50pm

Week 3!

- ~~Week 1: Deep Learning Basics (Thu 9/9)~~
- ~~Week 2: AutoEncoder (Tue 9/14)~~
- ~~Week 3: Convolutional Neural Networks (Thu 9/16)~~
- **Week 4: GAN (Tue 9/21)**
- Week 5: Word embeddings: Word2vec, GloVe (Thu 9/23)
- Week 6: Recurrent Neural Networks (Tue 9/28, Thu 9/30)
- Week 7: Review/Project pitch & Mid-term (Tue 10/5, Thu 10/7)
- Fall Break
- ...

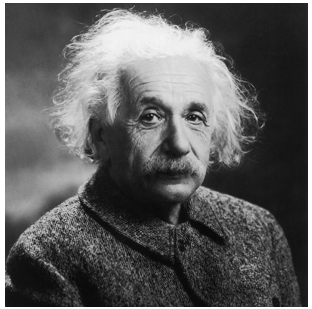
Course Policy about Questions

- Please use Zoom chat/Slack during the session
  - It seems that Zoom chat works better for us in classroom

- Please use the course Slack channel (**#topicsincomputing_fall21**)
  - To share the knowledge with classmates
  - If you have something that you don’t want to share with your classmates, please ask the question during the office hour

Why you should ask questions?

- To help bring yourself to the next level!



“You cannot solve a problem at the same level it was created” -- Albert Einstein

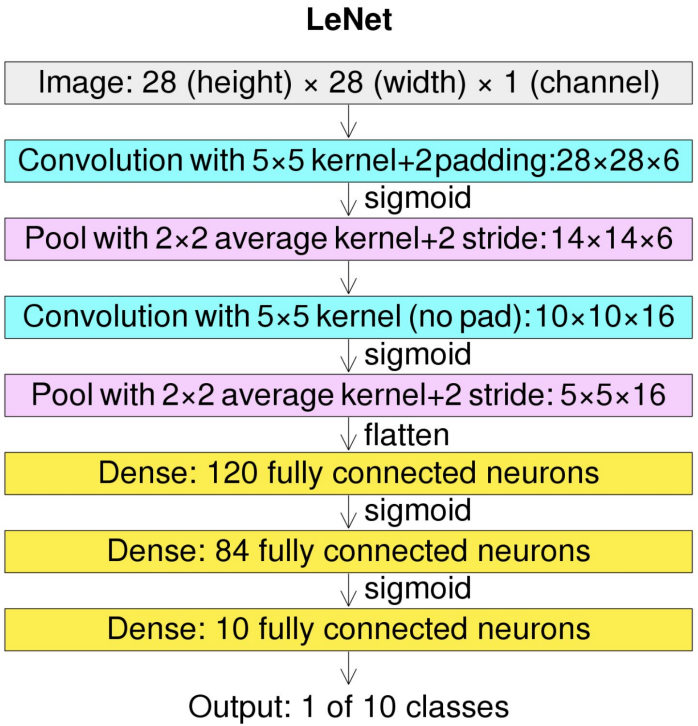
- That said, asking questions in public needs some courage (it shouldn’t be, though)
- If you don’t follow it, that’s the speaker’s fault. ;)
- “Open up a Q&A session with a naive question” -- my anecdote

Ask more, know more!

Recap

Key Concepts in CNN

- Convolution
- Pooling
- Fully-connected Layer



Convolution: Terminology

$x_{1,1}$	$x_{1,2}$	…	$x_{1,M}$
…			
…			
$x_{N,1}$	…		$x_{N,M}$

Input image /Feature map

*

w_1	w_2
w_3	w_4

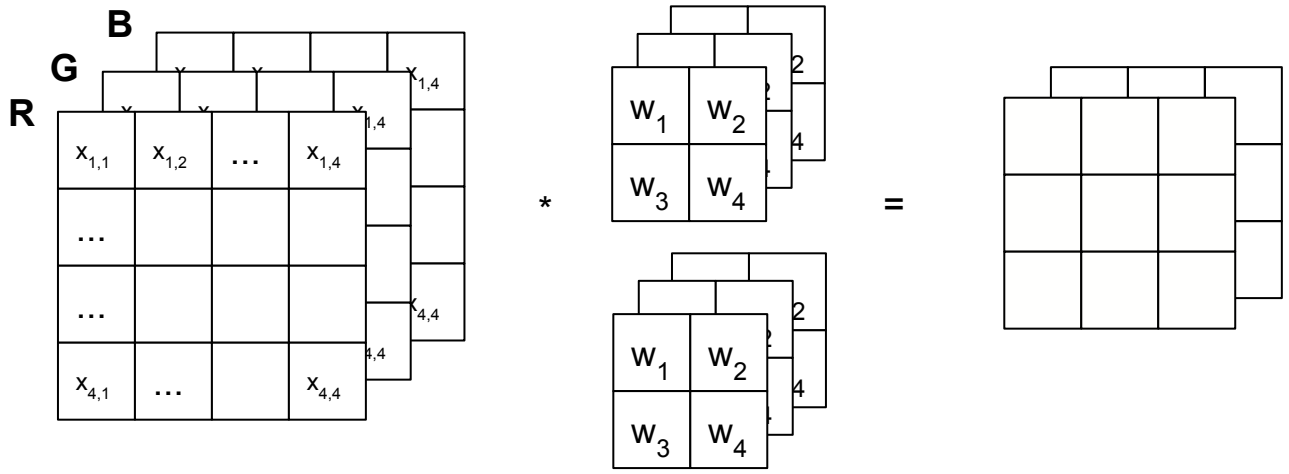
Convolutional filter (aka kernel)

=

Feature map

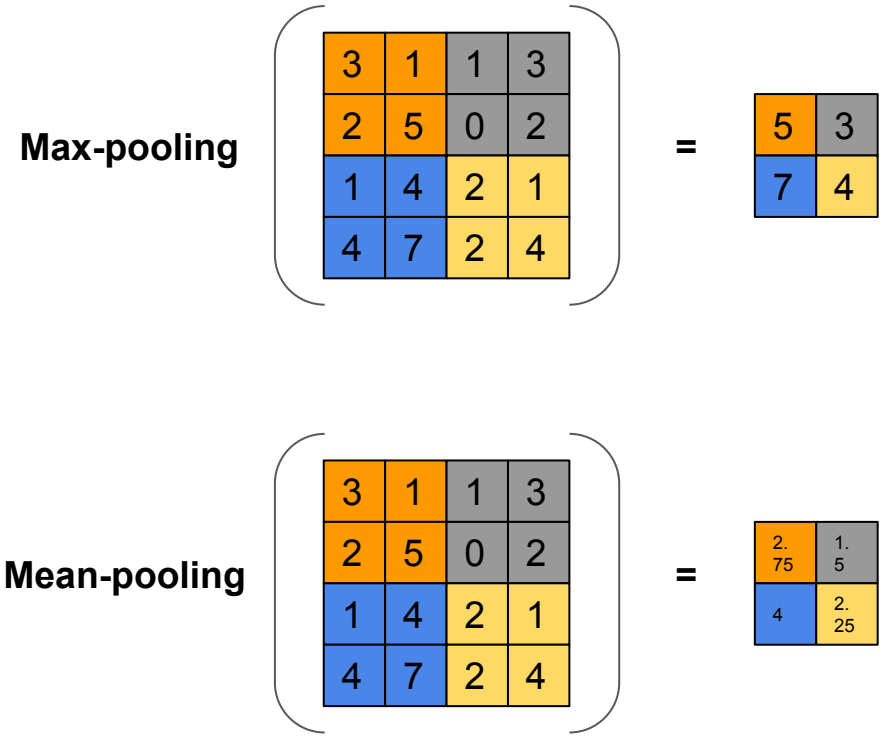
Convolutions on RGB image

```
Conv2d(in_channels=3, out_channels=2, kernel_size=2, stride=1, padding=0)
```



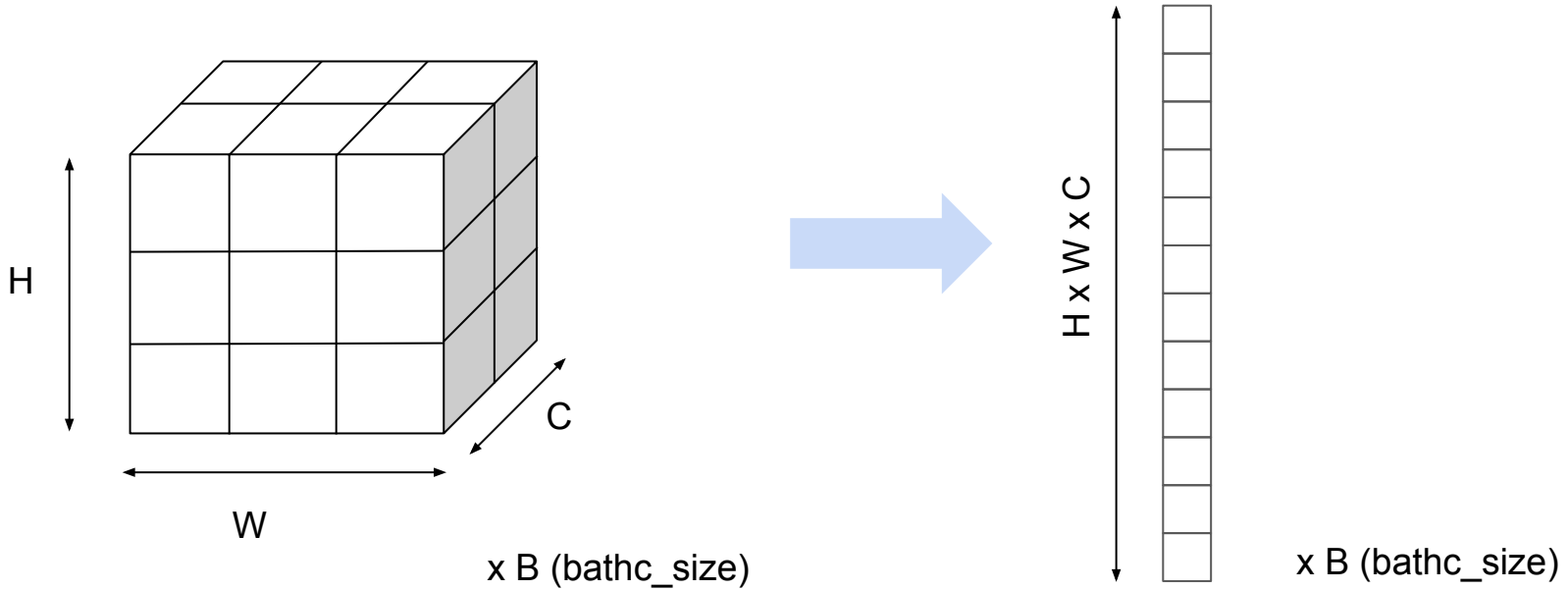
(1, 3, 4, 4) (2, 3, 2, 2) (1, **2**, 3, 3)

Mean and Max Pooling

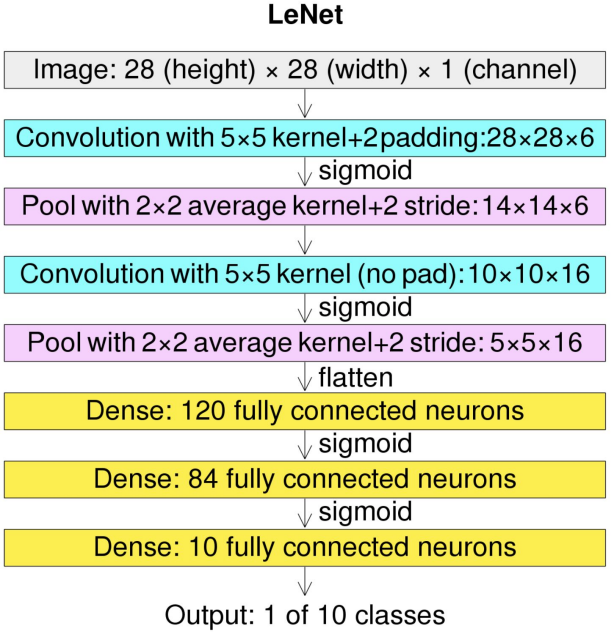


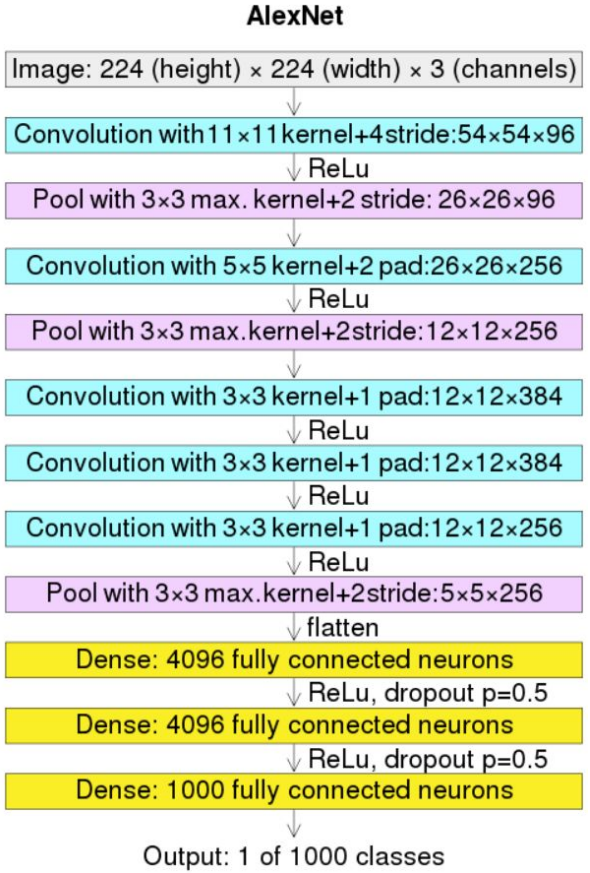
Flattening Feature Maps into Vectors

- Feature maps are not directly compatible with `Linear` layers
  - (batch_size, #channel, height, width) =! (batch_size, dim)



Model Design → Blueprint → PyTorch code





Week 3’s Slides: Advanced CNNs & Applications

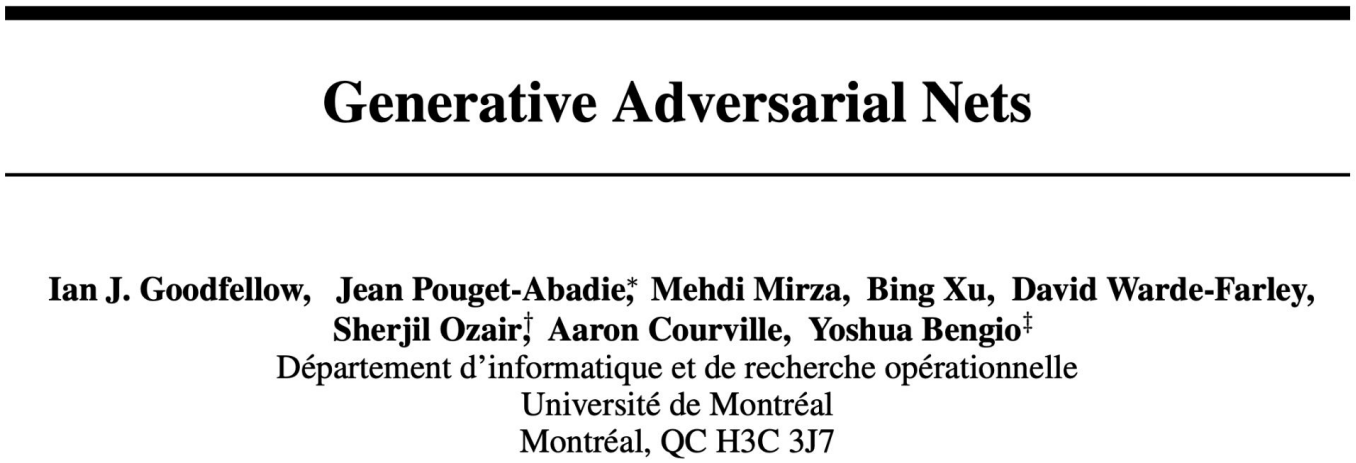
Questions?

Today’s Goal

- Learn the concept of **Generative Adversarial Networks (GANs)**
- Understand how you can implement GAN models with PyTorch
- Learn major GAN models and core techniques
  - GAN
  - DCGAN
  - Pix2Pix
  - CycleGAN

What are GANs? What are they used for?

