Topics:

- Image Segmentation and Decoders
- Generative Adversarial Networks

CS 4644-DL / 7643-A
ZSOLT KIRA

- **Assignment 3**
  - Due **March 9th 11:59pm EST**
  - **Oops.** Diffusion models accidentally included. No need to do it by Mar 9th! @258

- **Projects**
  - Project proposal due **March ~~15th~~ 17th**
  - Proposal description out on canvas @256

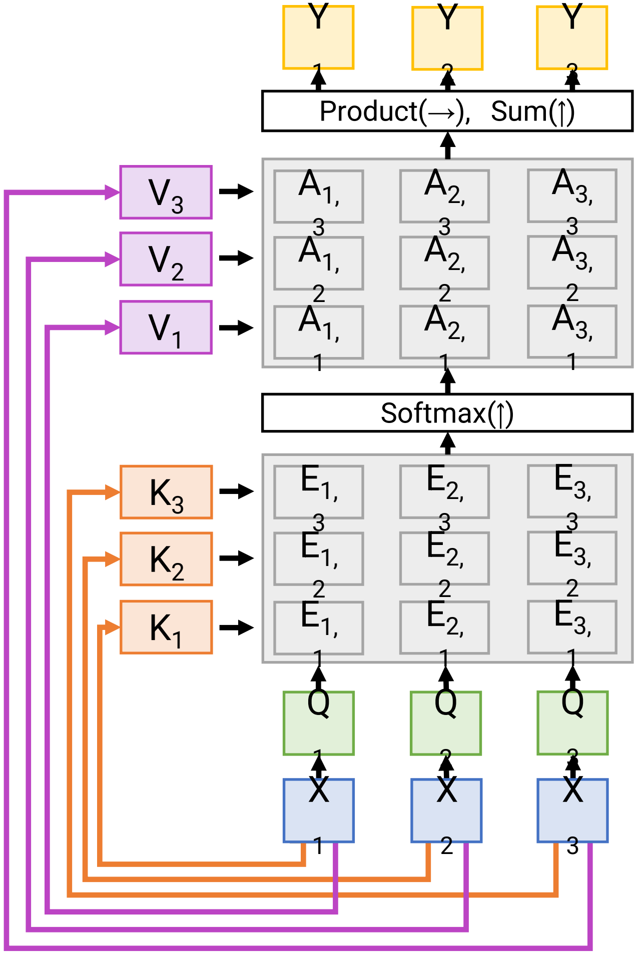
- Meta office hours today 3pm ET on language models

Self-Attention Layer

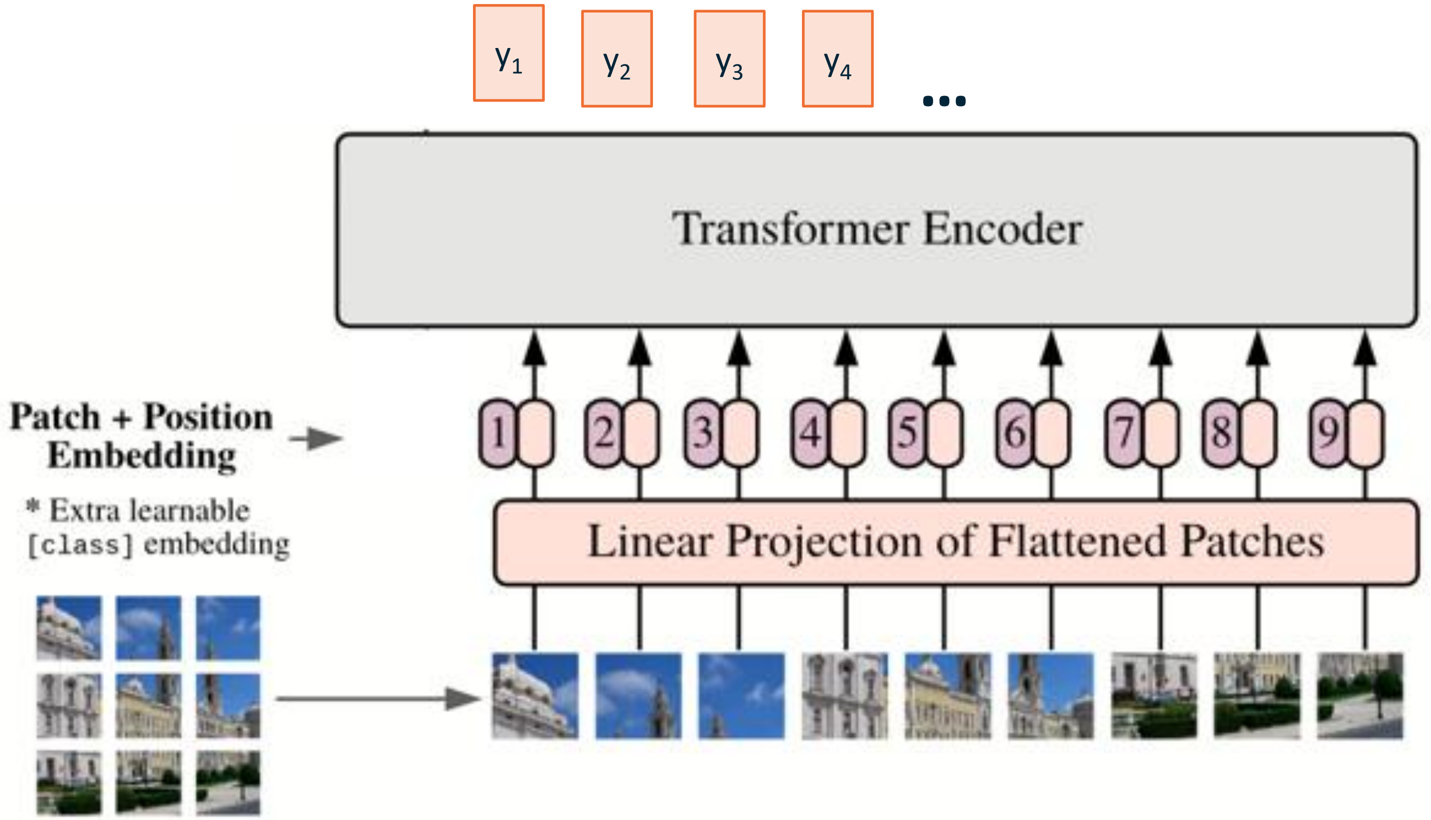
One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



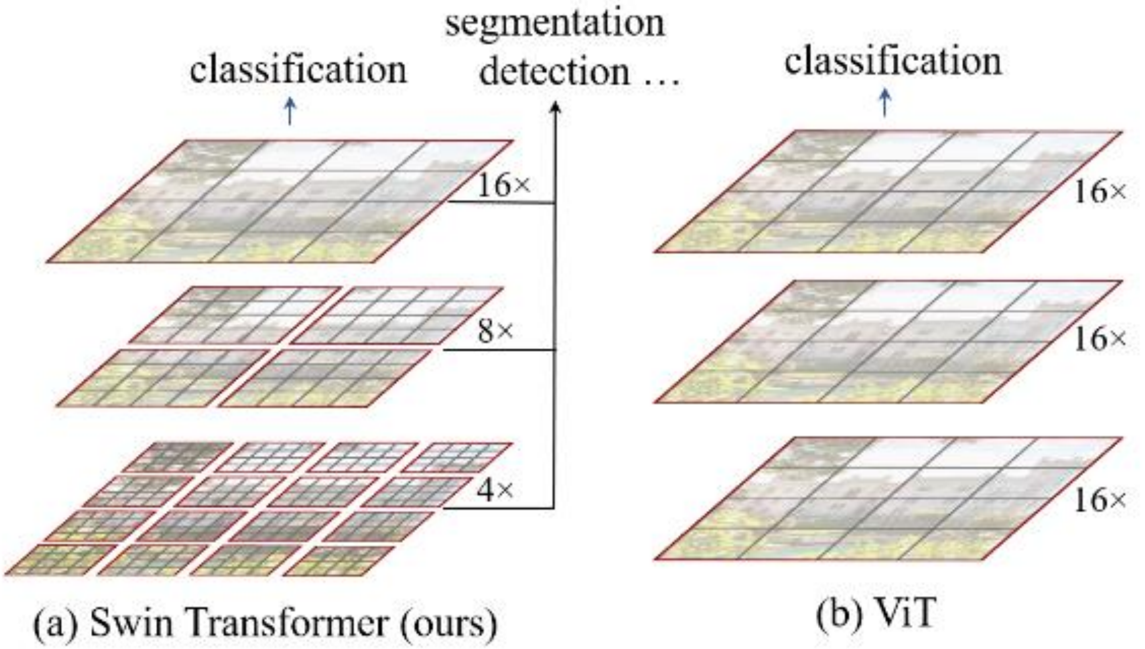
Slide credit: Justin Johnson



y_1
y_2
y_3
y_4
…
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

- How do we do classification?