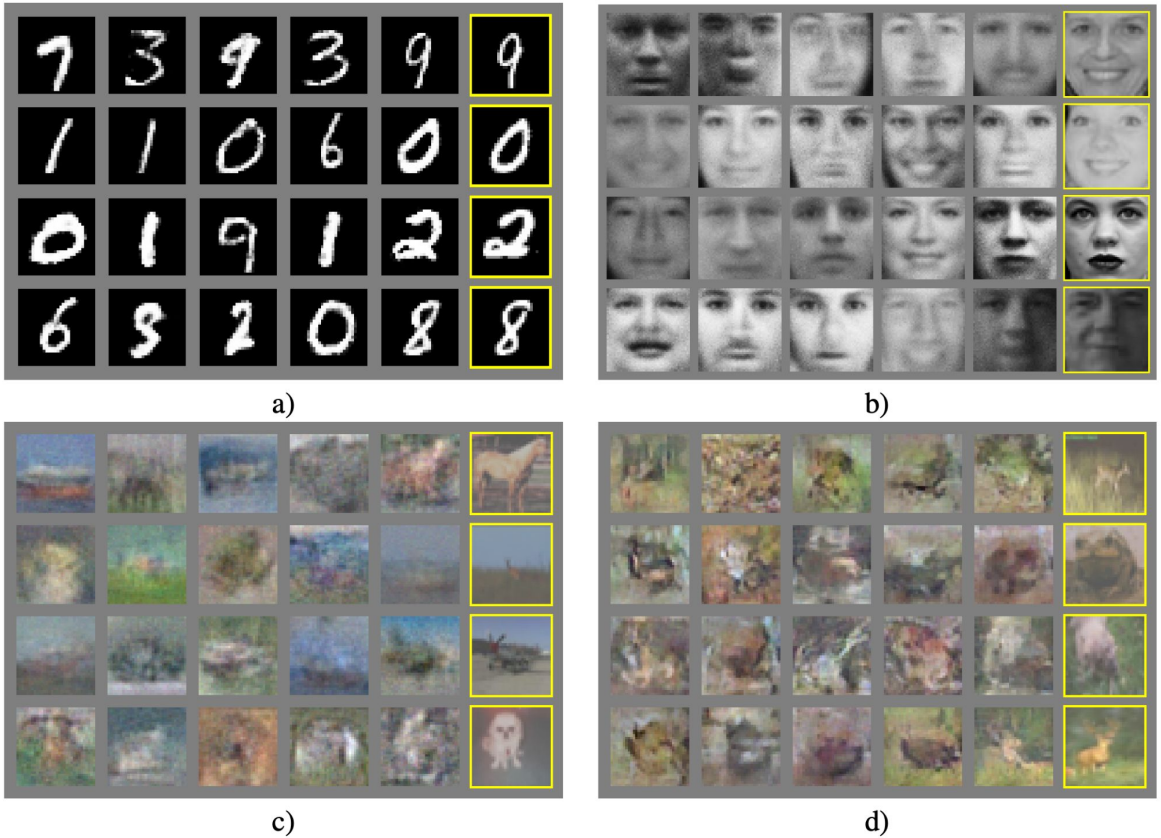
- Generative Adversarial Networks (GANs) are a Deep Learning framework to train a powerful **generation models** without using labeled data.
- The original paper: Generative Adversarial Nets [Goodfellow et al. 2014]



Generative Adversarial Nets

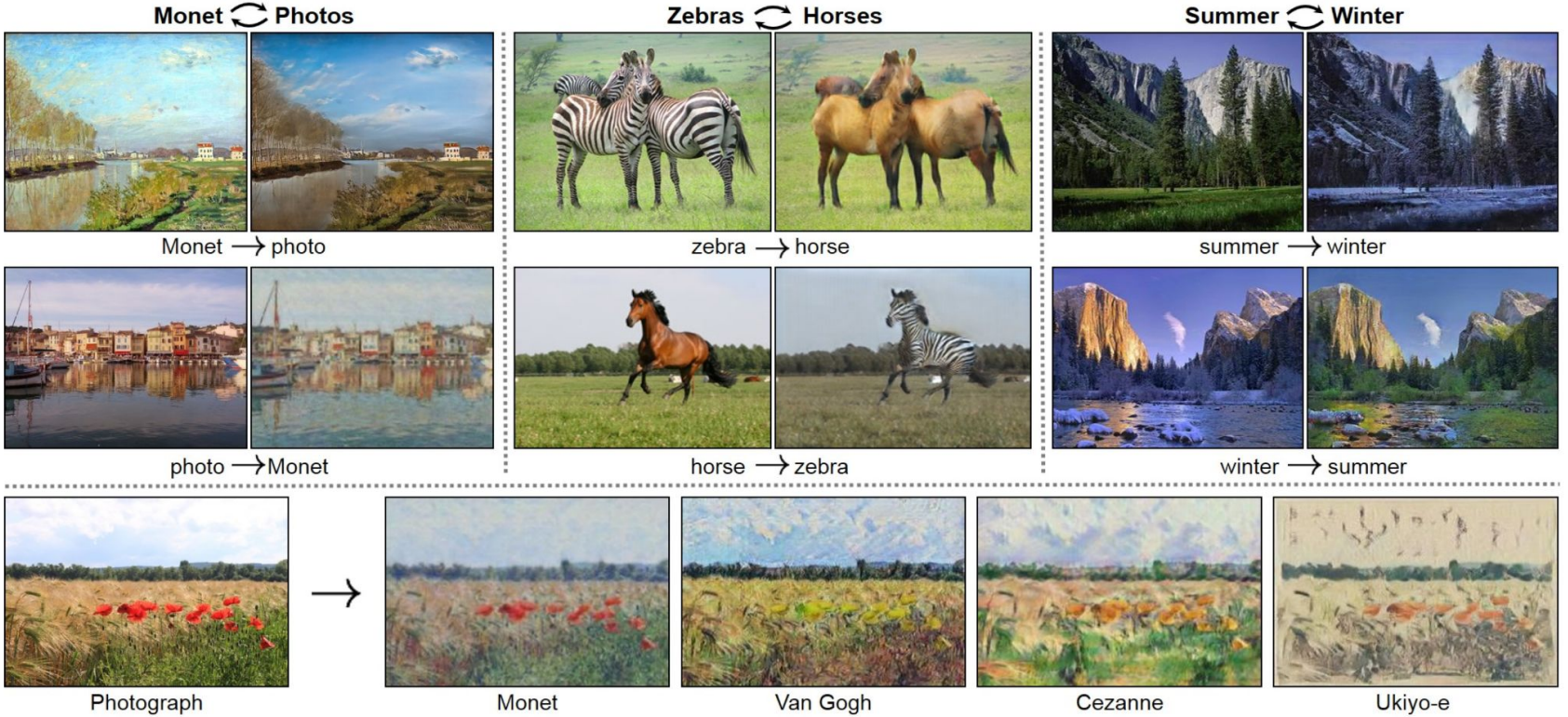
Ian J. Goodfellow, Jean Pouget-Abadie*, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Generated Images by GAN [Goodfellow et al. 2014]



Generated Images by DCGAN [Radford et al. 2016]

Image Translation by CycleGAN [Zhu and Park et. al. 2017]



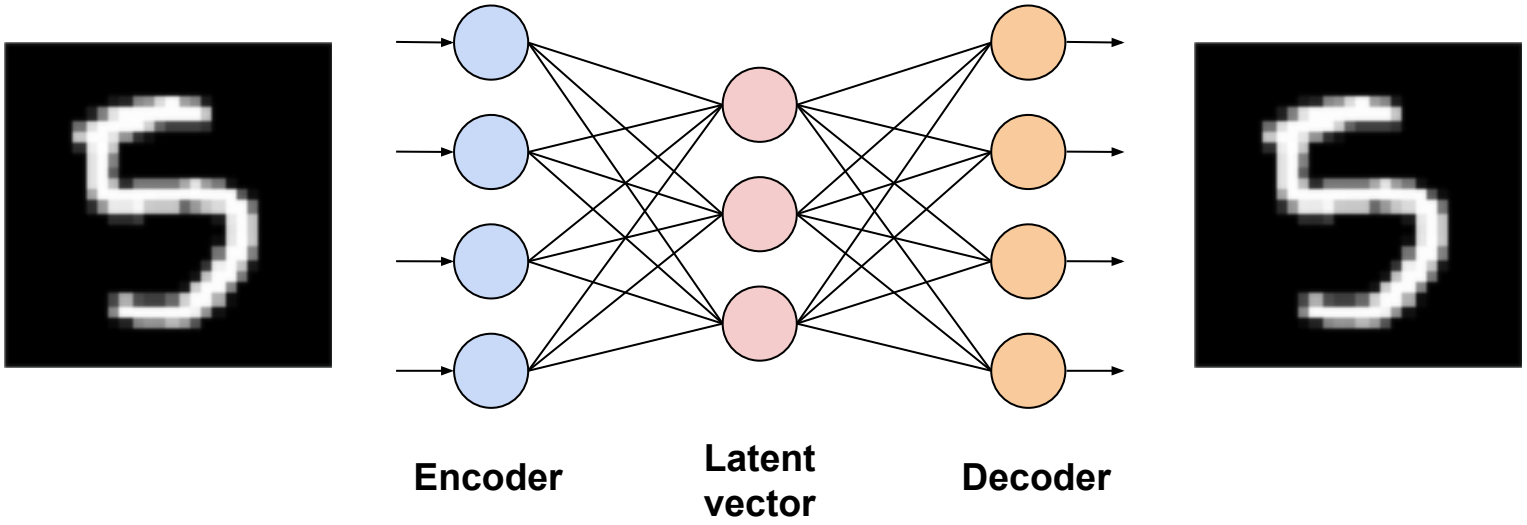
Deep Fake & Ethical issues

- It’s Getting Harder to Spot a Deep Fake Video | Bloomberg Quicktake - YouTube

https://en.wikipedia.org/wiki/Deepfake

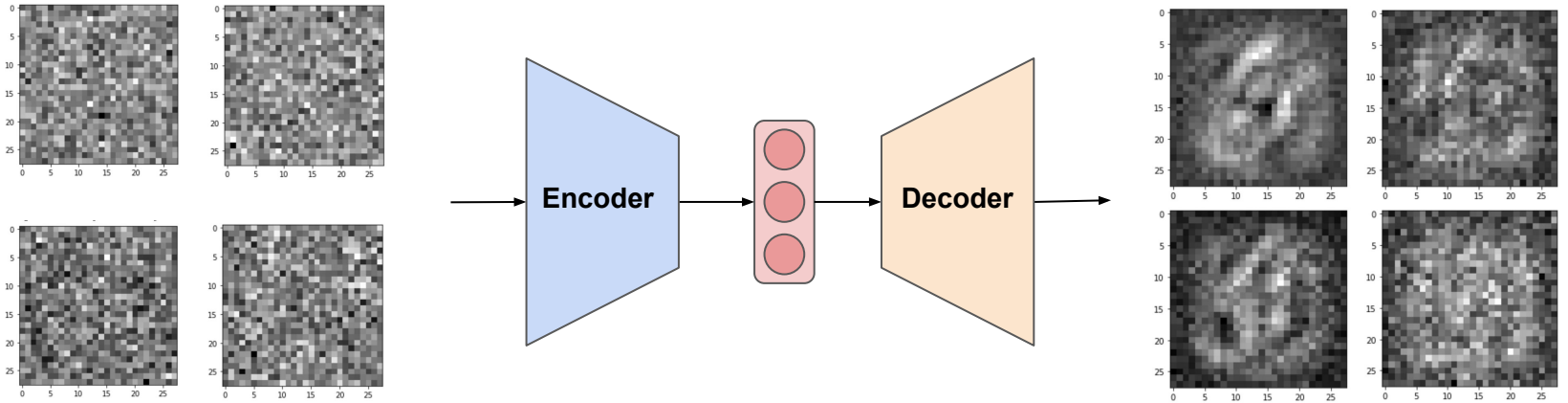
GAN Principles

Wait! You Said We Could Use Autoencoders for Image Generation!



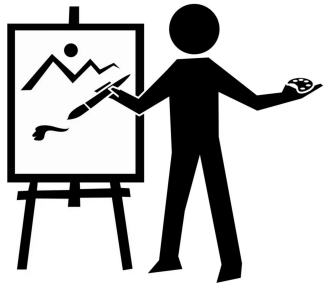
Yes, we could. May not be the best option. Why?

- The model is trained to reconstruct the original image
- No objective function toward generating “real” images



That’s the motivation behind GAN!

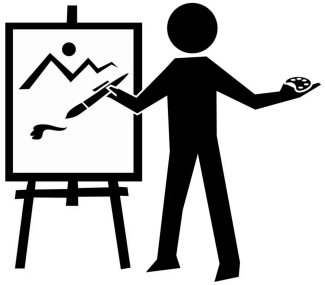
How GANs Work



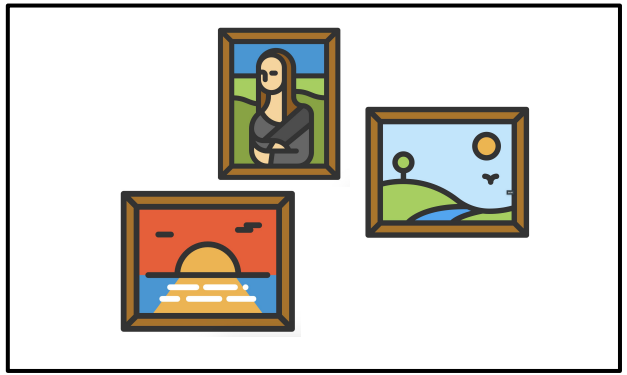
Art forger

Art appraiser

Story: Art forger vs. Art appraiser



Art forger



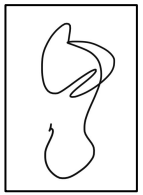
Real Paintings

Art appraiser

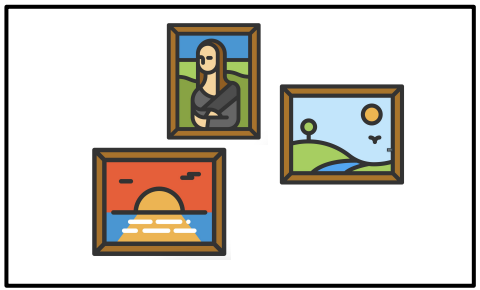
Step 1: Art forger draws random paintings

Art forger

draw

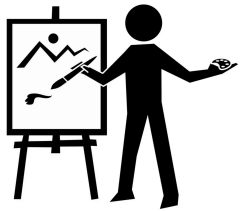


Art appraiser

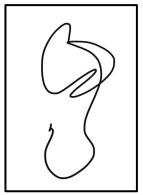


Real paintings

Step 2: Art appraiser is trained to predict True or Fake



Art forger

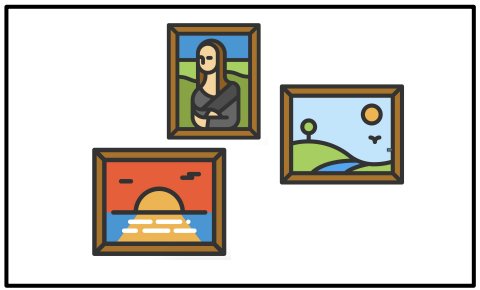


label: Fake

train

Level up! ↑

Art appraiser

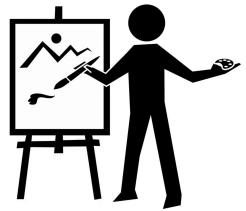


Real paintings

sample

label: Real

Step 3: Art appraiser predicts if the painting is fake
Art forger is informed of the prediction result