Patches as input to Self-Attention



Ideas:

- Use smaller patches (4x4x3)
- Project them to lower dimension (4)
- Merge tokens at deeper levels
- Full attention => Window attention
    - => Shifted window attention

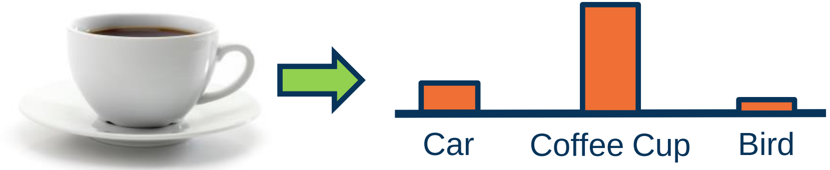
Swin Transformer: Hierarchical Vision Transformer using Shifted Windows
Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo

Swin Transformers

https://paperswithcode.com/sota/instance-segmentation-on-coco

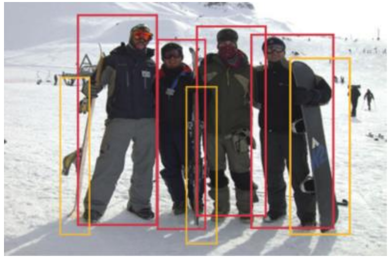
Image Segmentation Networks

Computer Vision Tasks



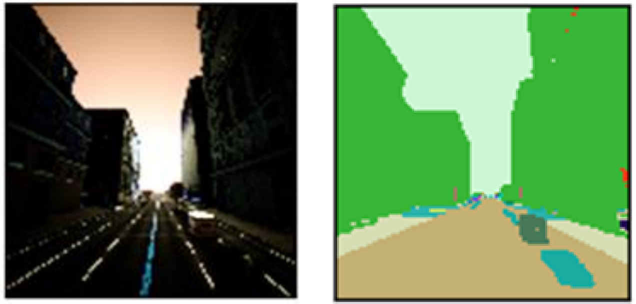
Classification

(Class distribution per image)



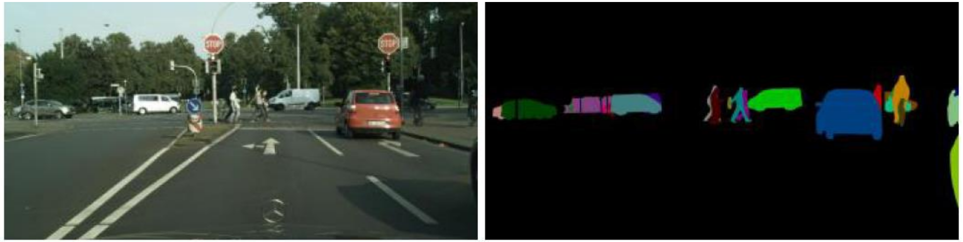
Object Detection

(List of bounding boxes with class distribution per box)



Semantic Segmentation

(Class distribution per pixel)



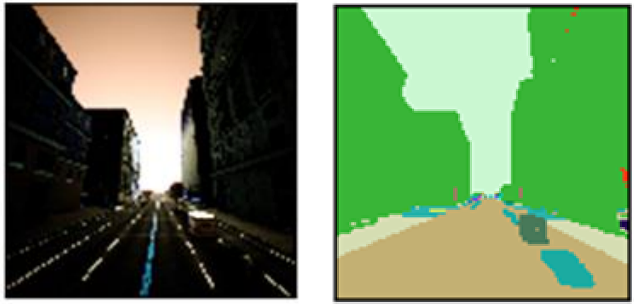
Instance Segmentation

(Class distribution per pixel with unique ID)

Segmentation Tasks

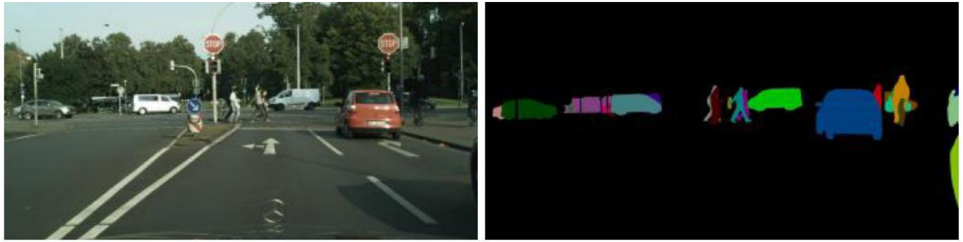
Given an image, output another image

- Each output contains class distribution per pixel
- More generally an image-to-image problem



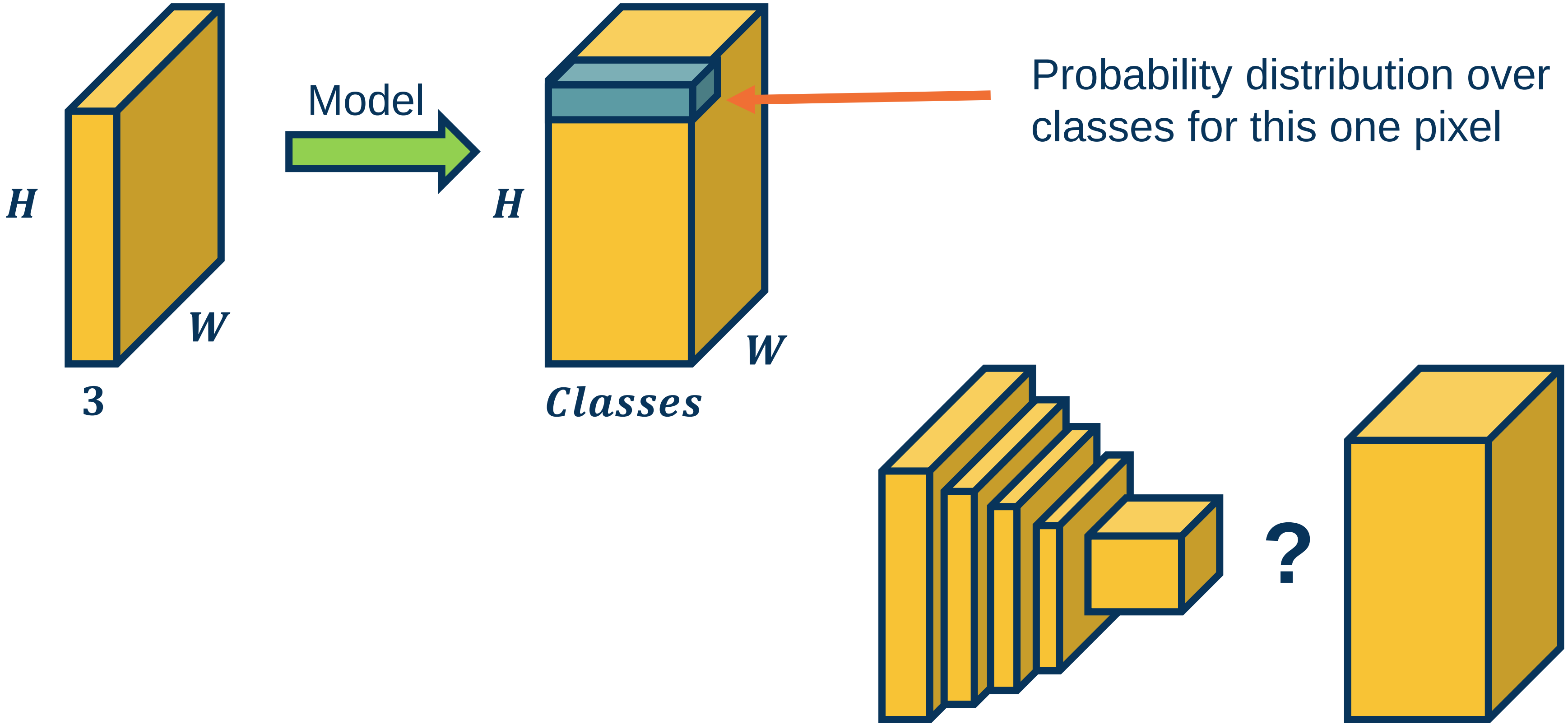
Semantic Segmentation

(Class distribution per pixel)



Instance Segmentation

(Class distribution per pixel with unique ID)



Probability distribution over classes for this one pixel

Input & Output

Convolutional Neural Network (CNN)

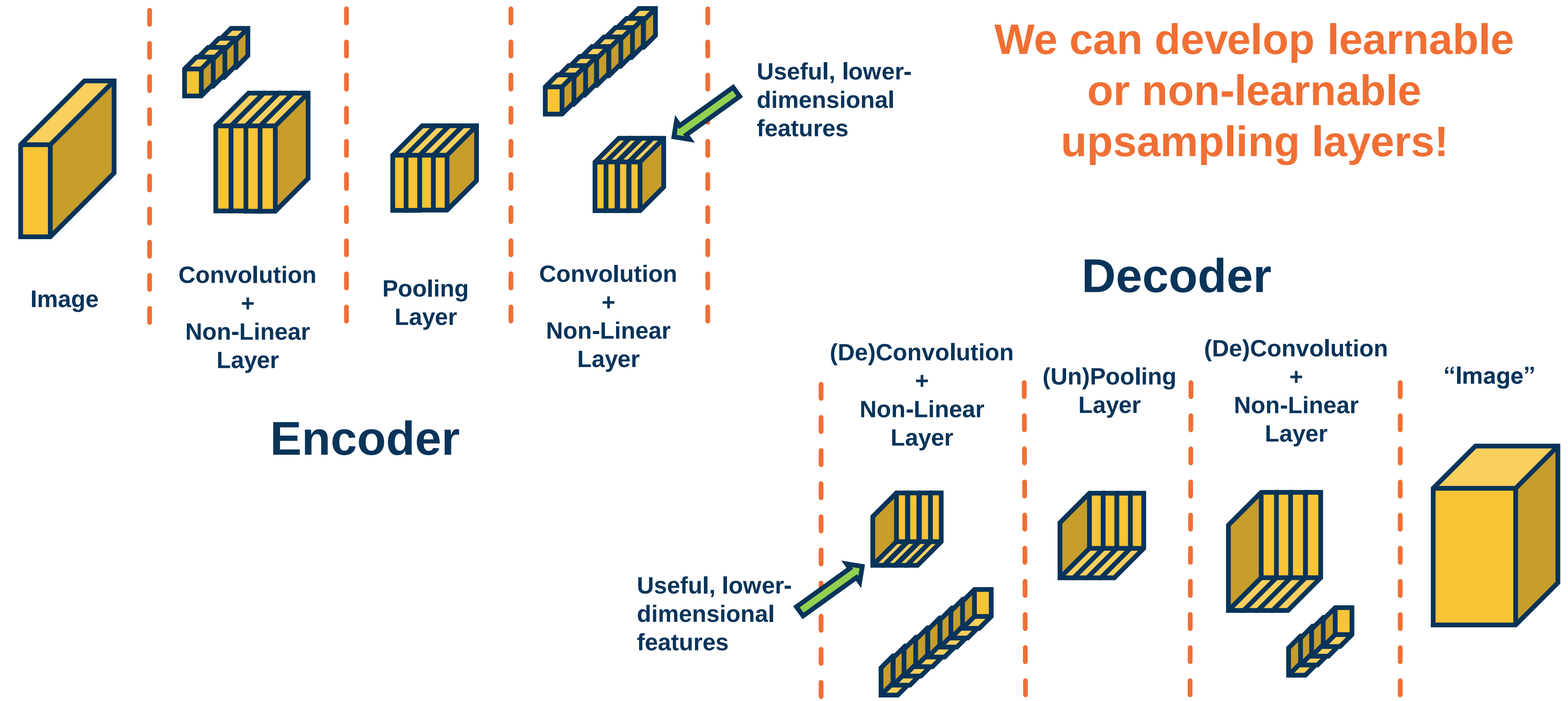


Image
Convolution + Non-Linear Layer
Pooling Layer
Convolution + Non-Linear Layer
Useful, lower-dimensional features
Encoder
Decoder
(De)Convolution + Non-Linear Layer
(Un)Pooling Layer
(De)Convolution + Non-Linear Layer
“Image”
Useful, lower-dimensional features

We can develop learnable or non-learnable upsampling layers!

Idea: “De”Convolution and UnPooling

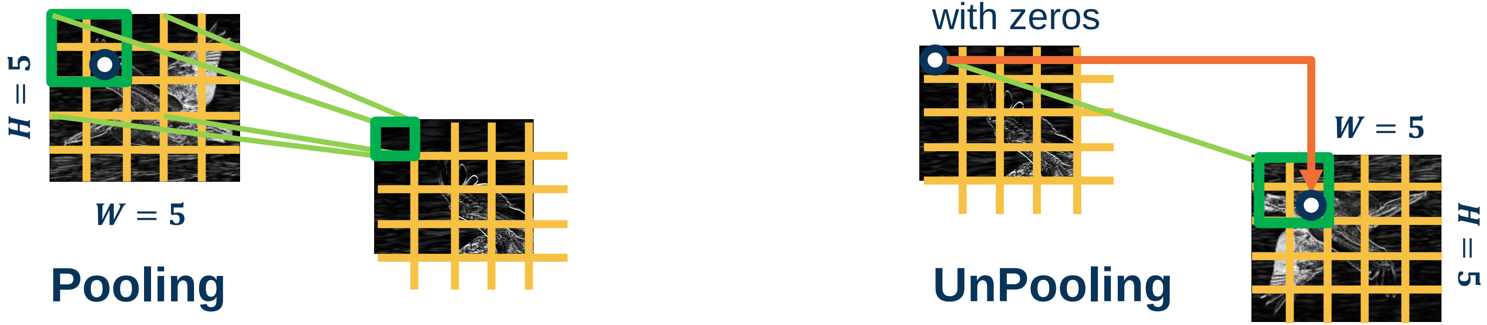
Max Unpooling

Example : Max pooling

- Stride window across image but perform per-patch **max operation**

$$X(0{:}1,0{:}1) = \begin{bmatrix}100 & 150\\ 100 & 200\end{bmatrix} \Rightarrow \max(0{:}1,0{:}1) = 200$$

Copy value to position chosen as max in encoder, fill reset of this window with zeros



Pooling

UnPooling

Idea: Remember max elements in encoder! Copy value from equivalent position, rest are zeros

$$X = \begin{bmatrix} 120 & 150 & 120 \\ 100 & 50 & 110 \\ 25 & 25 & 10 \end{bmatrix} \xrightarrow[\text{2x2 max pool}]{} Y = \begin{bmatrix} 150 & 150 \\ 100 & 110 \end{bmatrix}$$

Encoder

Decoder

$$X = \begin{bmatrix} 300 & 450 \\ 100 & 250 \end{bmatrix} \xrightarrow{\text{2x2 max unpool}} Y = \begin{bmatrix} 0 & 300 & - \\ 0 & 0 & - \\ - & - & - \end{bmatrix}$$

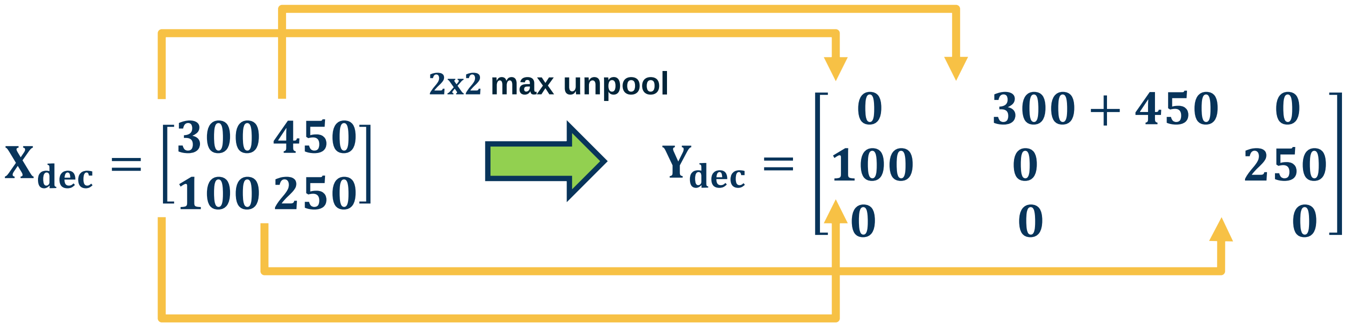
Max Unpooling Example (one window)

$$\mathrm{X_{enc}} = \begin{bmatrix} 120 & 150 & 120 \\ 100 & 50 & 110 \\ 25 & 25 & 10 \end{bmatrix} \xrightarrow{\text{2x2 max pool}} \mathrm{Y_{enc}} = \begin{bmatrix} 150 & 150 \\ 100 & 110 \end{bmatrix}$$