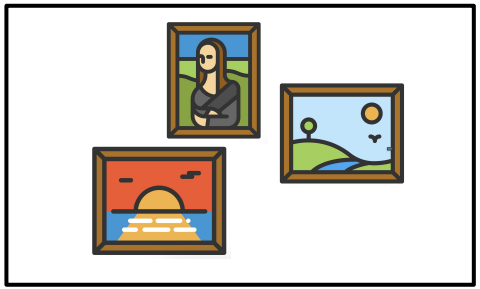
Art forger

Inform

Prediction: Fake

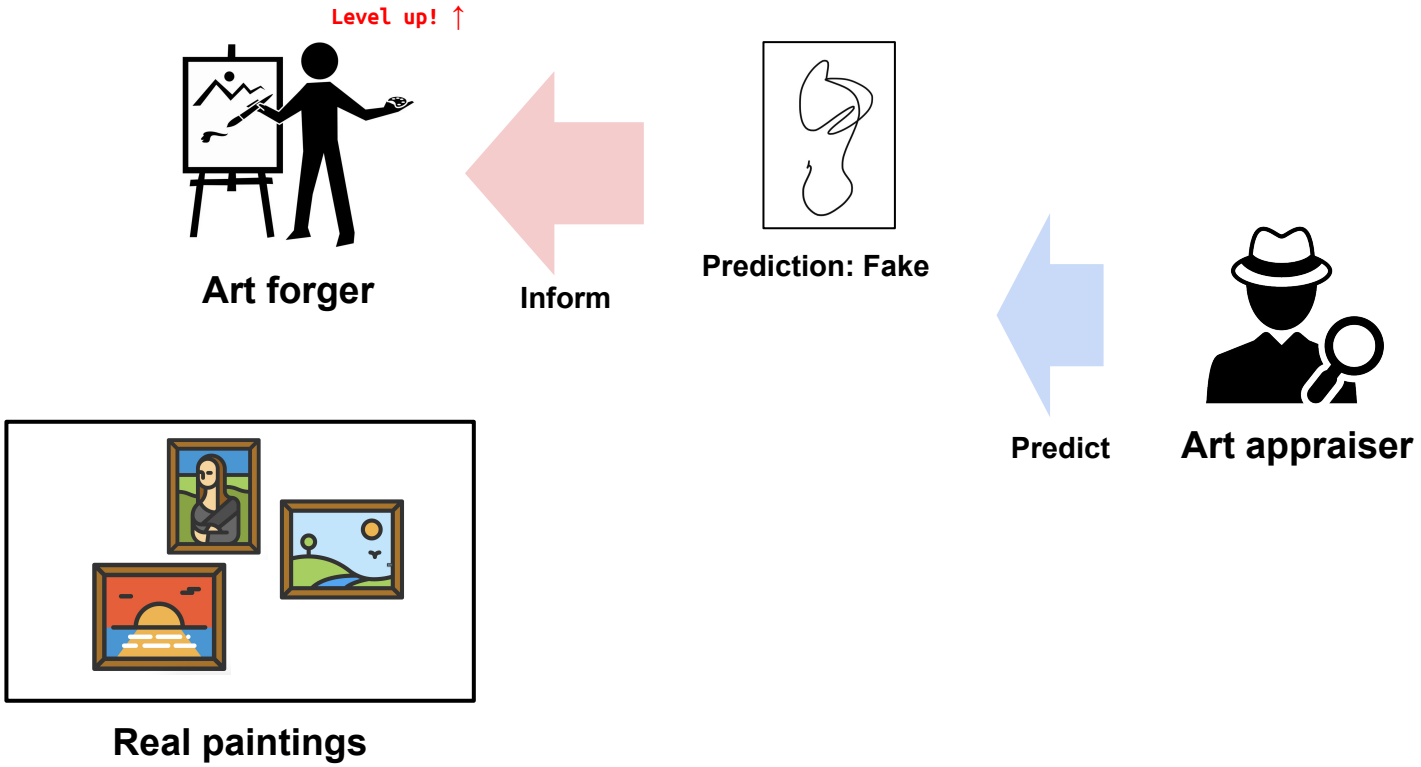
Predict

Art appraiser

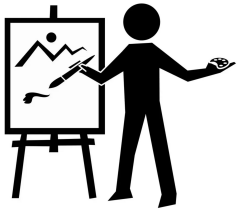


Real paintings

Step 4: Art forger is trained to be better at drawing

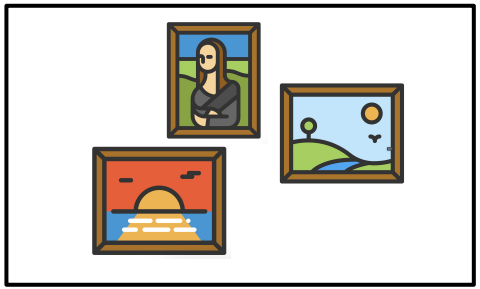


Step 5: Repeat Steps 1-4



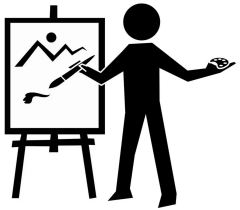
Art forger

Art appraiser



Real paintings

Step 1

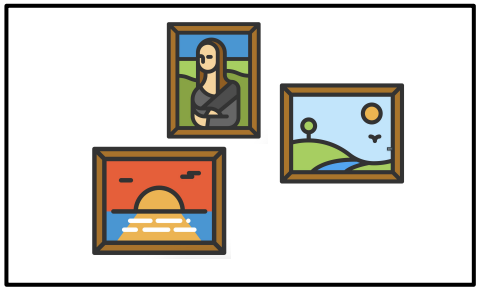


Art forger

draw

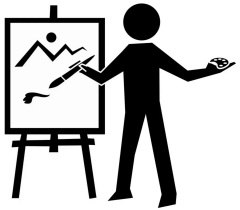
label: Fake

Art appraiser



Real paintings

Step 2



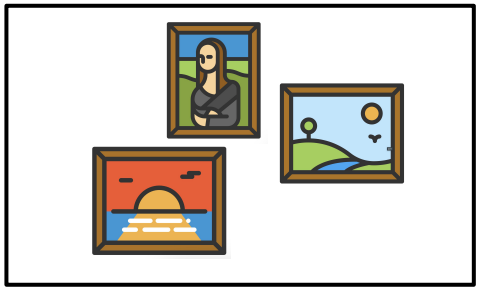
Art forger

draw

label: Fake

train

Art appraiser

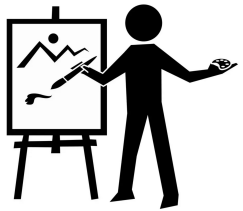


Real paintings

sample

label: Real

Step 3



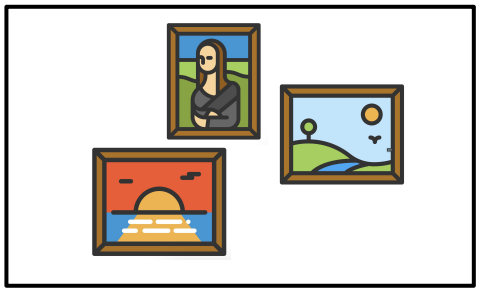
Art forger

Inform

Predict: Fake

Predict

Art appraiser



Real paintings

Step 4

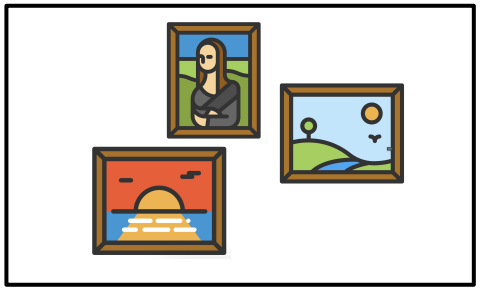
Art forger

Inform

Predict: Fake

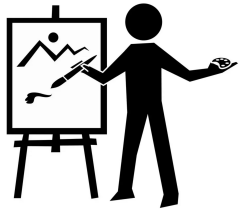
Predict

Art appraiser



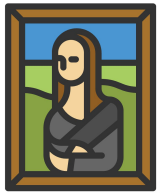
Real paintings

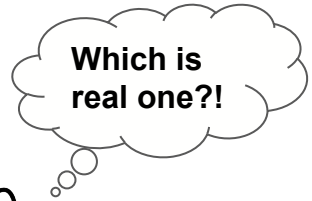
Eventually, Art forger would become really good at drawing fake paintings!



Art forger

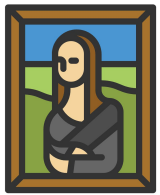
draw

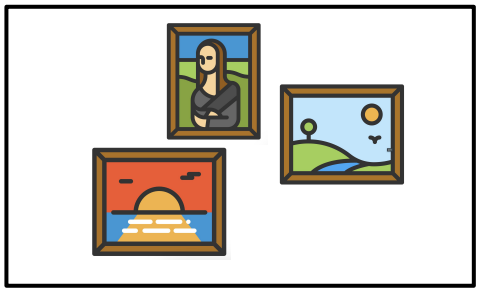




Art appraiser

Predict

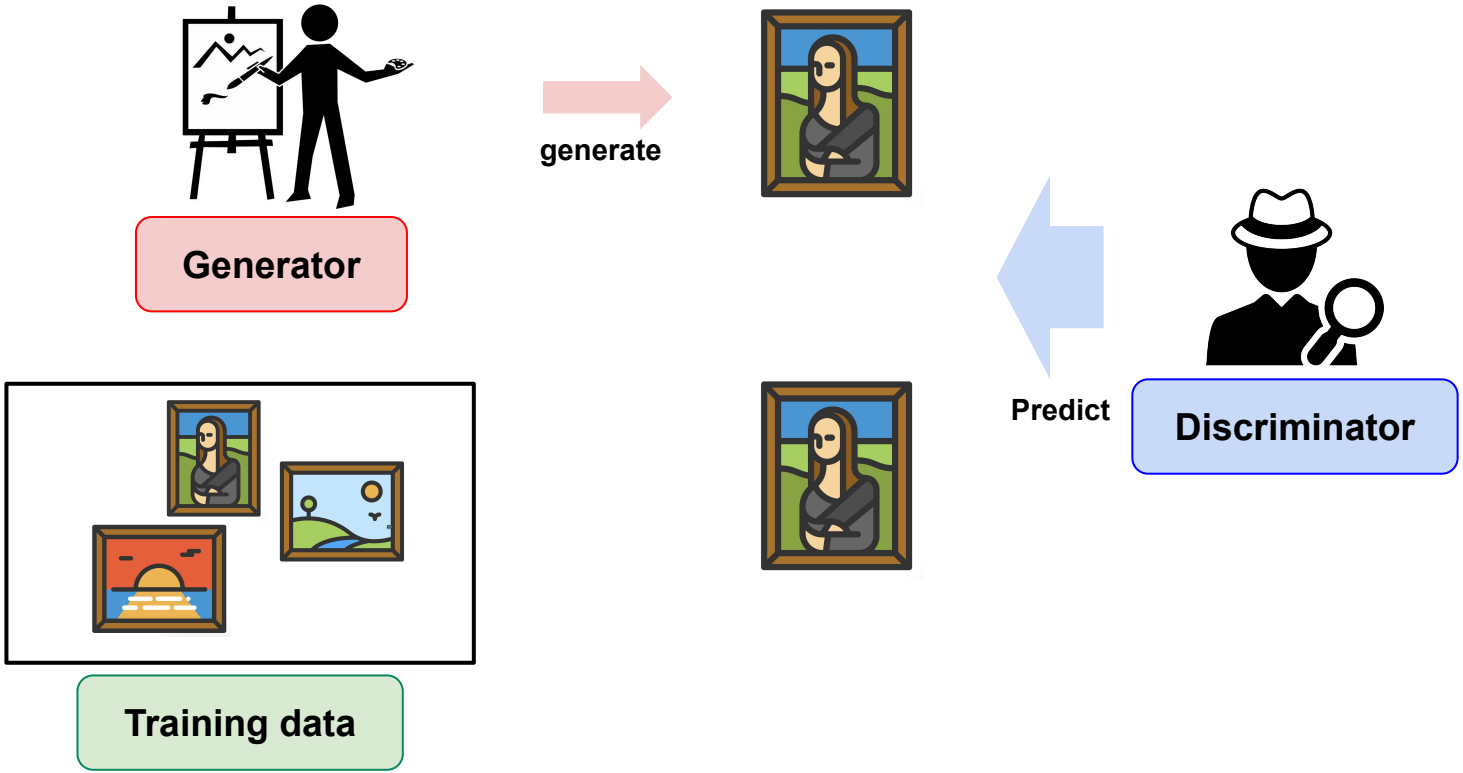




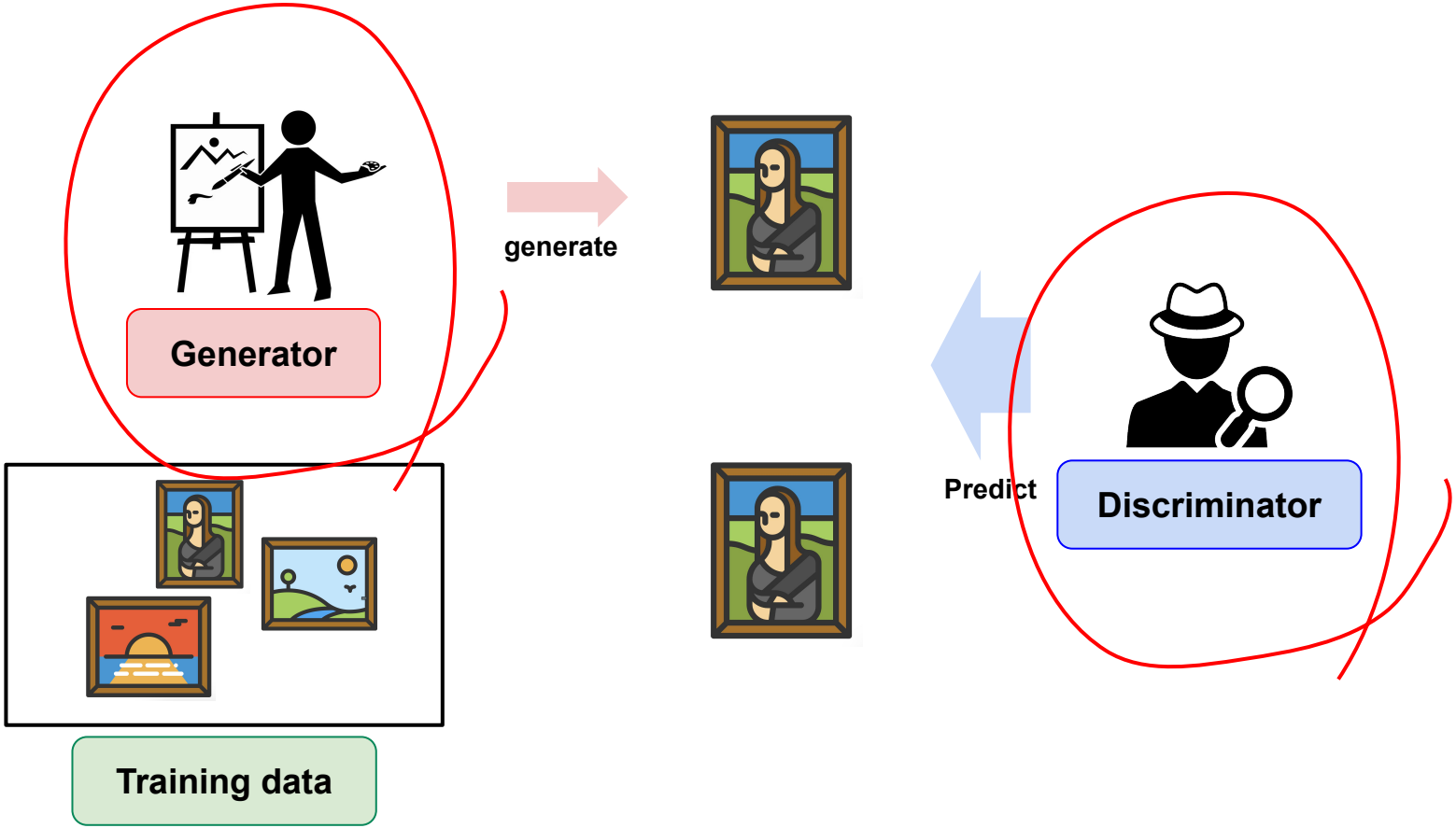
Real paintings

Generative Adversarial Network (GAN)

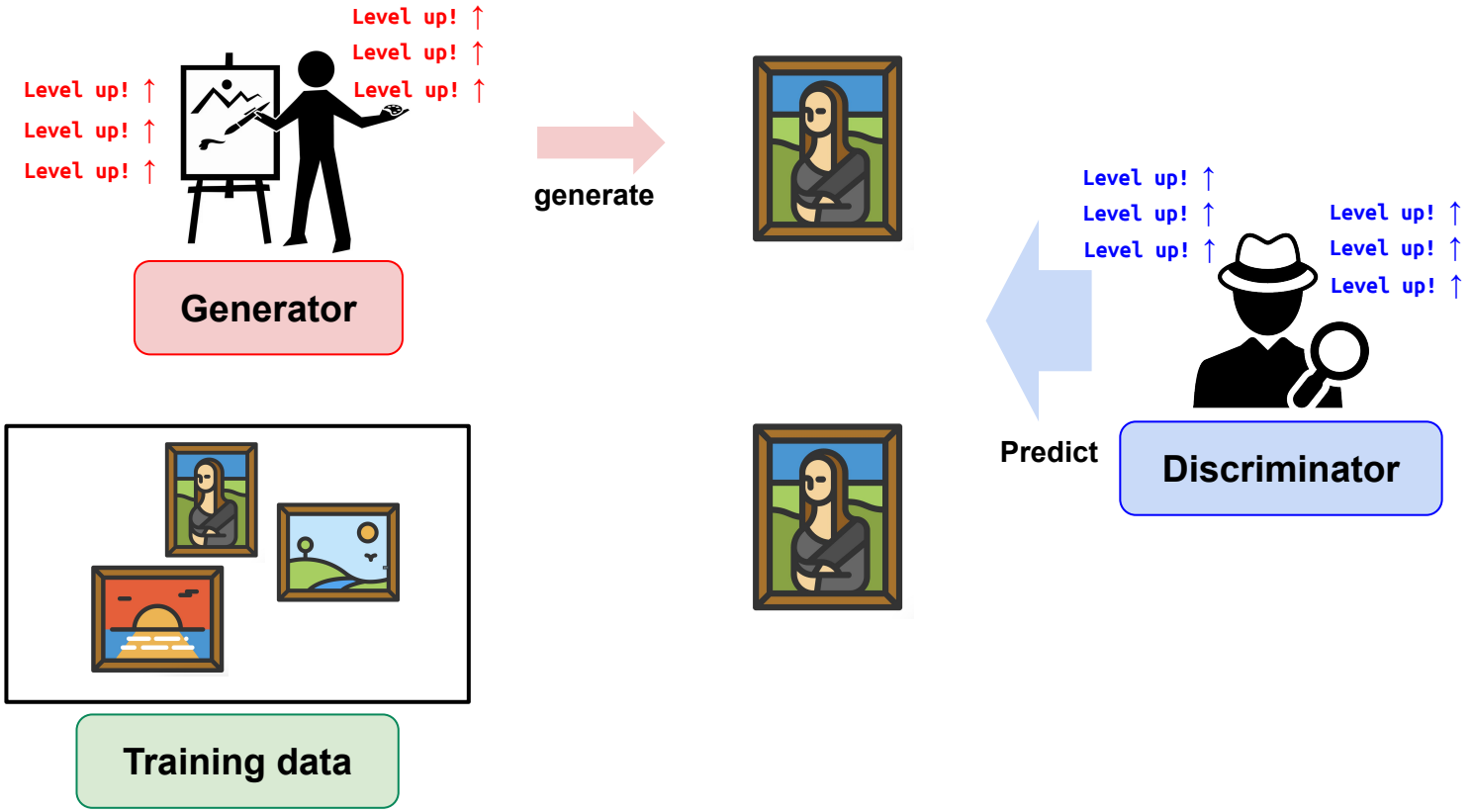
Generator + Discriminator + Training data