Contributions from multiple windows are summed

Encoder

Decoder



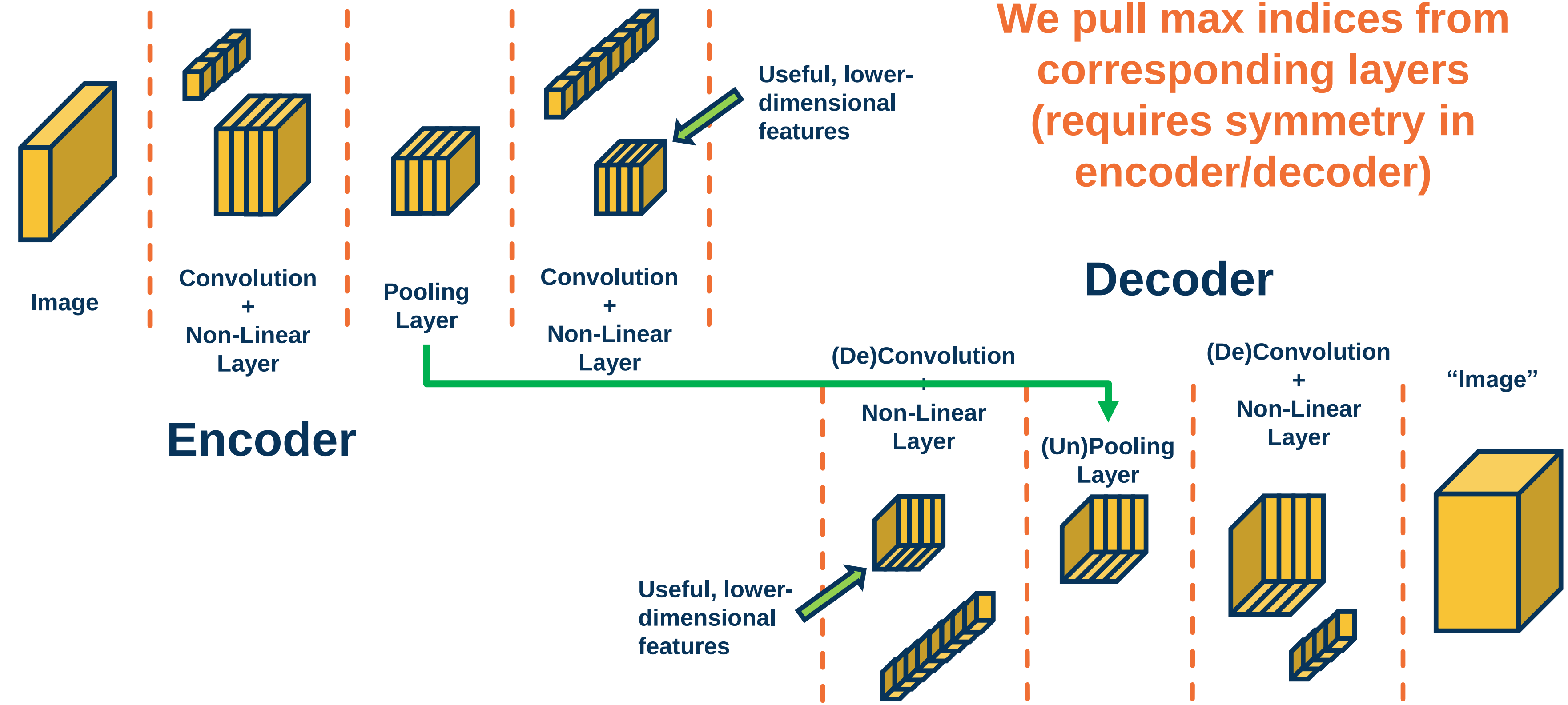
2x2 max unpool

$$\mathrm{X_{dec}} = \begin{bmatrix} 300 & 450 \\ 100 & 250 \end{bmatrix} \rightarrow \mathrm{Y_{dec}} = \begin{bmatrix} 0 & 300+450 & 0 \\ 100 & 0 & 250 \\ 0 & 0 & 0 \end{bmatrix}$$

Max Unpooling Example

Symmetry in Encoder/Decoder

Convolutional Neural Network (CNN)



Image

Convolution + Non-Linear Layer

Pooling Layer

Convolution + Non-Linear Layer

Useful, lower-dimensional features

Encoder

Decoder

(De)Convolution + Non-Linear Layer

(Un)Pooling Layer

(De)Convolution + Non-Linear Layer

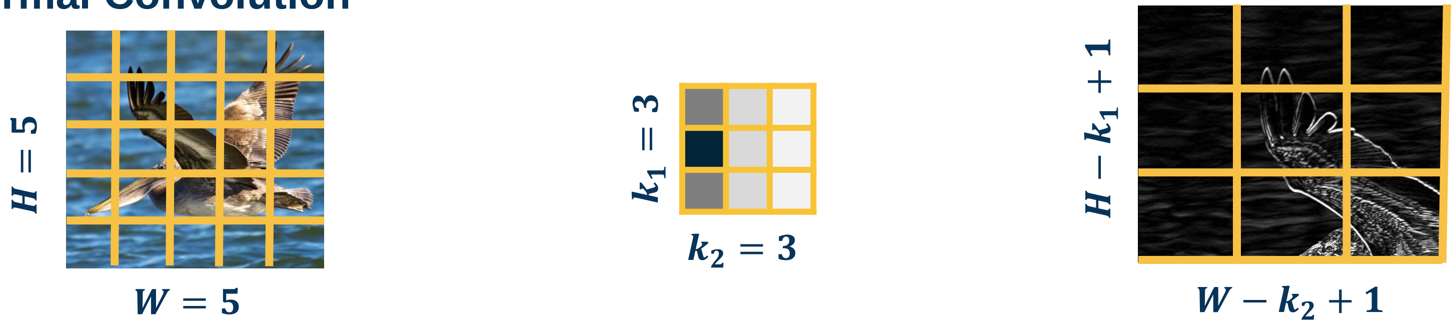
“Image”

Useful, lower-dimensional features

We pull max indices from corresponding layers (requires symmetry in encoder/decoder)

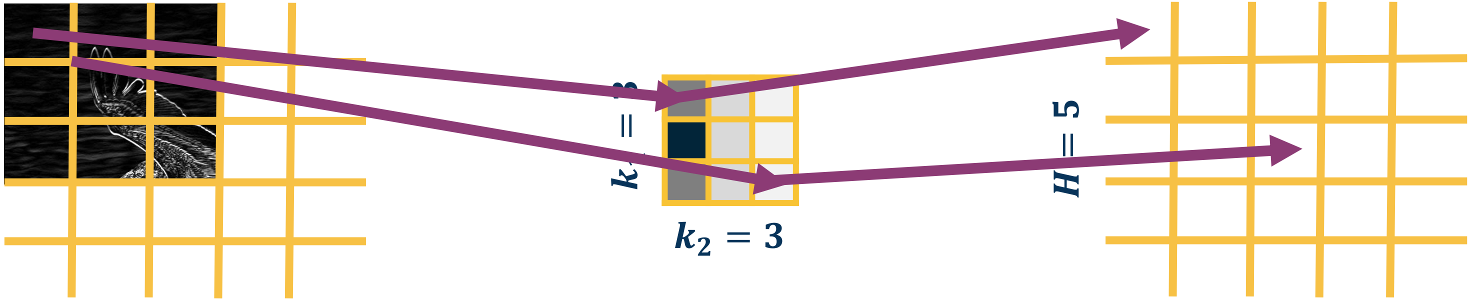
How can we *upsample* using convolutions and learnable kernel?

Normal Convolution



Transposed Convolution (also known as “deconvolution”, fractionally strided conv)

Idea: Take each input pixel, multiply by learnable kernel, “stamp” it on output



$$X = \begin{bmatrix} 120 & 150 & 120 \\ 100 & 50 & 110 \\ 25 & 25 & 10 \end{bmatrix} \qquad K = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}$$

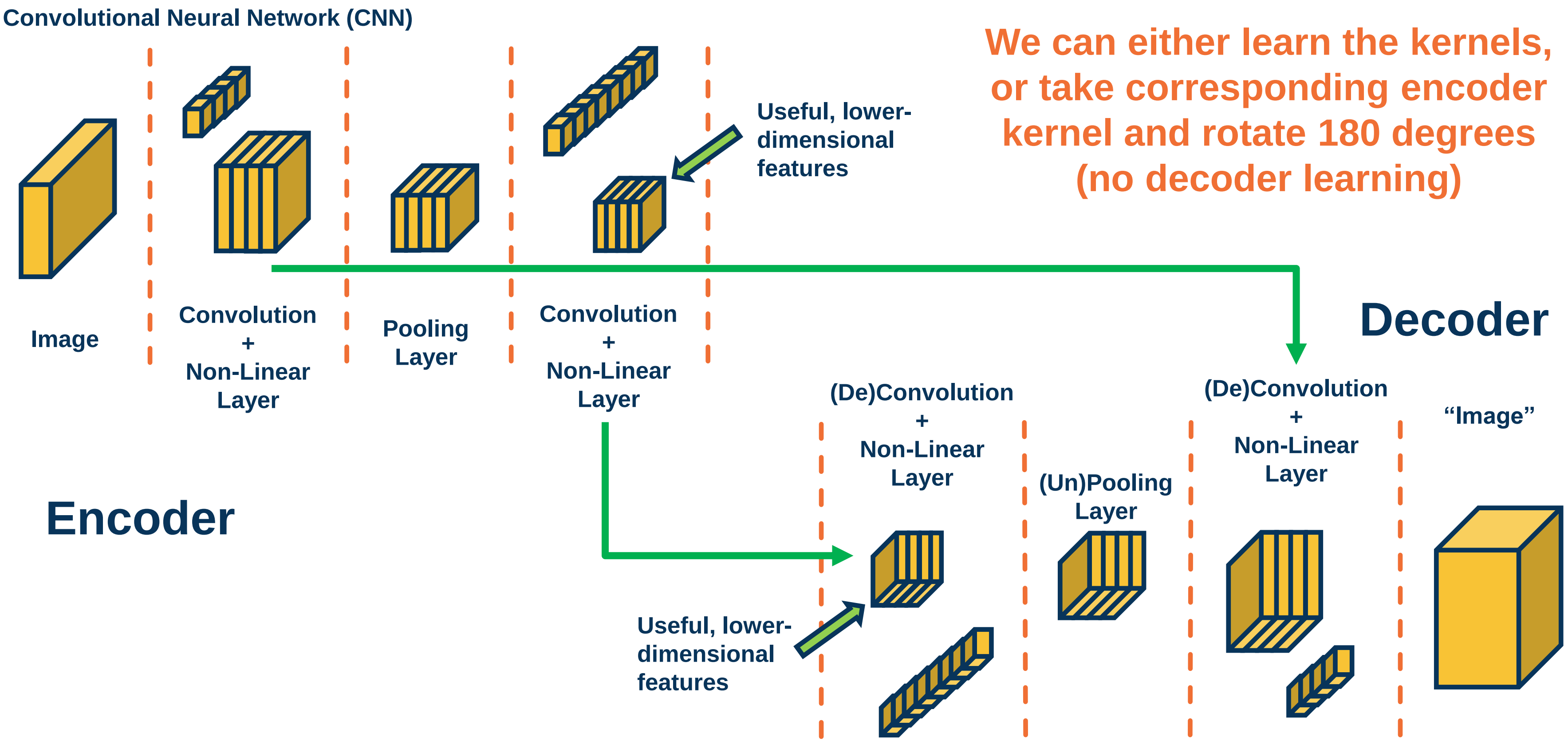
Contributions from multiple windows are summed

$$\begin{bmatrix} 120 & -120 & 0 & 0 \\ 240 & -240 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 120 & -120+150 & -150 & 0 \\ 240 & -240+300 & -300 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Incorporate $X(0,0)$

Incorporate $X(1,0)$

Transposed Convolution Example

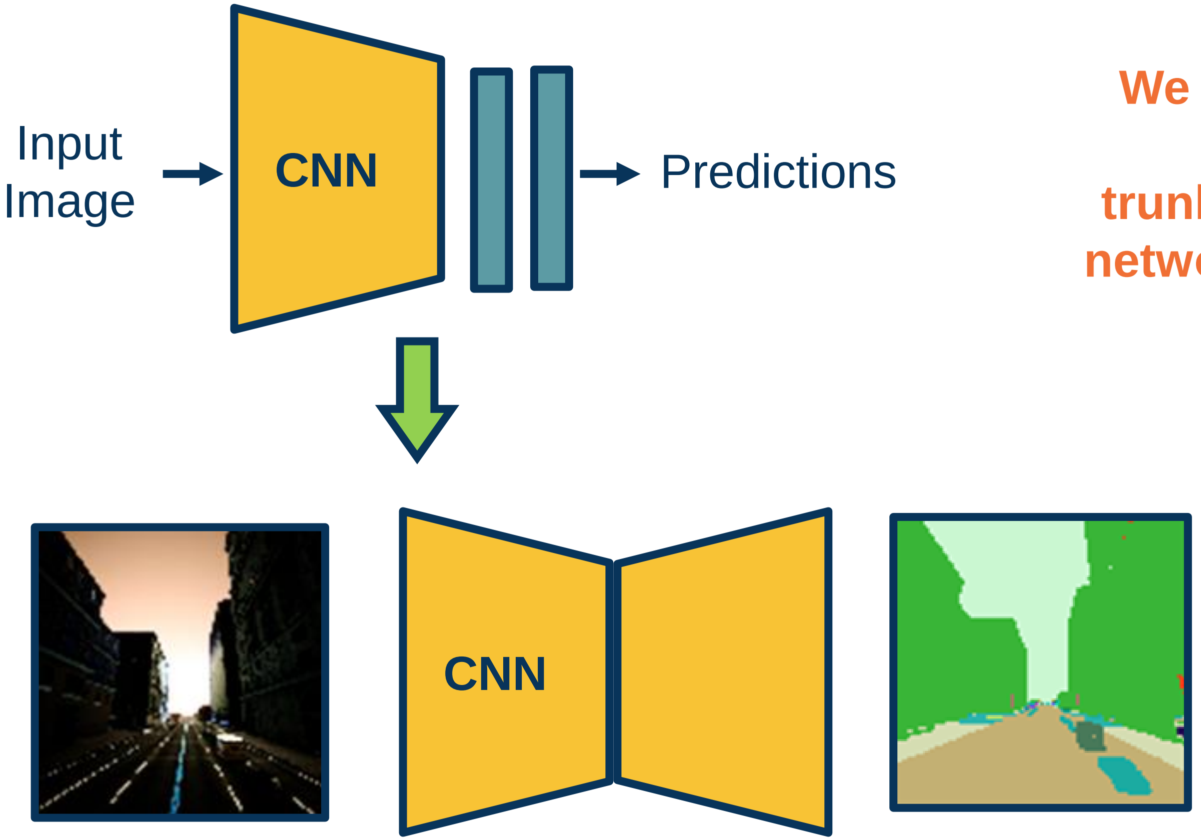


Convolutional Neural Network (CNN)
Image
Convolution + Non-Linear Layer
Pooling Layer
Convolution + Non-Linear Layer
Useful, lower-dimensional features
Encoder
Decoder
(De)Convolution + Non-Linear Layer
(Un)Pooling Layer
(De)Convolution + Non-Linear Layer
“Image”
Useful, lower-dimensional features

We can either learn the kernels, or take corresponding encoder kernel and rotate 180 degrees (no decoder learning)