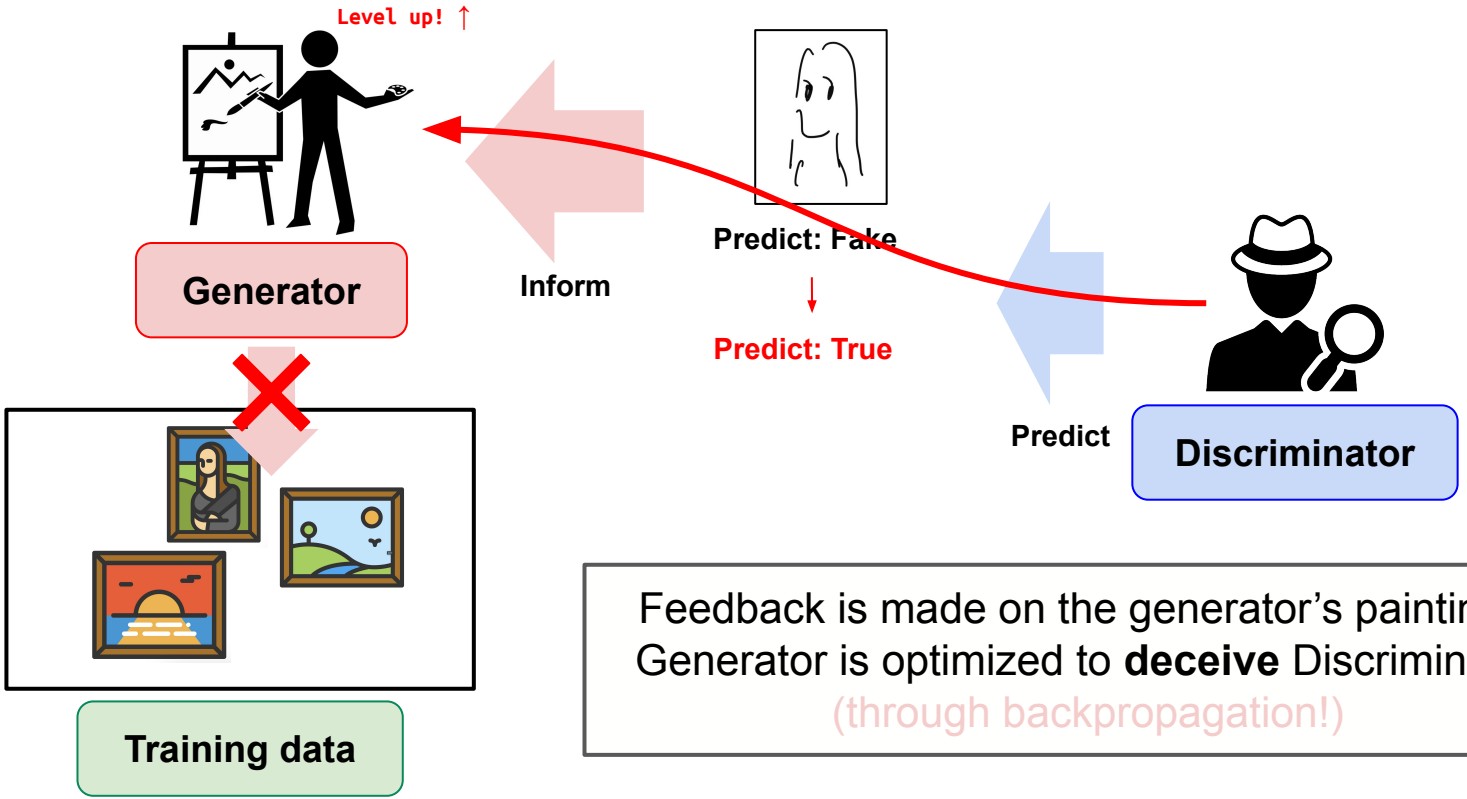
Generator and Discriminator are Neural Networks!



Key point 1: Both models are trained to be better at generation and detection

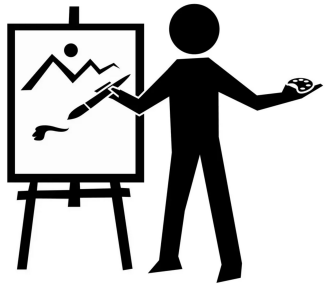


Key point 2: Generator does NOT directly look at real paintings



Feedback is made on the generator's paintings!
Generator is optimized to **deceive** Discriminator
(through backpropagation!)

End of Story



Art forger

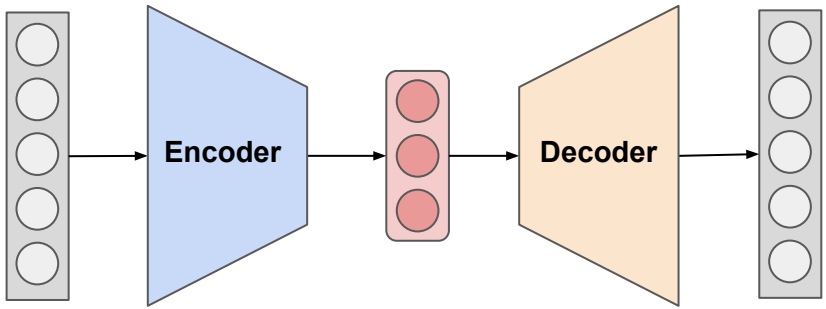
vs.

Art appraiser

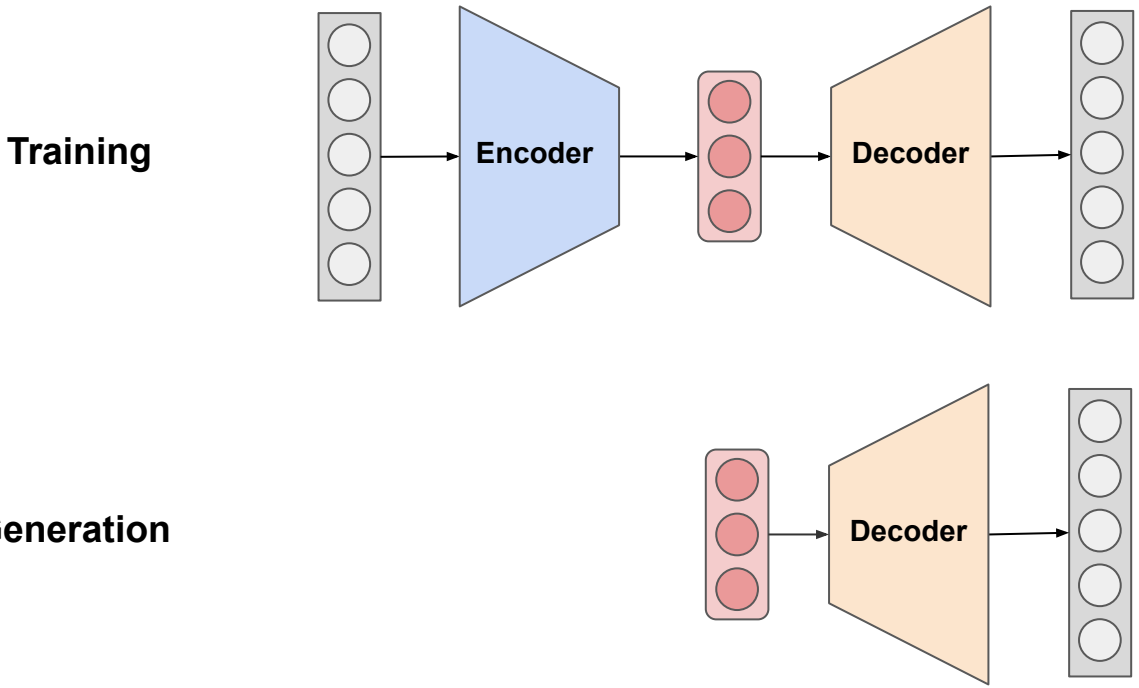
Questions?

Question: How Can We Use Autoencoder for Image Generation?

- Encoder: Input values → Latent vectors
- Decoder: Latent vectors → Output values

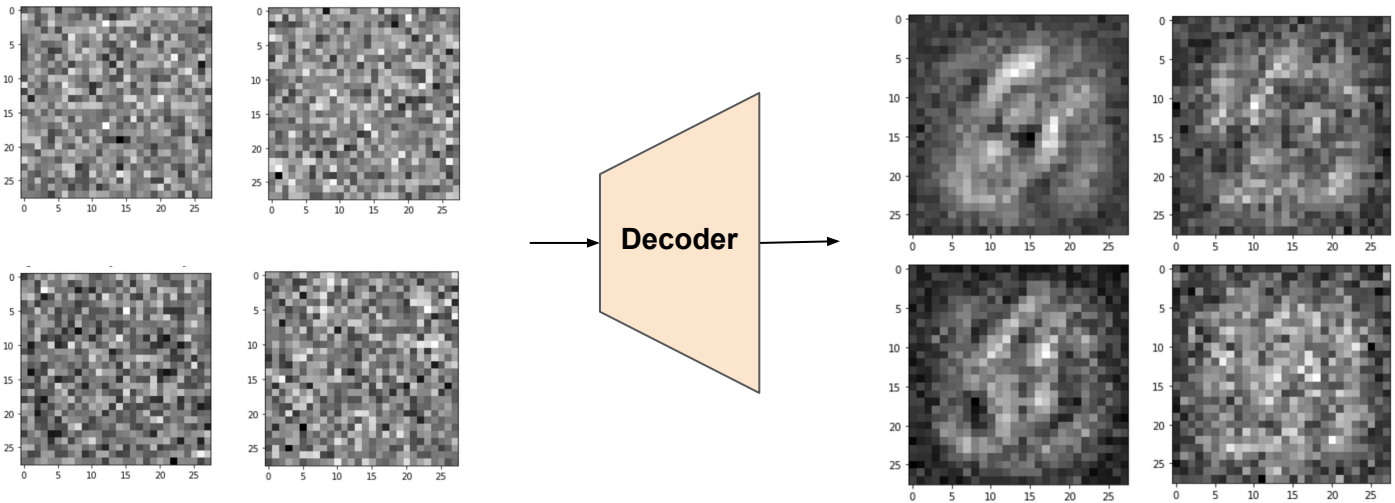


Generating Images Using Trained Autoencoder

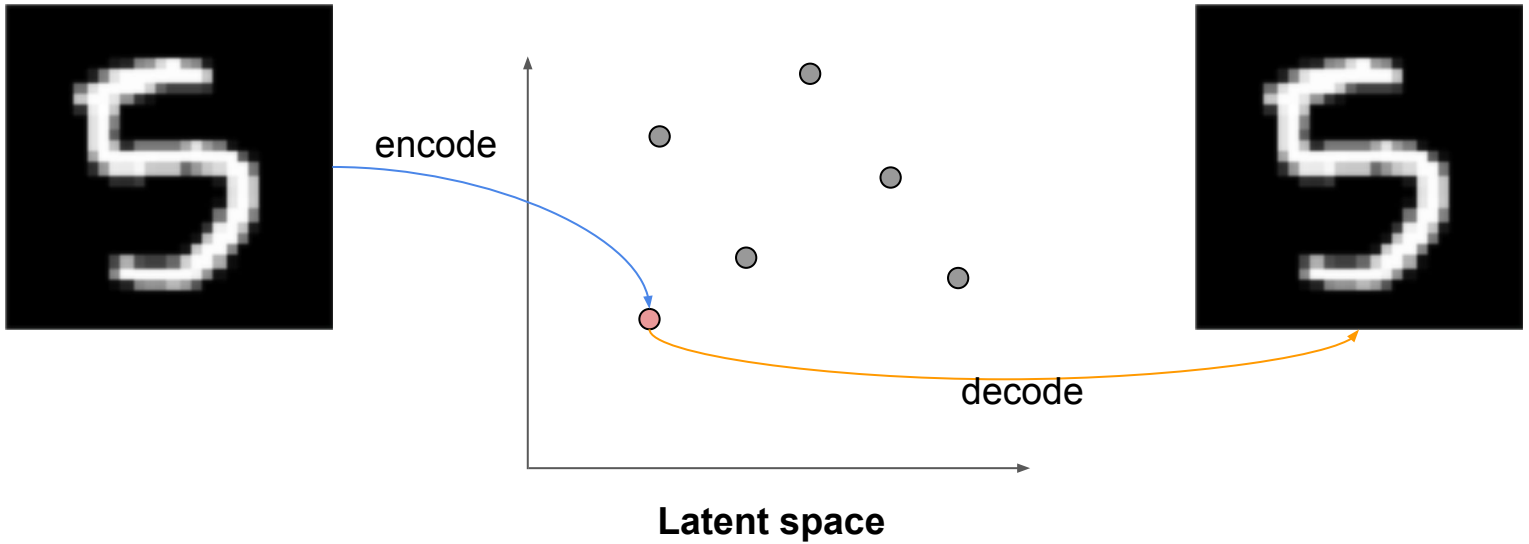


GAN's Generator

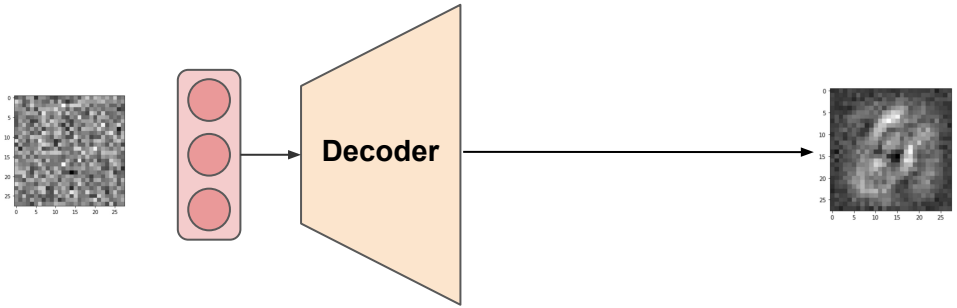
- Input: Random noise (i.e., randomly sampled latent vector)
- Output: Image (e.g., 28x28)



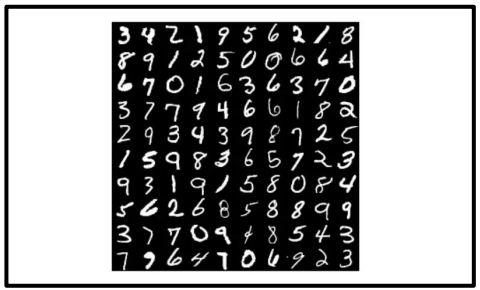
Why Random Noise? What Does It Mean? “Random Sampling” in Latent Space



GAN Architecture



Generator

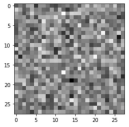


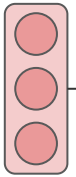
Training data

How does it look like?

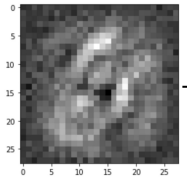
Discriminator

GAN Architecture





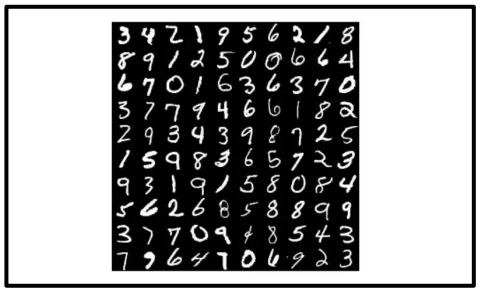
Decoder



Generator

Encoder

Real/Fake



Discriminator

Training data

Note: Why Do We Call It Encoder?

- f: Input representations → Latent representation
- Deep Learning classifier = Encoder + Output layer

Training GAN models (The original paper version)

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

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

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Training GAN models (1/2)

- Iteratively updating Discriminator and Generator using **different optimizers**

```
Input: model_G, model_D, optimizer_G, optimizer_D, data, NumEpoch, BatchSize, ...

For i = 1 to NumEpoch
    batches ← CreateMiniBatch(data, BatchSize)
    For j = 1 to len(Batches)
        # Train Discriminator
        batch ← batches[j]                      # All real examples
        out_D1 ← model_D(batch)
        loss_D1 ← loss_fn(out_D1, "all_real")
        loss_D1.backpropagate()
        optimizer_D.step()
        fake ← model_G(random_noise)            # All fake examples
        out_D2 ← model_D(fake)
        loss_D2 ← loss_fn(out_D2, "all_fake")
        loss_D2.backpropagate()
        optimizer_D.step()
```

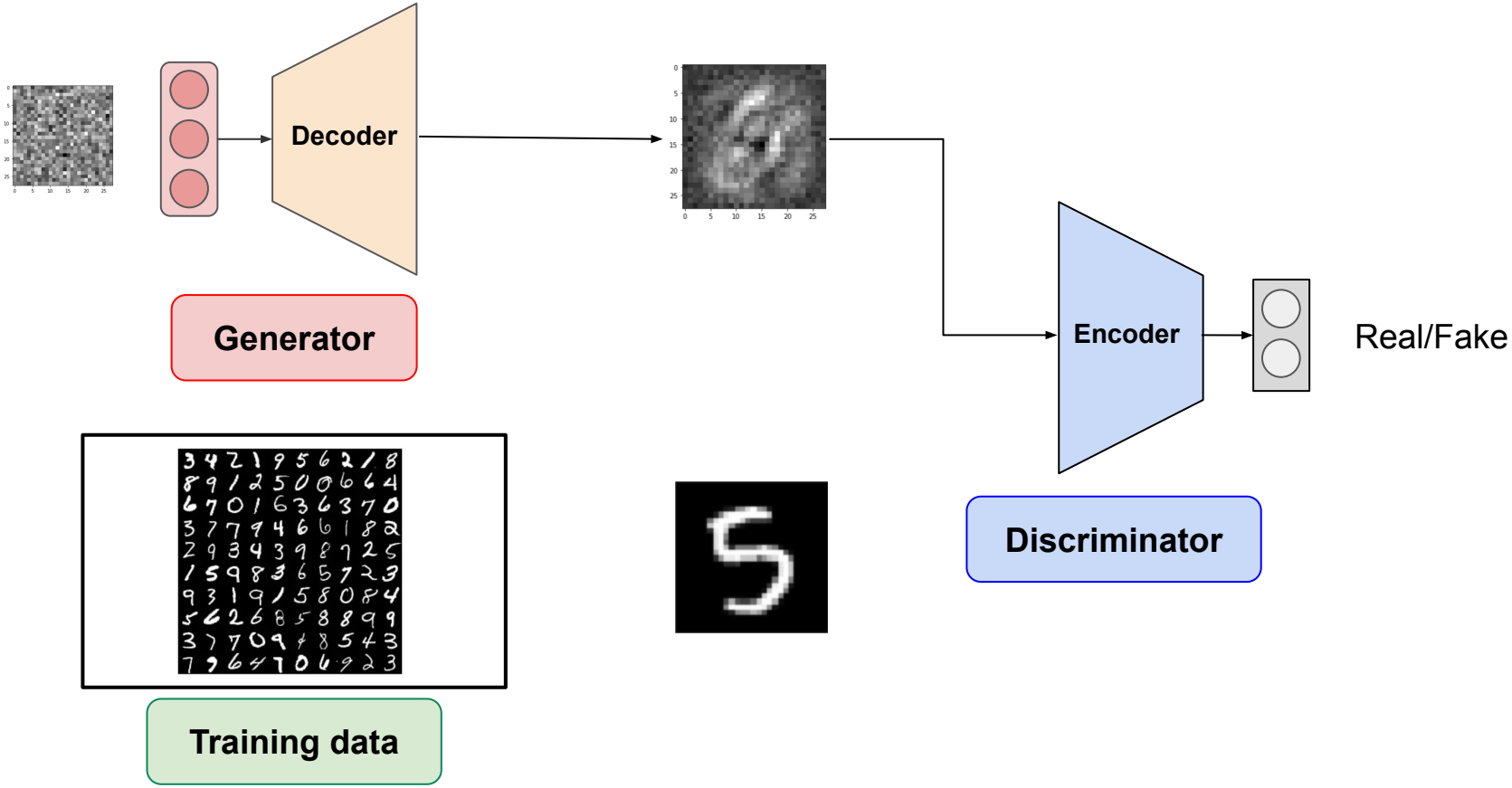
Training GAN models (2/2)

- Iteratively updating Discriminator and Generator using **different optimizers**

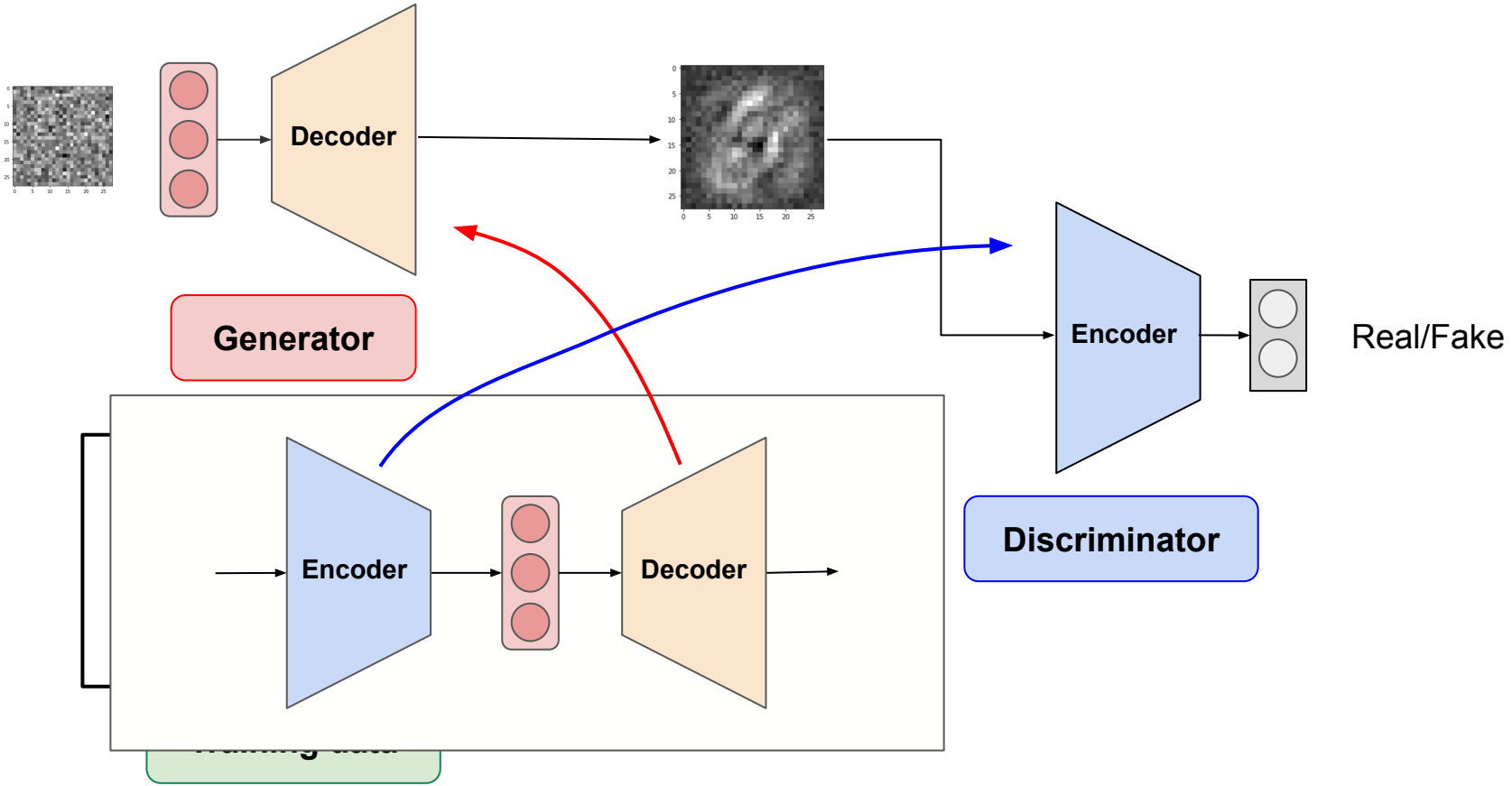
```
For i = 1 to NumEpoch
    batches ← CreateMiniBatch(data, BatchSize)
    For j = 1 to len(Batches)
        # Train Discriminator
        ...
        fake ← model_G(random_noise)
        ...

        # Train Generator
        freeze(model_D)                    # Not to update model_D
        out_G ← model_D(fake)
        loss_G ← loss_fn(out, "all_real")   # Update model_G to deceive model_D
        loss_G.backpropagate()
        optimizer_G.step()
        unfreeze(model_D)
```

Questions?

Let’s Design your first GAN model

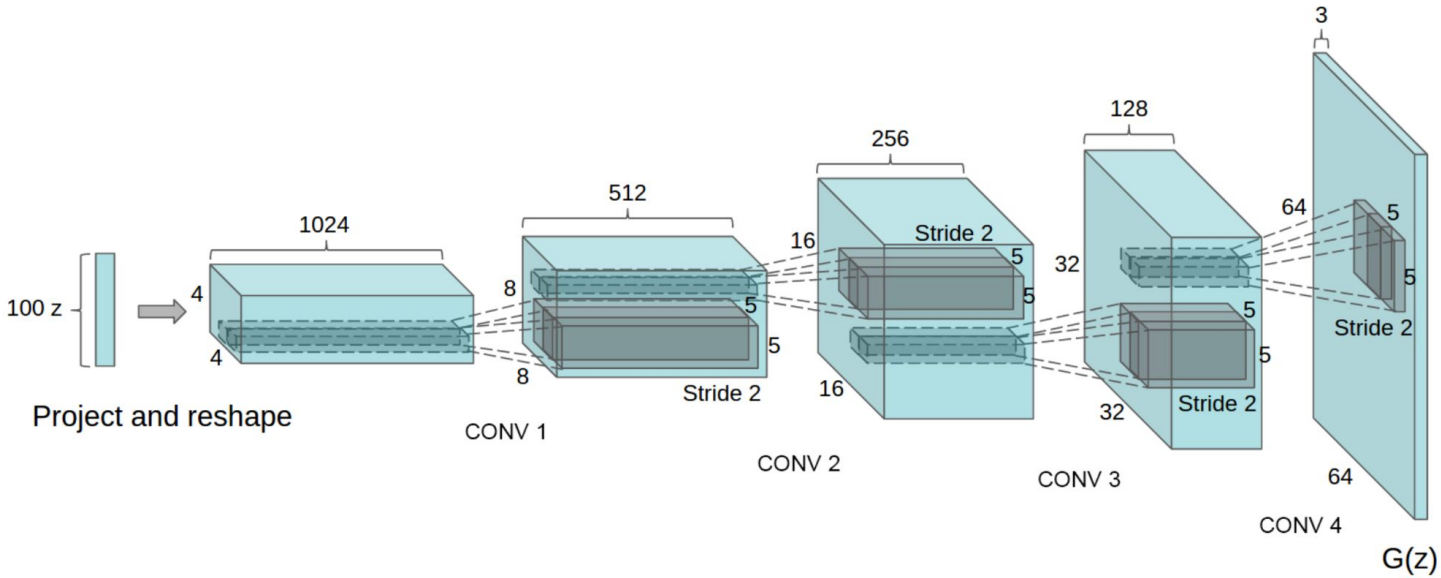


Any Autoencoder Code Can Be Reused!



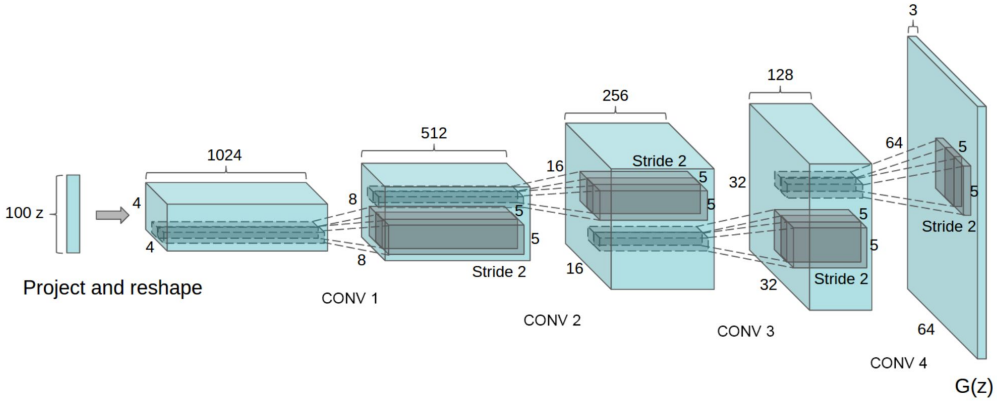
Deep Convolutional GAN (DCGAN) [Radford et al. 2016]

- The use of (Transposed) Convolutional layers
  - In the analogy of Autoencoder vs. Convolutional Autoencoder



DCGAN Blueprint

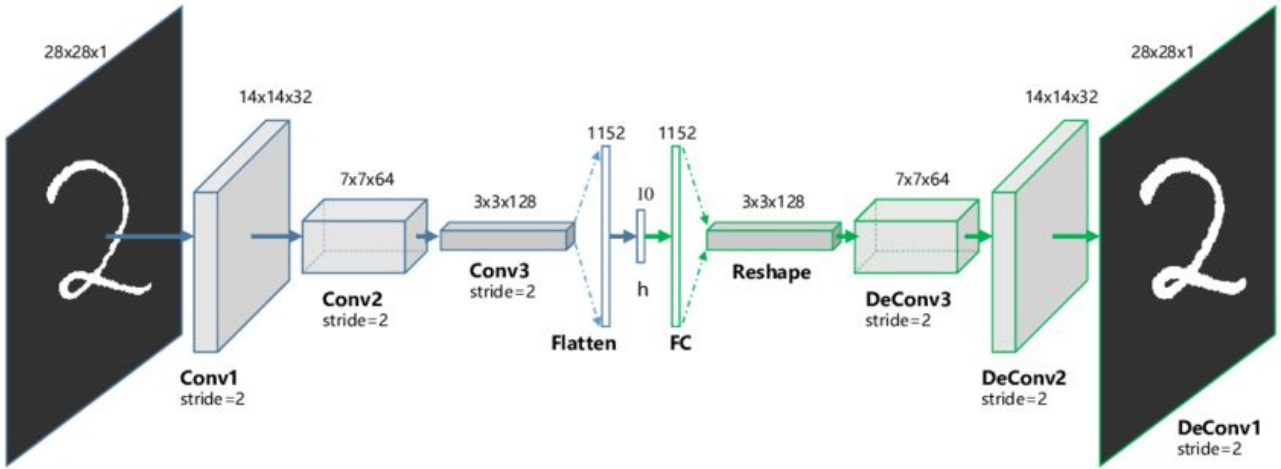
(*) fractional-strided convolution
= transposed convolution



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.

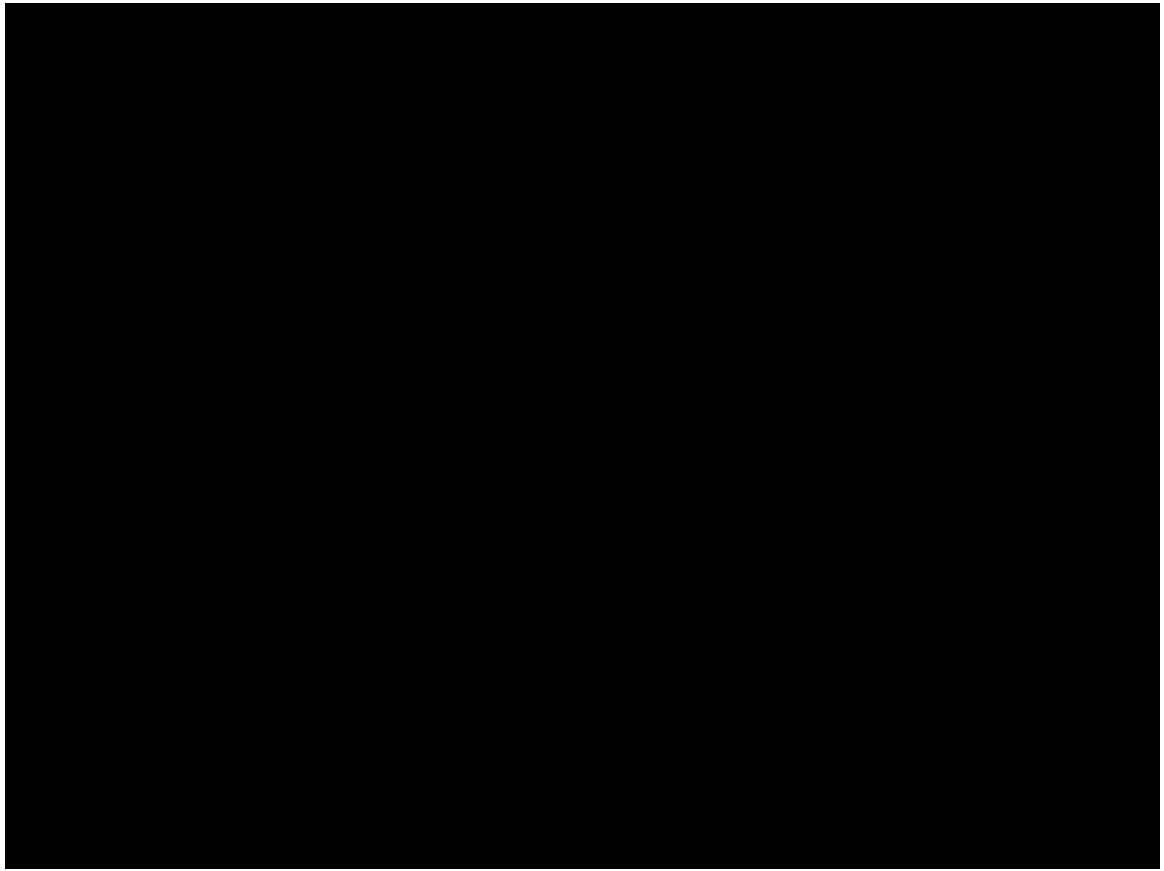
Okay. We have Convolutional Autoencoder Code