Symmetry in Encoder/Decoder

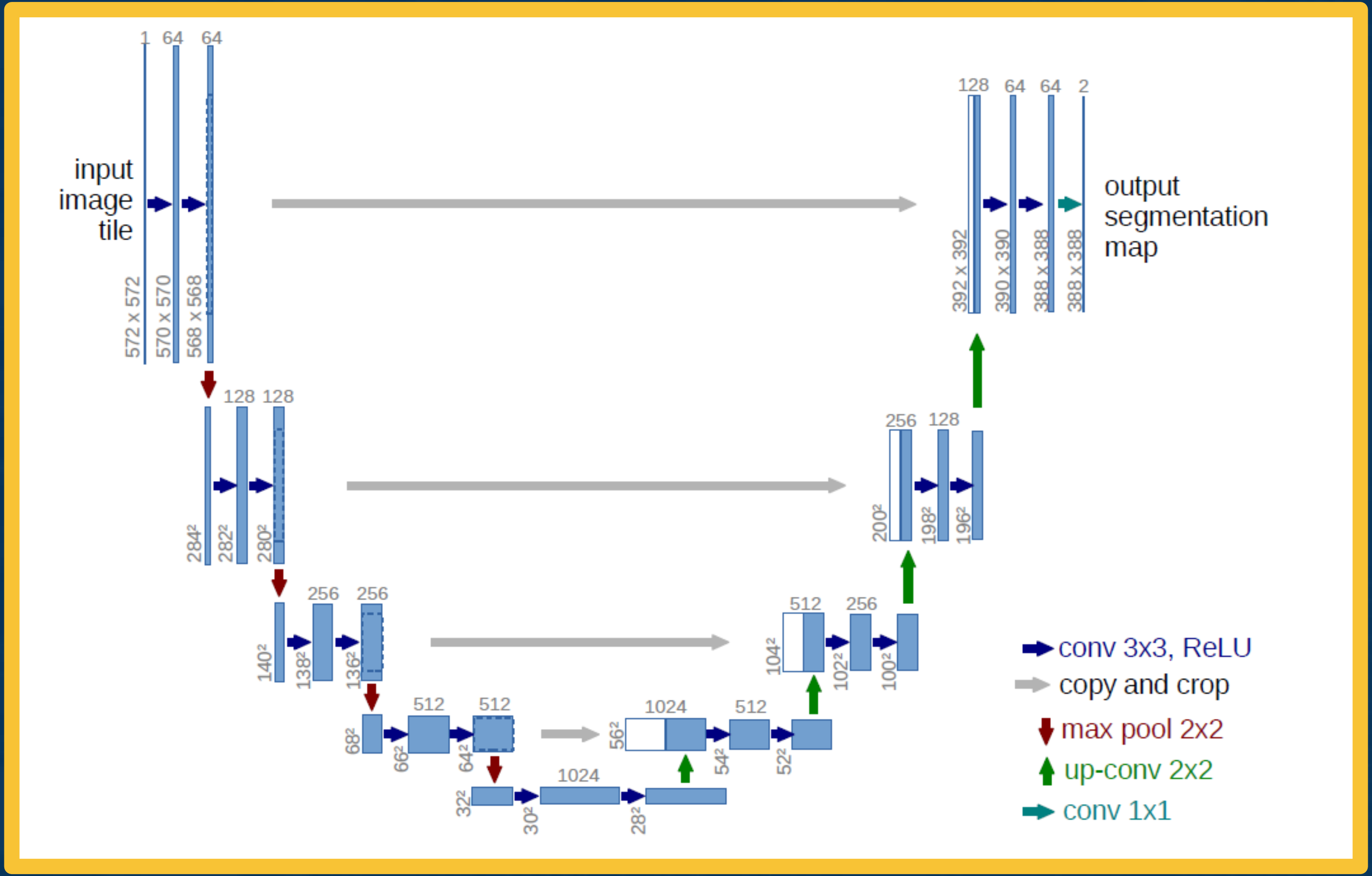
Transfer Learning



We can start with a pre-trained trunk/backbone (e.g. network pretrained on ImageNet)!

U-Net

You can have skip connections to bypass bottleneck!



Ronneberger, et al., "U-Net: Convolutional Networks for Biomedical Image Segmentation", 2015

Summary

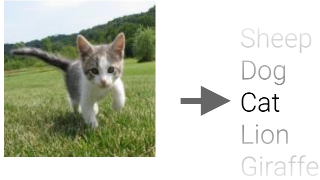
- Various ways to get **image-like outputs,** for example to predict segmentations of input images
- We can have various upsampling layers that actually increase the size
- Encoder/decoder architectures are popular ways to leverage these to perform general image-to-image tasks
- Other methods exist:
  - Fully convolutional neural networks

Generative Models: Introduction

Spectrum of Low-Labeled Learning

Supervised Learning

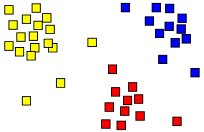
- Train Input: $\{X, Y\}$
- Learning output: $f : X \rightarrow Y$, $P(y|x)$
- e.g. classification



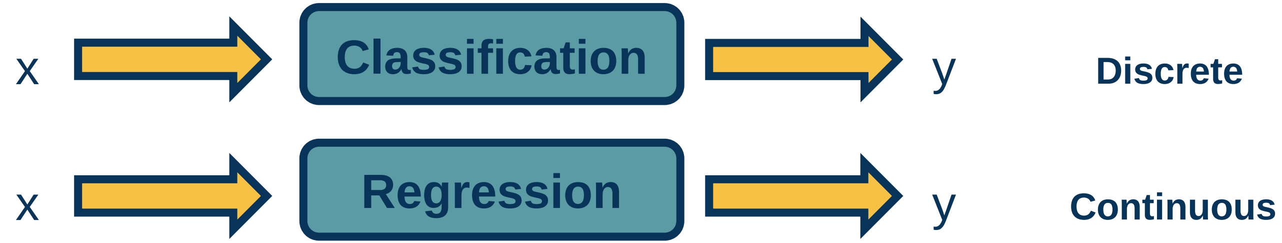
Less Labels →

Unsupervised Learning

- Input: $\{X\}$
- Learning output: $P(x)$
- Example: Clustering, density estimation, etc.

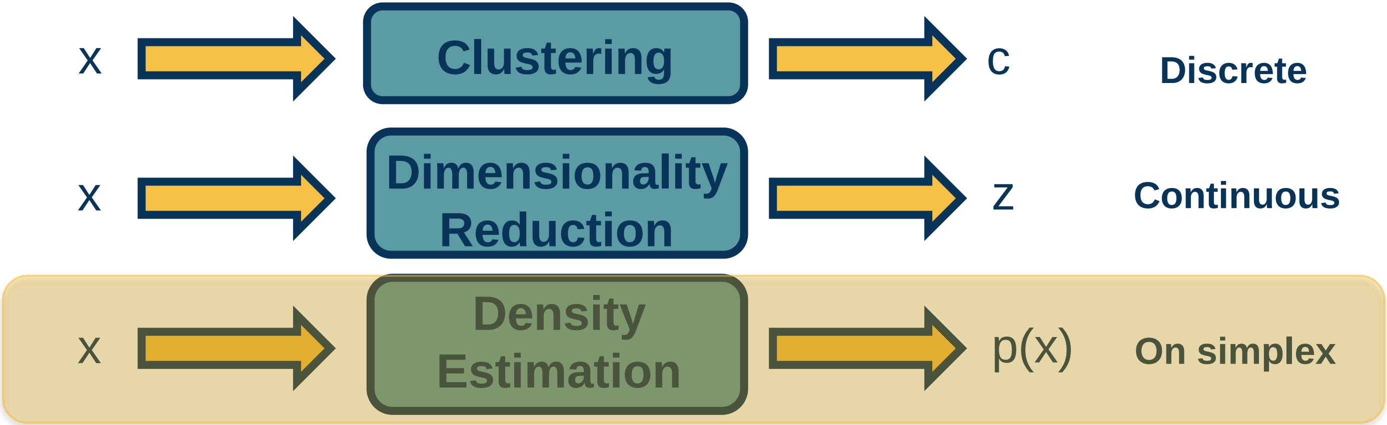


Supervised Learning



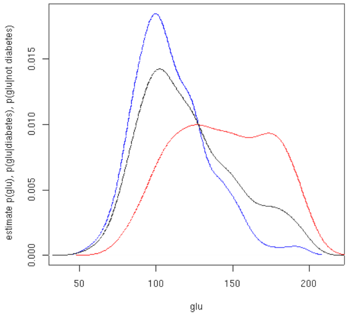
x	Classification	y	Discrete
x	Regression	y	Continuous

Unsupervised Learning



x	Clustering	c	Discrete
x	Dimensionality Reduction	z	Continuous
x	Density Estimation	p(x)	On simplex

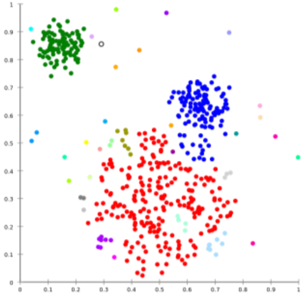
Traditional unsupervised learning methods:



Density estimation

Modeling $P(x)$

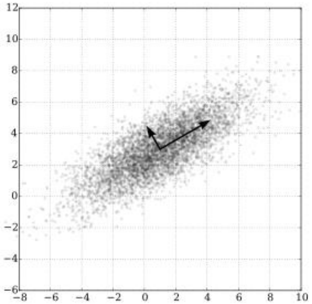
Deep Generative Models



Clustering

Comparing/ Grouping

Metric learning & clustering



Principal Component Analysis

Representation Learning

Almost all deep learning!

Similar in deep learning, but **from neural network/learning perspective**

Discriminative vs. Generative Models

- Discriminative models model $P(y|x)$
  - Example: Model this via neural network, SVM, etc.