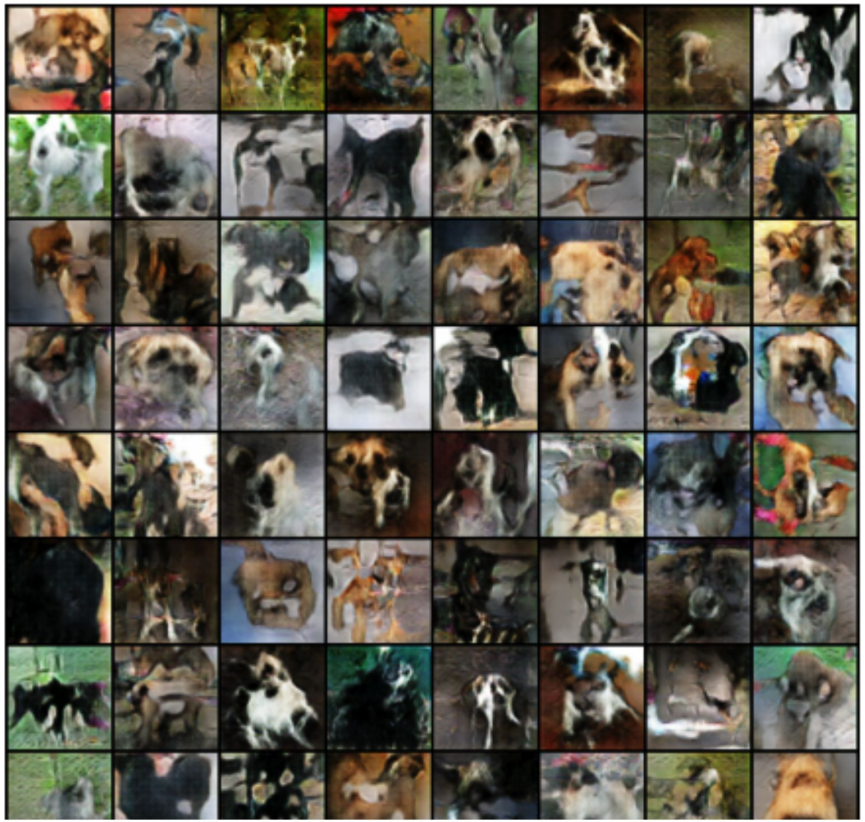
Hands-on Time: DCGAN for Cats-vs-Dogs

- Note: please make sure you have created a Kaggle account and accept the terms and conditions for the Dogs vs. Cats dataset (Link)
- Google Colab

DCGAN Dog Image Generation (20 epochs)

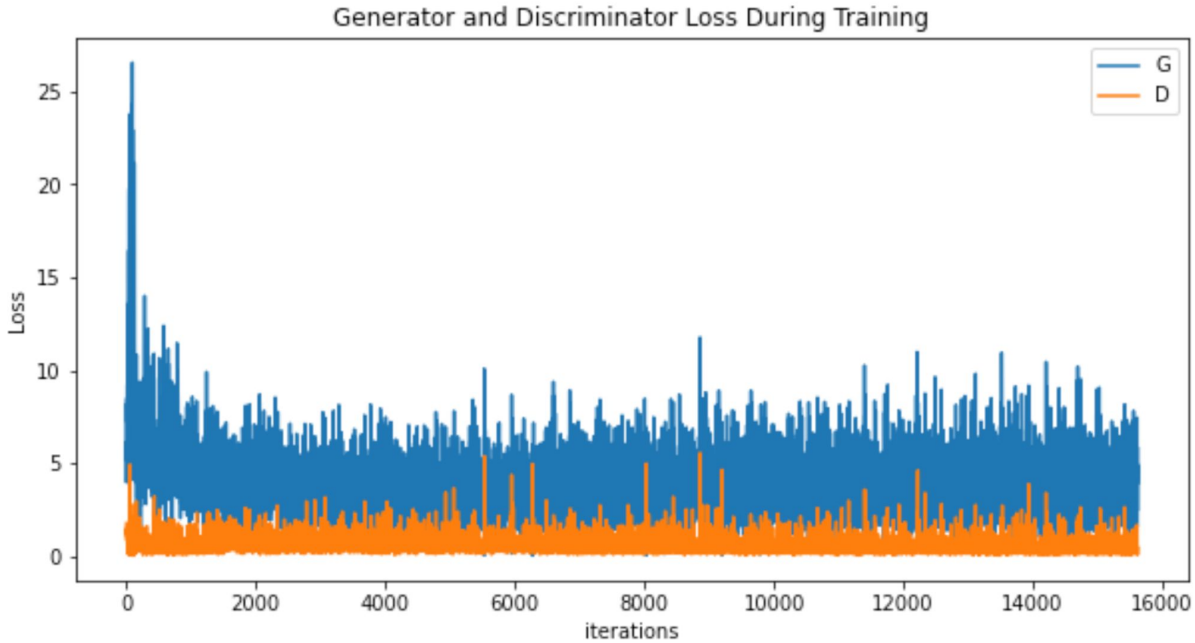


Final Results

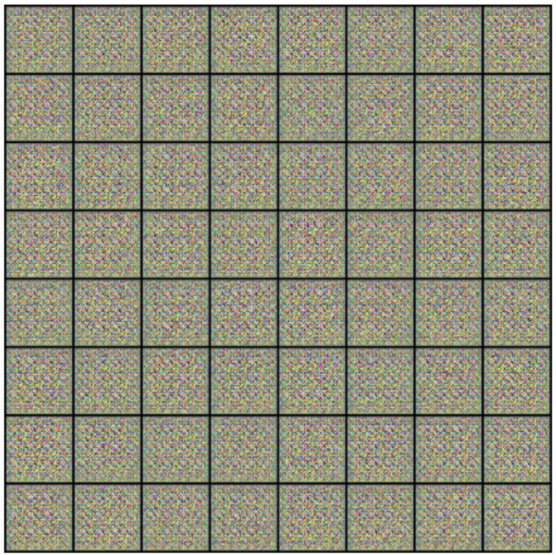


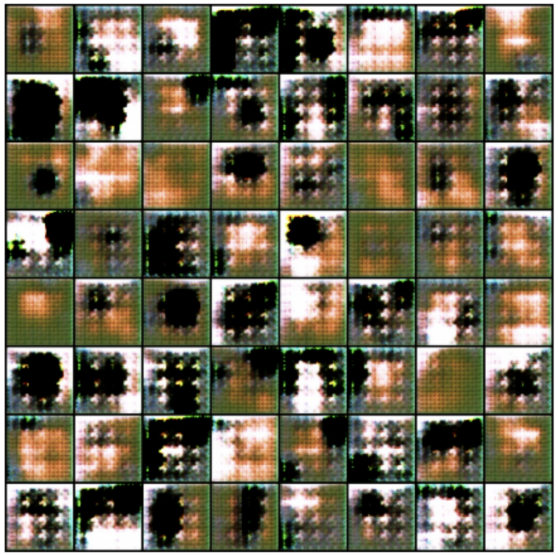
Training loss of Generator/Discriminator

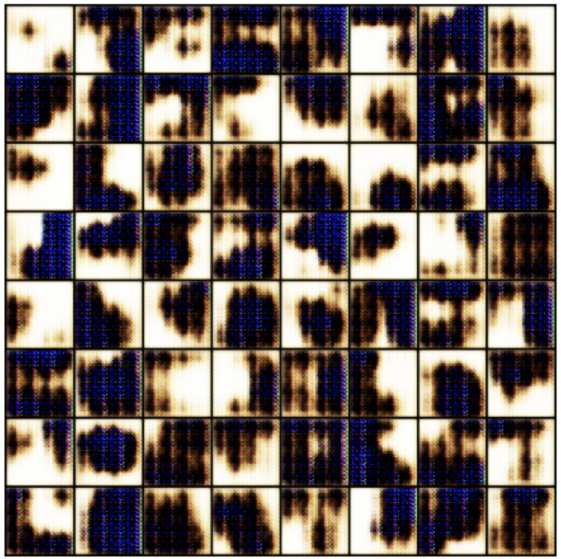
- No clear descending patterns



GANs are sensitive to the optimization configuration







Tips: Loading Custom Image Dataset

- images
  - class_name1
    - 1_1.jpg
    - 1_2.jpg
    - ...
  - class_name2
    - 2_1.jpg
    - 2_2.jpg
    - ...

```
image_size = 64 # 128, 224
dataset = torchvision.datasets.ImageFolder(
    root="images", # it considers subdirectory as the class name
    transform=transforms.Compose(
        [transforms.Resize((image_size, image_size)),
         transforms.CenterCrop(image_size),
         transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]))
```

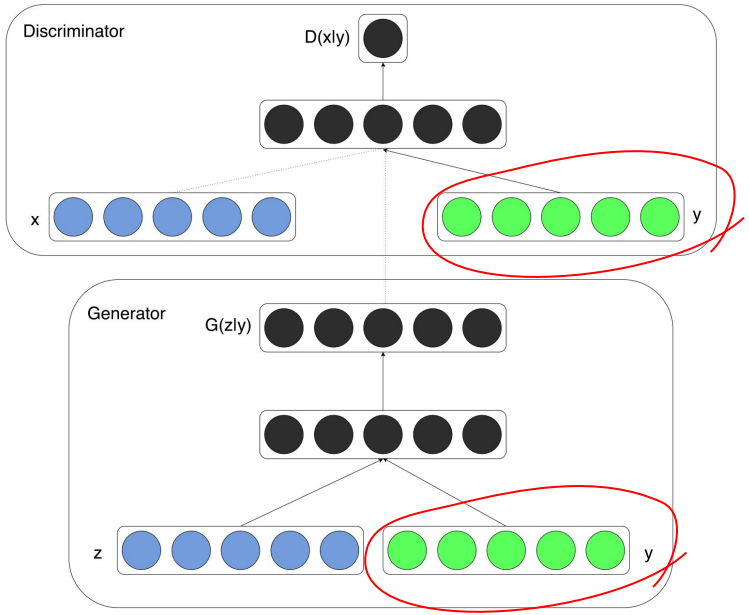
Advanced GANs

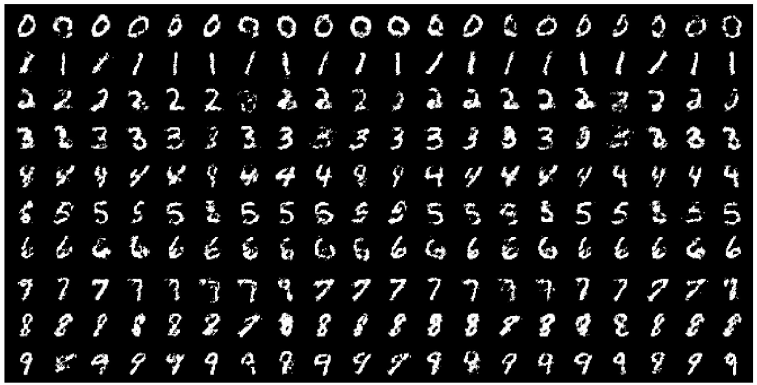
Advanced GANs

- ✅ DCGAN
- Conditional GAN
- Pix2Pix
- CycleGAN

Conditional GAN (CGAN) [Mirza and Osindero 2014]

- We cannot directly specify what types of images to generate → CGAN

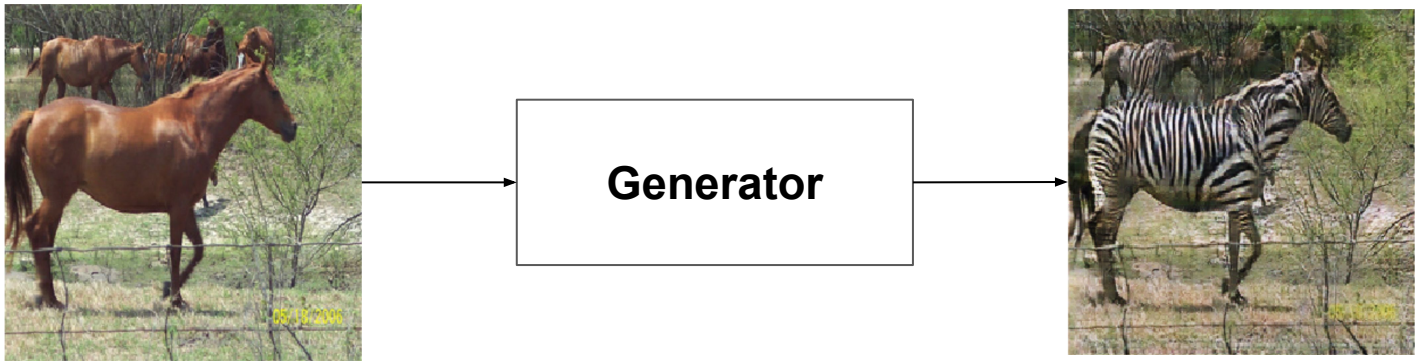




Conditional generations

Attention: Generator != Decoder for Style Transfer GANs

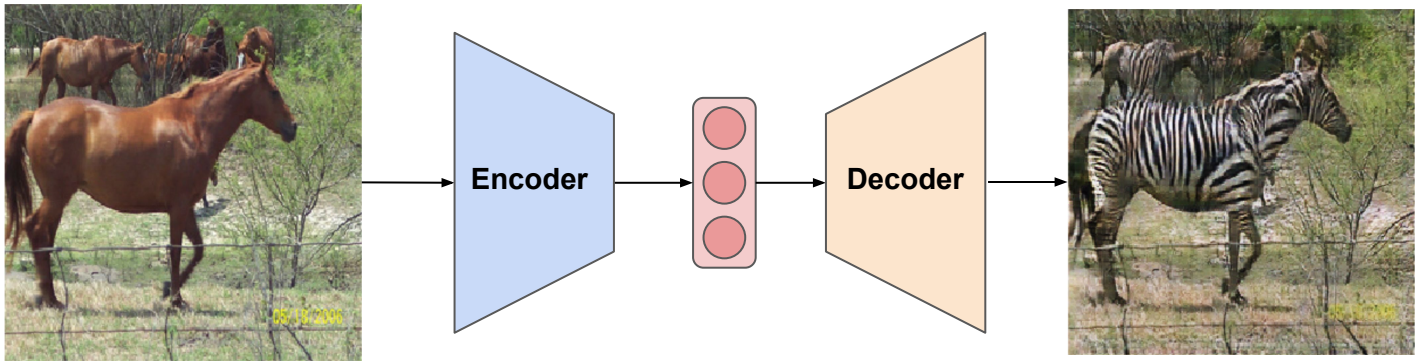
- Random noise is sufficient for “random” image generation
- Style Transfer GANs’ Generators **translate images into different images**



How should such Generators look like?

Attention: Generator != Decoder for Style Transfer GANs

- Random noise is sufficient for “random” image generation
- Style Transfer GANs’ Generators **translate images into different images**

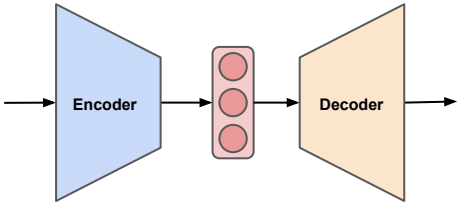


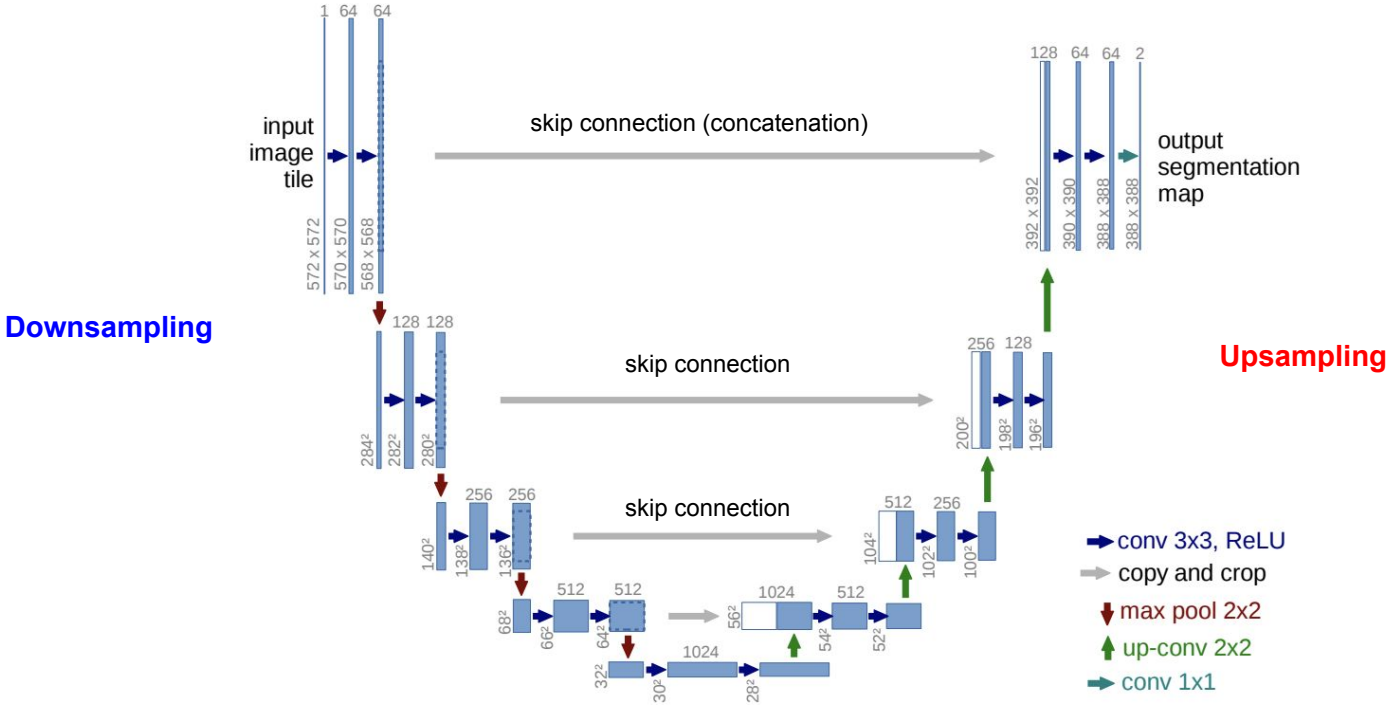
Encoder-Decoder models!