- Generative models model $P(x)$

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Discriminative vs. Generative Models

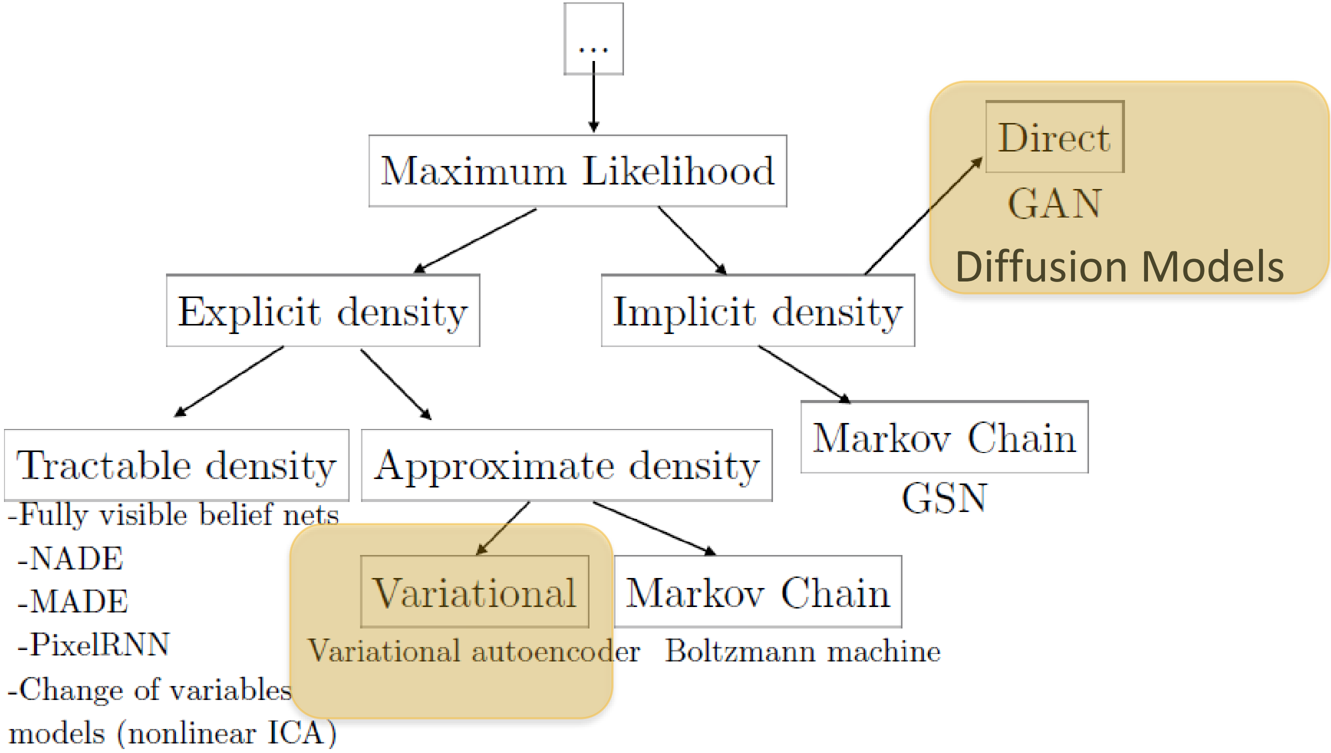
- Discriminative models model $P(y|x)$
  - Example: Model this via neural network, SVM, etc.

- Generative models model $P(x)$
- We can parameterize our model as $P(x, \theta)$ and use maximum likelihood to optimize the parameters given an unlabeled dataset:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \arg\max_{\boldsymbol{\theta}} \log \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right).\end{aligned}$$

- They are called generative because they can often generate *samples*
  - Example: Multivariate Gaussian with estimated parameters $\boldsymbol{\mu}, \boldsymbol{\sigma}$

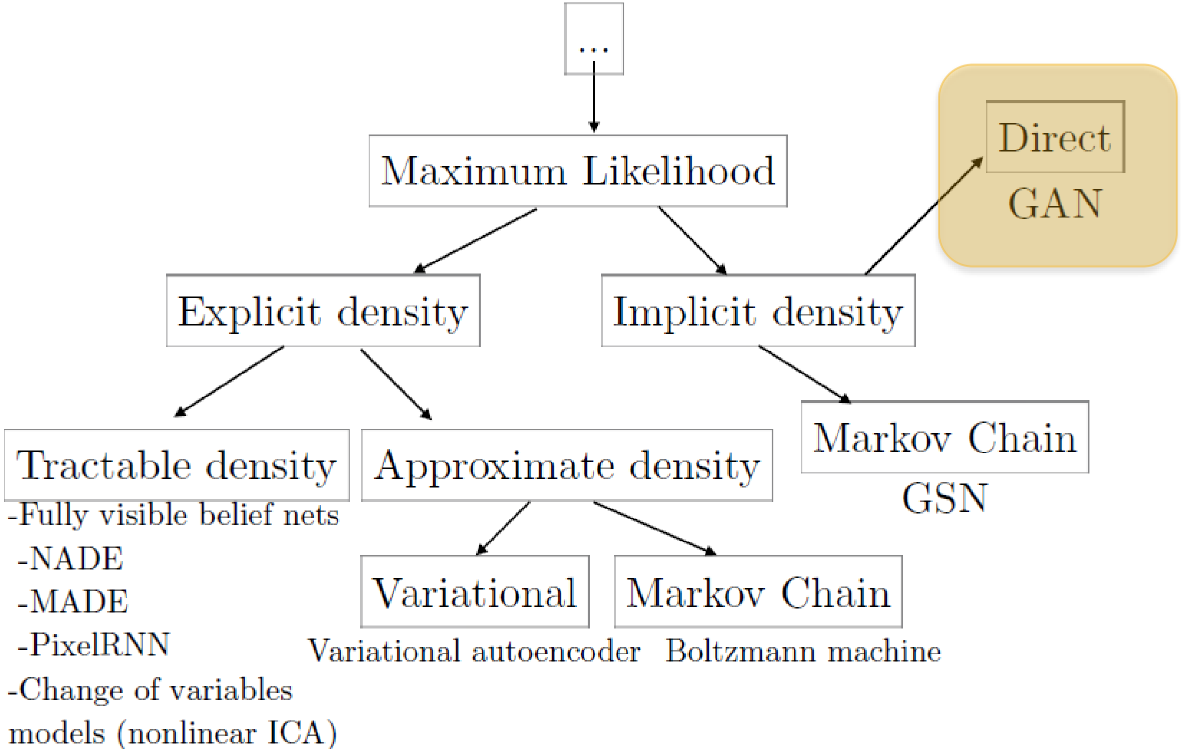
Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks



Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Generative Models

Generative Adversarial Networks (GANs)



...

Maximum Likelihood

Explicit density

Implicit density

Direct

GAN

Tractable density

-Fully visible belief nets

-NADE

-MADE

-PixelRNN

-Change of variables models (nonlinear ICA)

Approximate density

Markov Chain

GSN

Variational

Variational autoencoder

Markov Chain

Boltzmann machine

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Generative Models

- *Implicit* generative models do not actually learn an explicit model for $\boldsymbol{p(x)}$

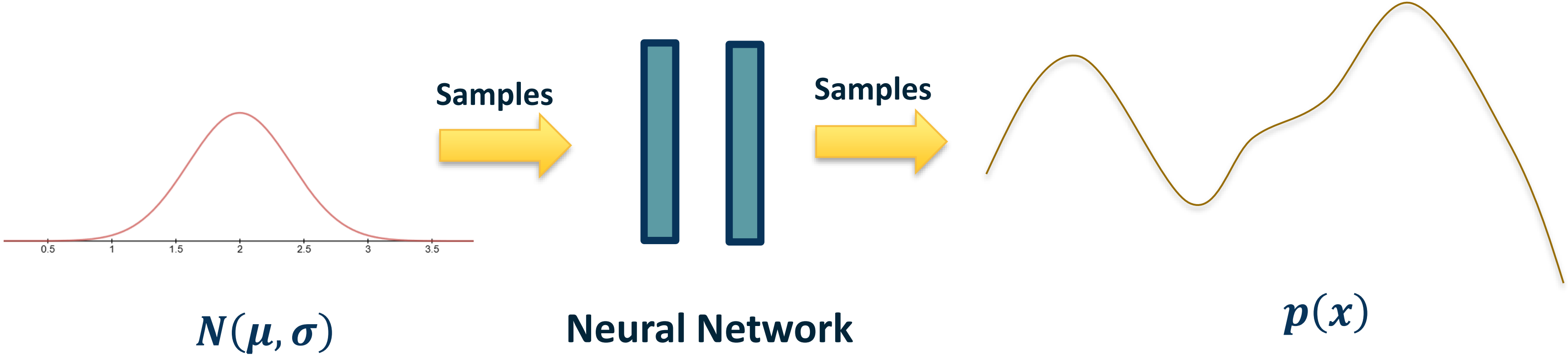
- Instead, learn to *generate samples* from $\boldsymbol{p(x)}$
  - Learn good feature representations
  - Perform data augmentation
  - Learn world models (a simulator!) for reinforcement learning

- How?
  - **Learn to sample** from a neural network output
  - **Adversarial training** that uses one network's predictions to train the other (dynamic loss function!)
  - **Lots of tricks** to make the optimization more stable

Implicit Models

Learning to Sample

- We would like to *sample* from $\boldsymbol{p(x)}$ using a neural network
- **Idea:**
  - Sample from a simple distribution (Gaussian)
  - Transform the sample to $\boldsymbol{p(x)}$



Samples

Samples

$\boldsymbol{N(\mu, \sigma)}$

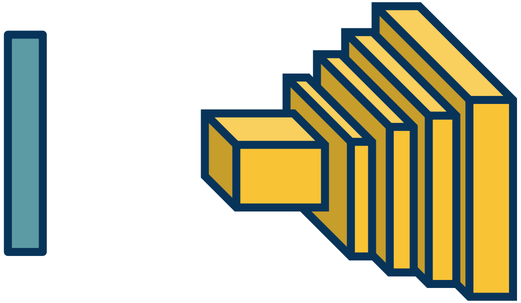
Neural Network

$\boldsymbol{p(x)}$

- Input can be a vector with (independent) Gaussian random numbers
- We can use a CNN to generate images!