Generator

U-Net

Convolutional Autoencoder + Skip connection

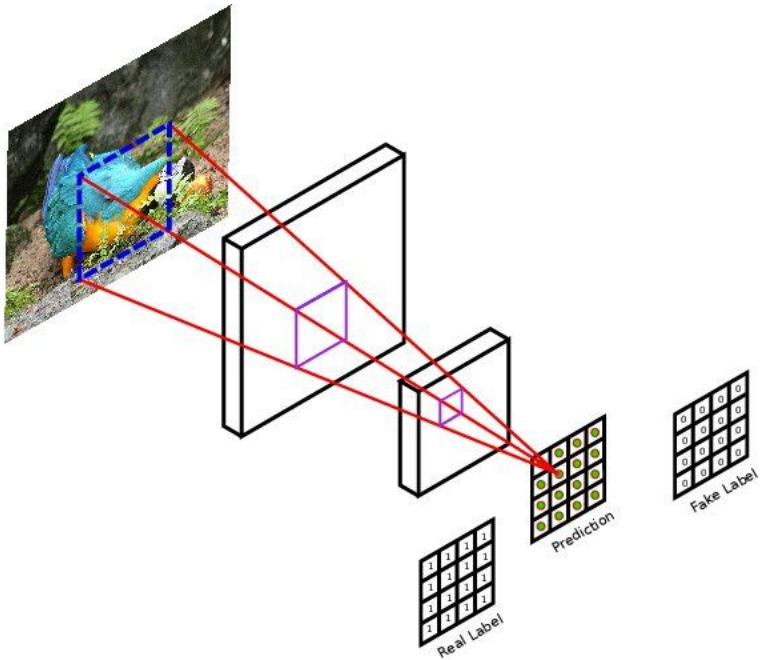


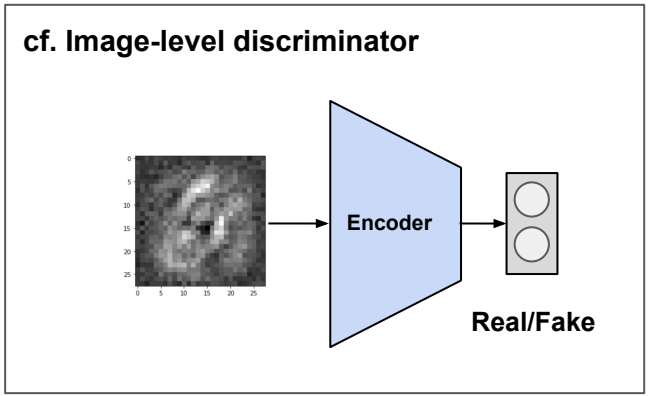


https://arxiv.org/abs/1505.04597

PatchGAN (Discriminator, not a GAN model!)

- Modeling **pixel-level discriminator** by extracting **different patches** from an image





https://arxiv.org/abs/1803.07422

Pix2Pix [Isola et al. 2017]

- U-Net (Generator) + PatchGAN (Discriminator) + Paired Images

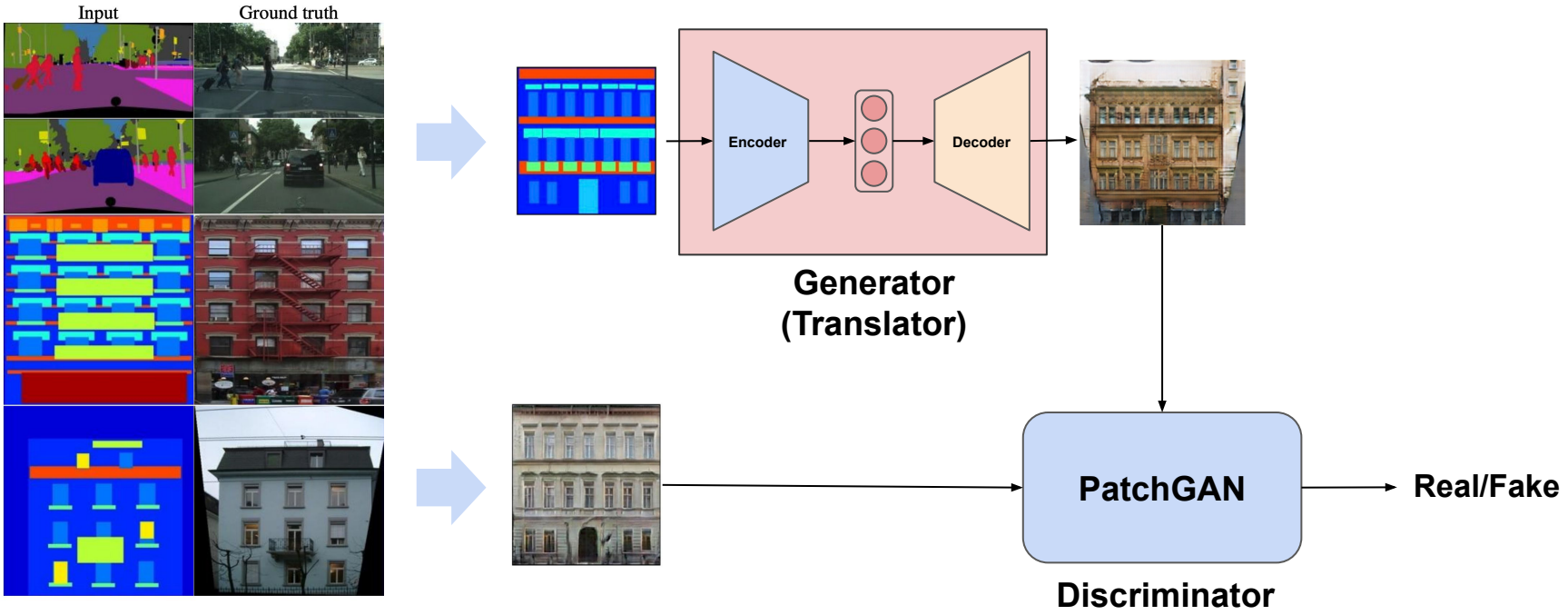


Image Translation from Unpaired Images

Horse

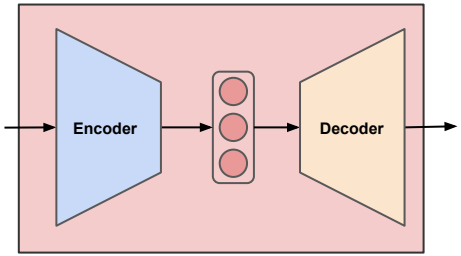


Zebra

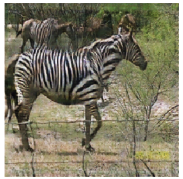


CycleGAN

Real image in domain A

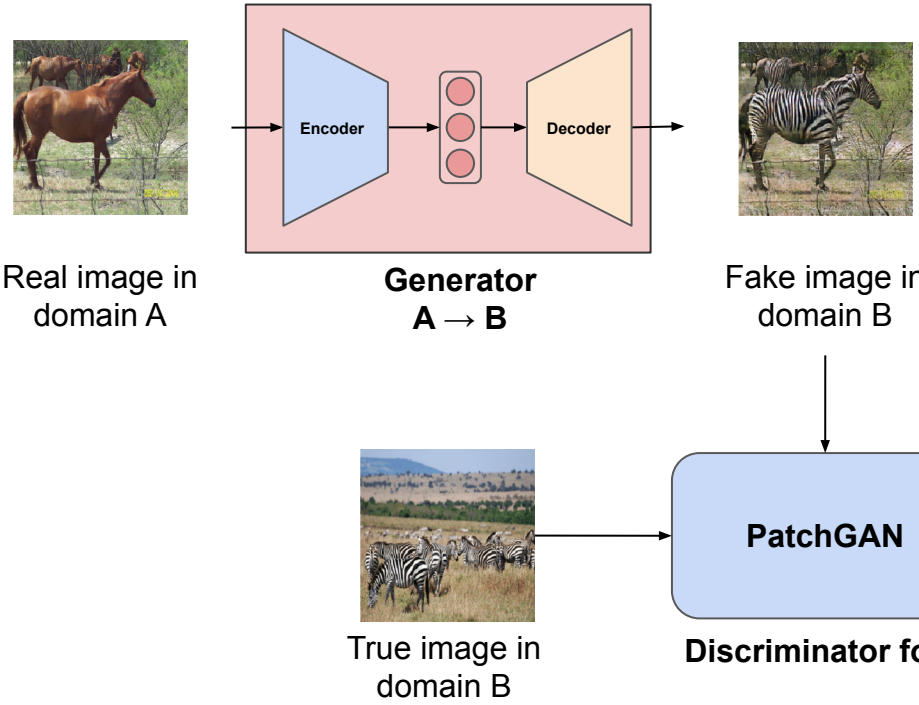


Generator
A → B



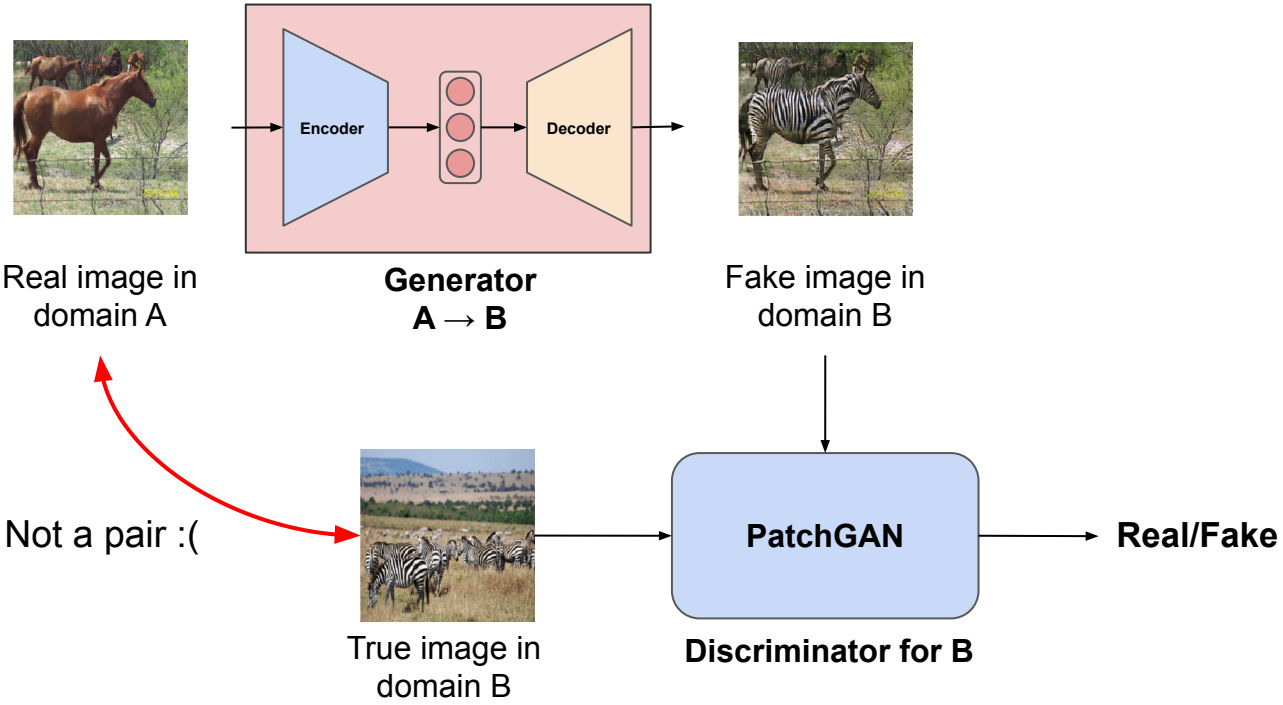
Fake image in domain B

CycleGAN


Encoder
Decoder
Real image in domain A
Generator A → B
Fake image in domain B
True image in domain B
PatchGAN
Real/Fake
Discriminator for B

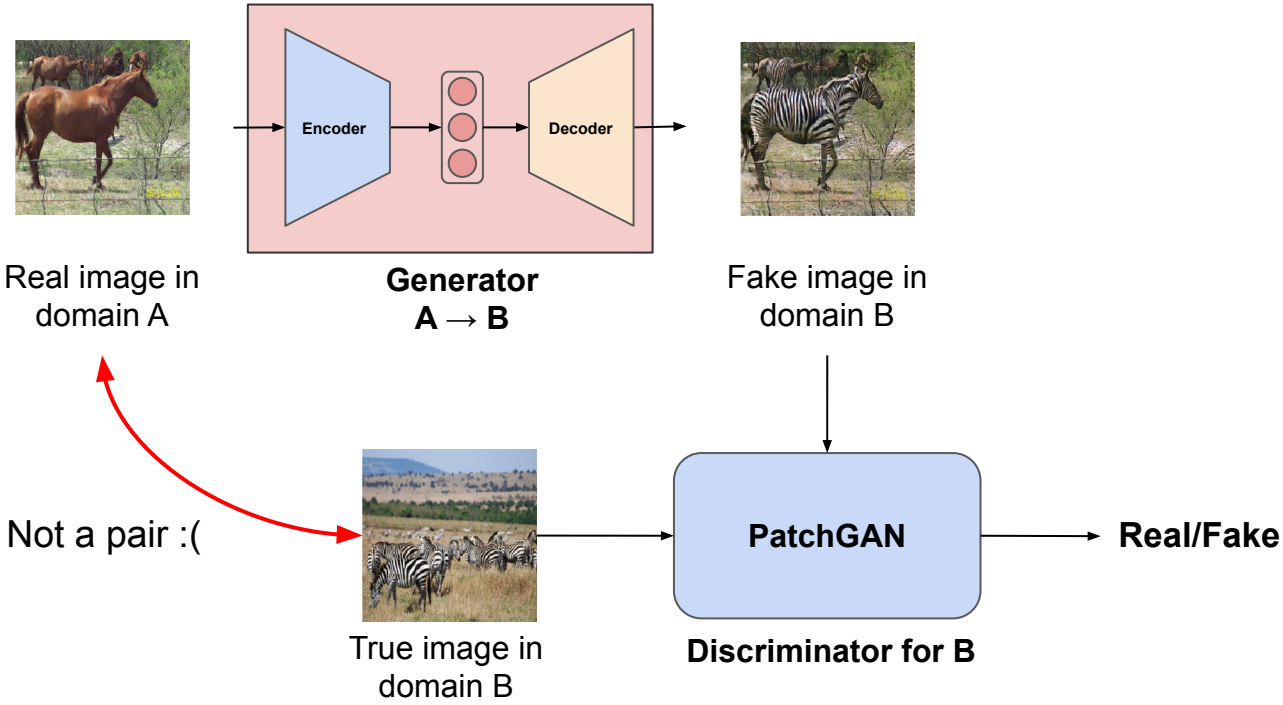
CycleGAN

This is not enough why?

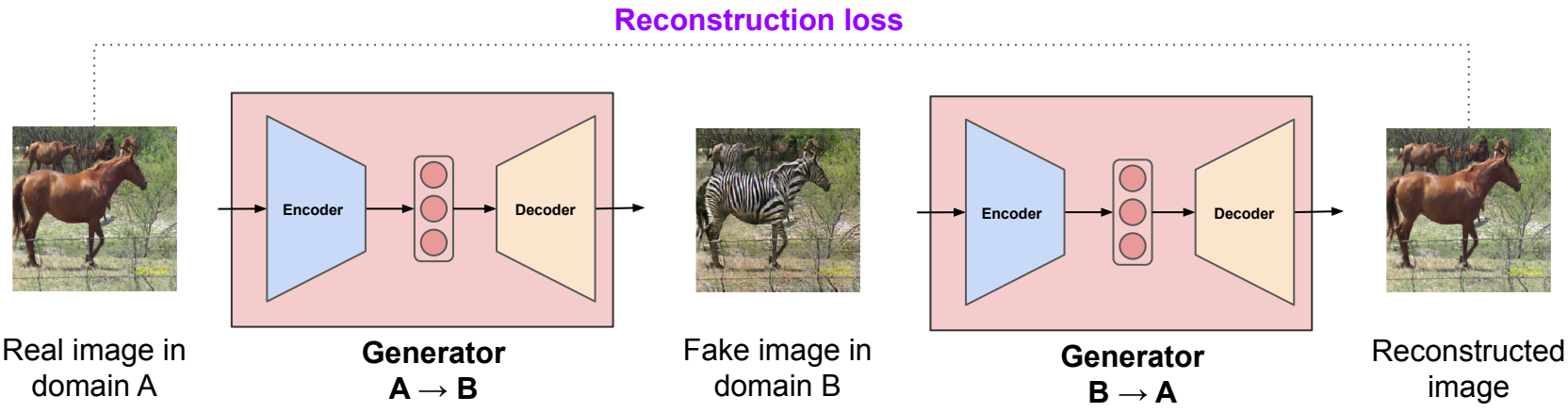


CycleGAN

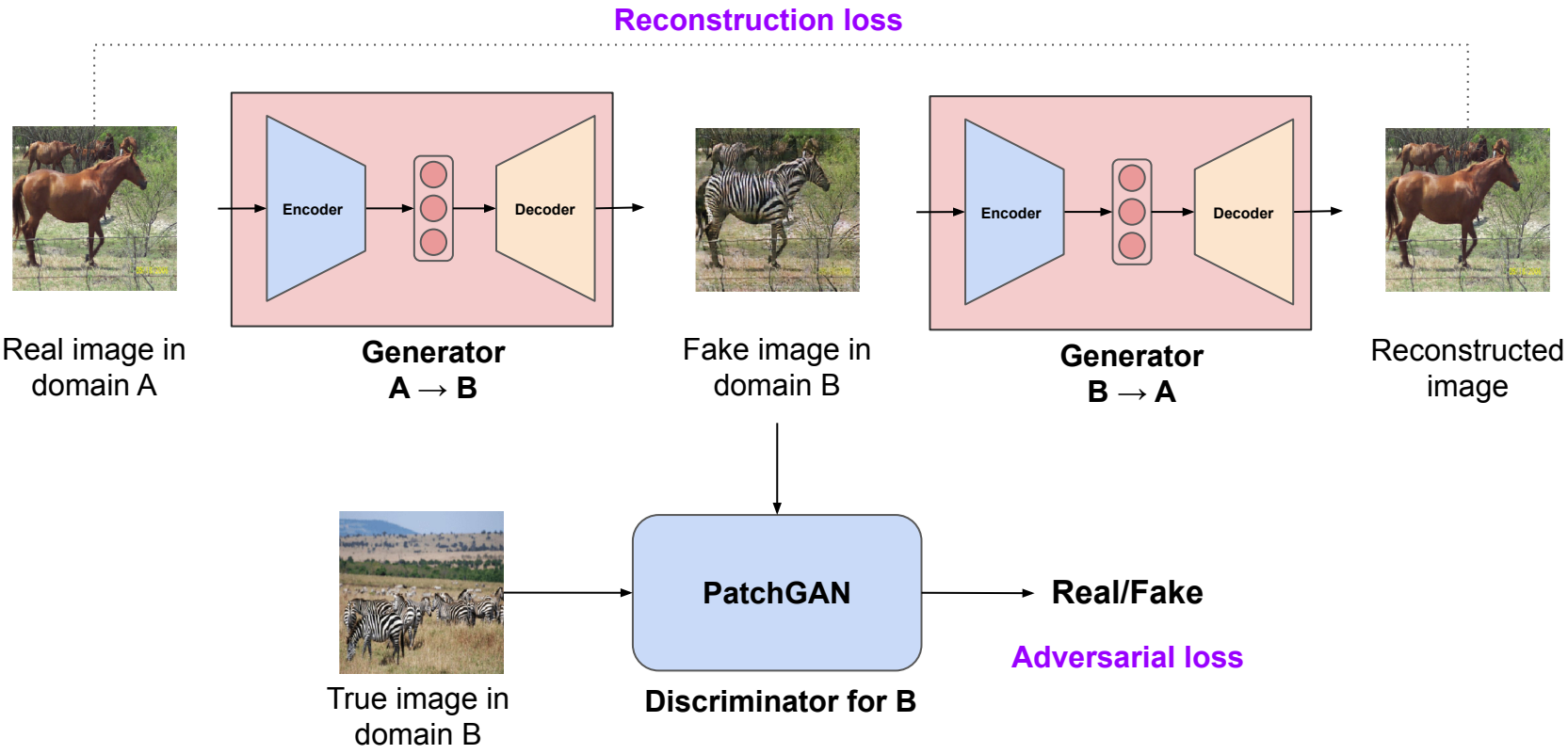
This is not enough why?



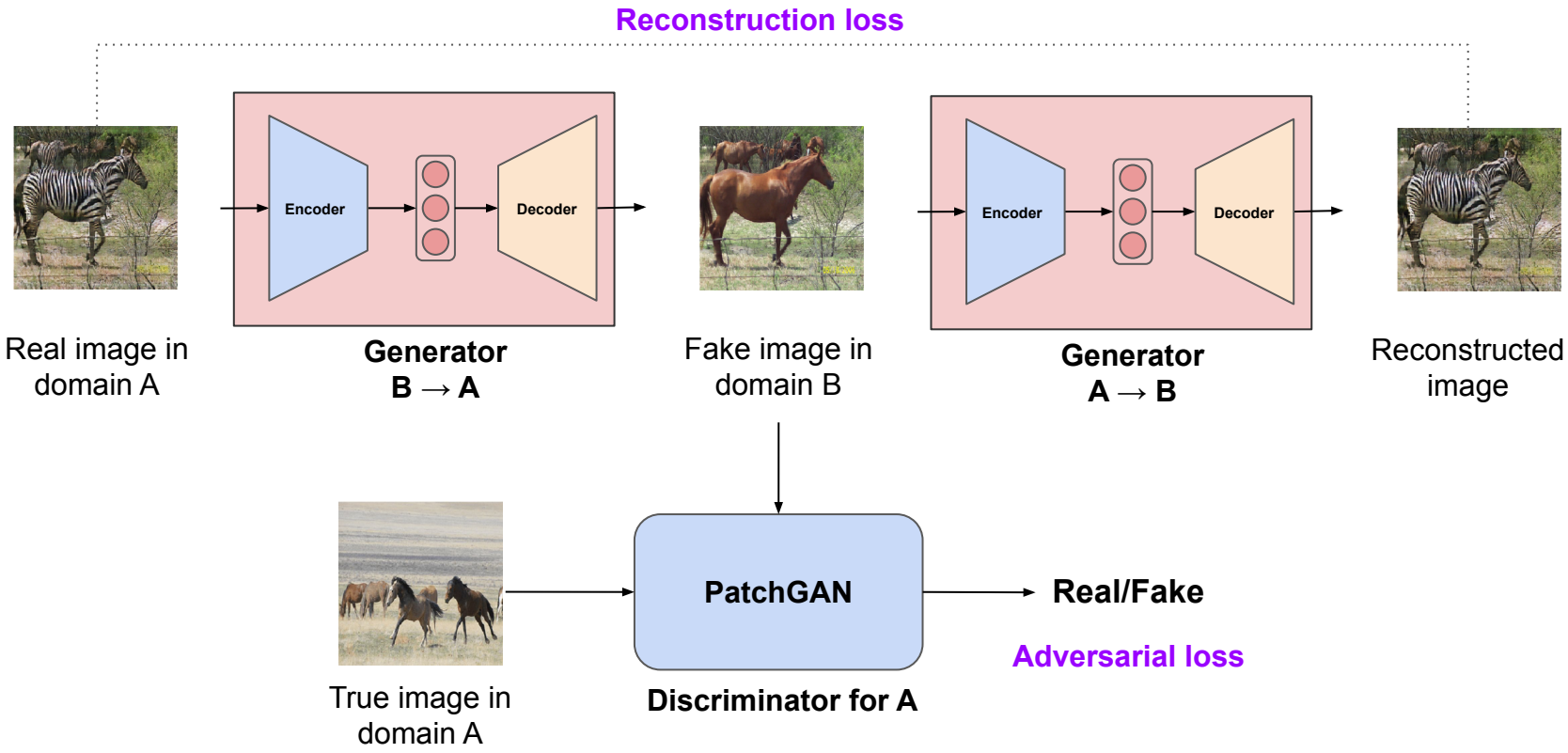
CycleGAN


Reconstruction loss
Encoder
Decoder
Encoder
Decoder
Real image in domain A
Generator A → B
Fake image in domain B
Generator B → A
Reconstructed image
PatchGAN
Real/Fake
True image in domain B
Discriminator for B

CycleGAN: 2x Generator + 2x Discriminator



CycleGAN: 2x Generator + 2x Discriminator



CycleGAN: Summary

- Learning image translation models from unpaired images
- 2x Generator + 2x Discriminator
- Reconstruction loss + Adversarial loss

Advanced GANs

- ✅ DCGAN
- ✅ Conditional GAN
- ✅ Pix2Pix
  - ✅ U-Net (Generator)
  - ✅ PatchGAN (Discriminator)
- ✅ CycleGAN

Discussion: GAN Applications?

Sketch-to-Image (e.g., pix2pix, CycleGAN)

Image Translation (e.g., CycleGAN)

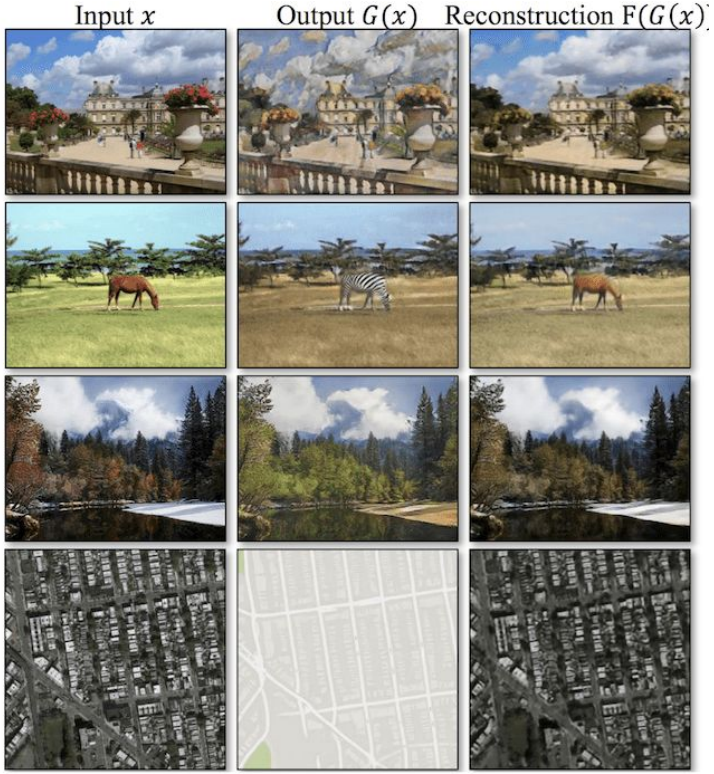
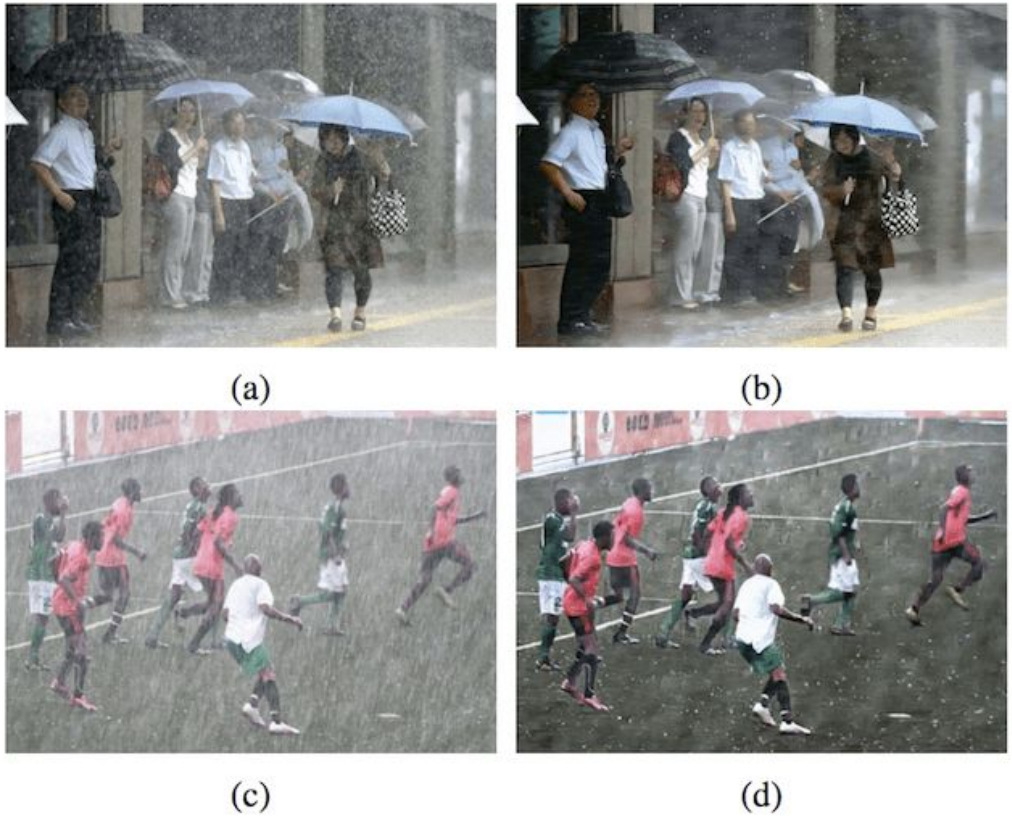
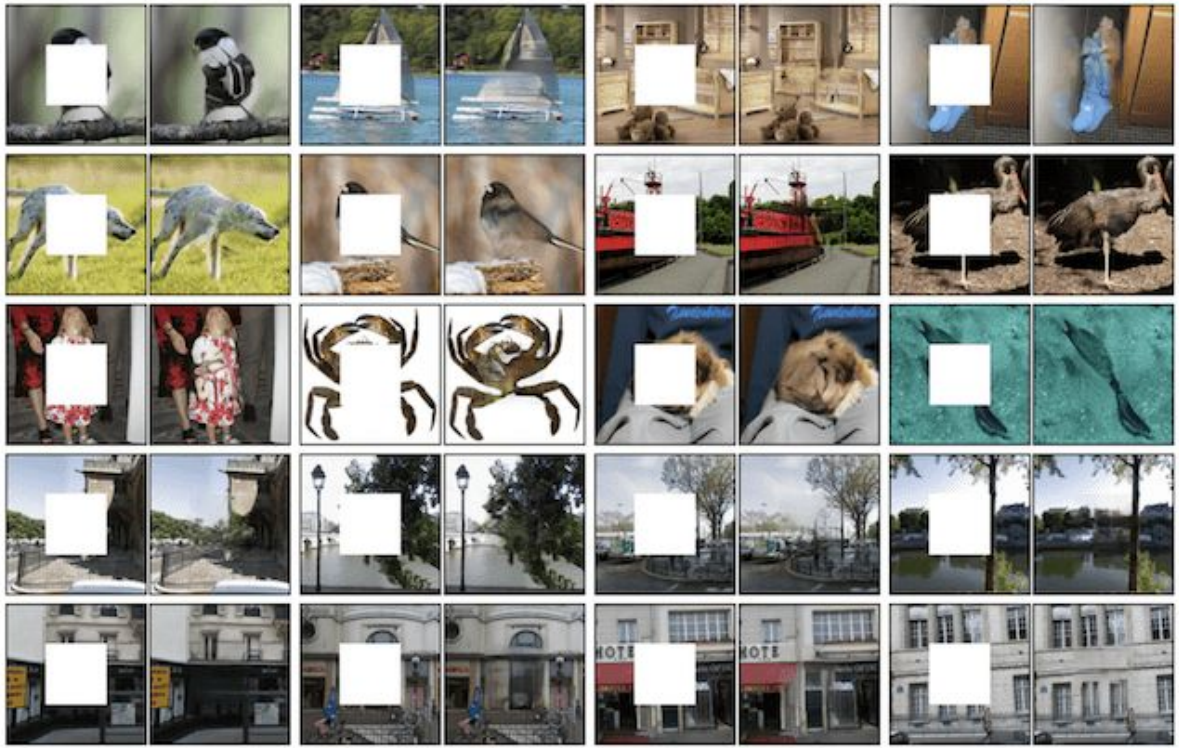


Image De-raining



[Zhang et al. 2017]

Photo Inpainting



[Pathak et al. 2016]

Summary

- The basic concept of **Generative Adversarial Networks (GANs)**
- DCGAN implementation with PyTorch
- Major GAN models and core techniques
  - GAN
  - DCGAN
  - Pix2Pix
  - CycleGAN
  - + U-Net, PatchGAN
- Discussion about GAN applications

We will start discussing Deep Learning techniques for text!