Generator

Vector of Random Numbers



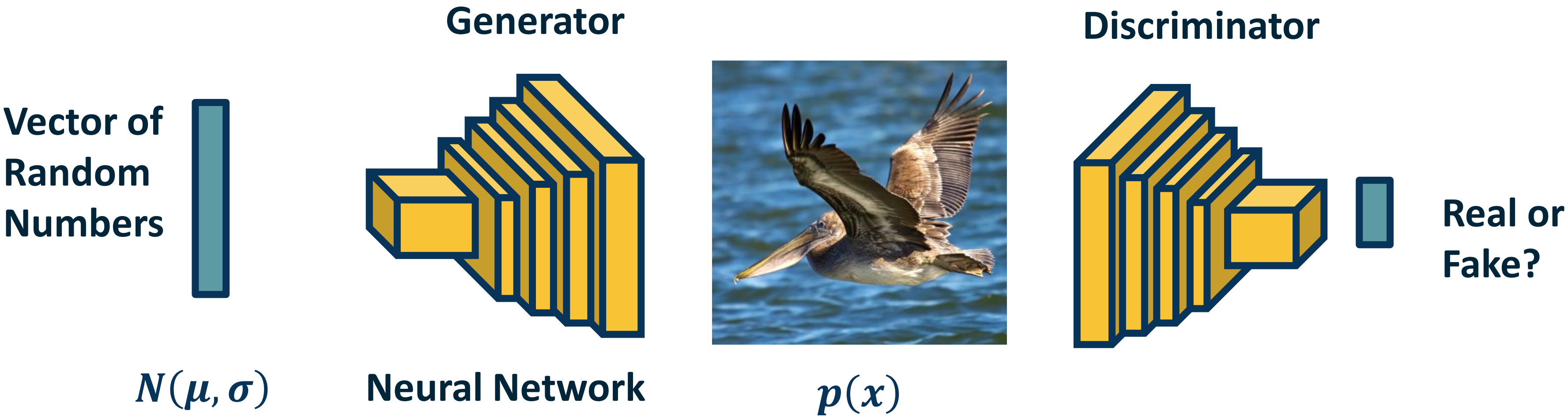
$N(\mu, \sigma)$

Neural Network

$p(x)$

Generating Images

- **Goal:** We would like to generate *realistic* images. How can we drive the network to learn how to do this?
- **Idea:** Have *another* network try to distinguish a real image from a generated (fake) image
    - **Why?** Signal can be used to determine how well it's doing at generation



Vector of Random Numbers

$N(\mu, \sigma)$

Generator

Neural Network

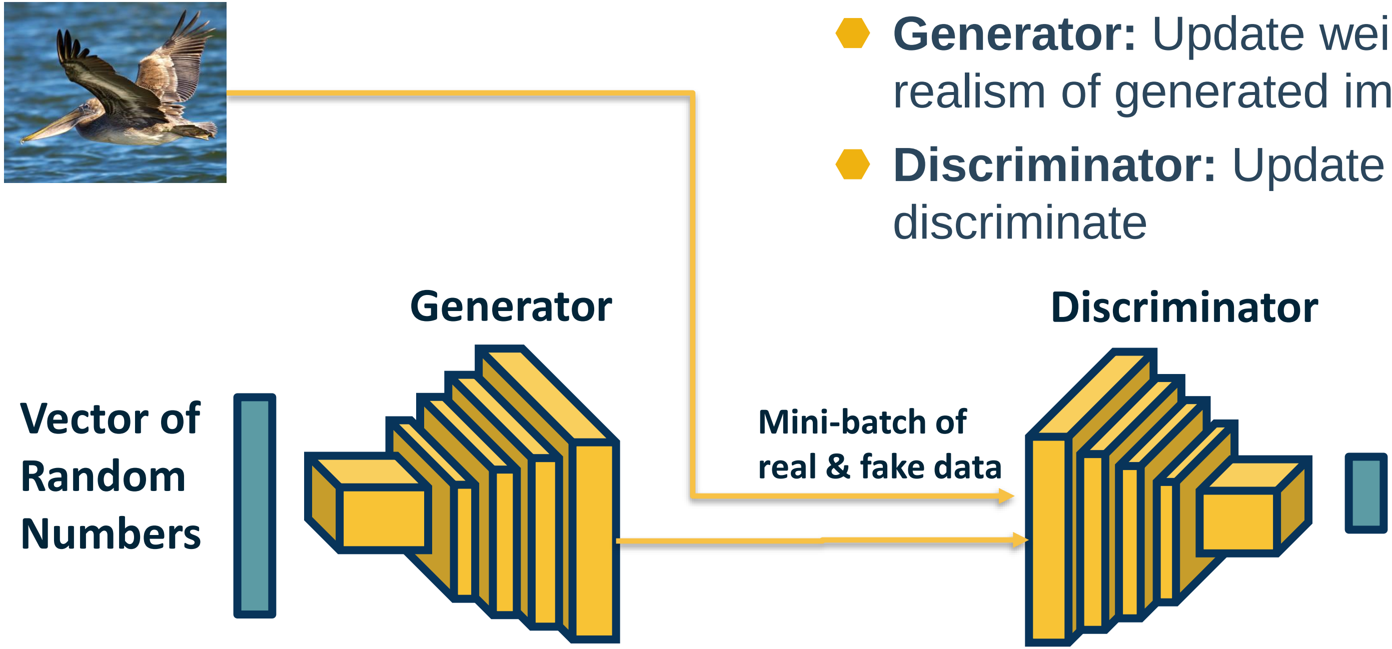
$p(x)$

Discriminator

Real or Fake?

Adversarial Networks

- **Generator:** Update weights to improve realism of generated images
- **Discriminator:** Update weights to better discriminate



Vector of Random Numbers

Generator

Mini-batch of real & fake data

Discriminator

Cross-entropy (Real or Fake?)

We know the answer (self-supervised)

Question: What loss functions can we use (for each network)?

- Since we have two networks competing, this is a mini-max two player game
  - Ties to game theory
  - Not clear what (even local) Nash equilibria are for this game

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Mini-max Two Player Game

- Since we have two networks competing, this is a mini-max two player game
  - Ties to game theory
  - Not clear what (even local) Nash equilibria are for this game

- The full mini-max objective is:

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

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Mini-max Two Player Game

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

  - where $D(x)$ is the discriminator outputs probability ([0,1]) of **real** image
  - x is a **real image** and $G(z)$ is a **generated** image

- The discriminator wants to **maximize** this:
  - $D(x)$ is pushed up (to 1) because x is a real image
  - $1 - D\big(G(z)\big)$ is also pushed up to 1 (so that $\mathrm{D}\big(\mathrm{G}(\mathrm{z})\big)$ is pushed down to 0)
  - In other words, discriminator wants to classify real images as real (1) and fake images as fake (0)

Discriminator Perspective

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

- where $D(x)$ is the discriminator outputs probability ([0,1]) of **real** image
- x is a **real image** and $G(z)$ is a **generated** image

- The generator wants to **minimize** this:
  - $1 - D\big(G(z)\big)$ is pushed down to 0 (so that $\mathrm{D}\big(\mathrm{G}(\mathrm{z})\big)$ is pushed up to 1)
  - This means that the generator is **fooling** the discriminator, i.e. succeeding at generating images that the discriminator can’t discriminate from real

Generator Perspective

- Since we have two networks competing, this is a mini-max two player game
  - Ties to game theory
  - Not clear what (even local) Nash equilibria are for this game

- The full mini-max objective is:

Sample from fake

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \underbrace{\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\textbf{How well discriminator does (0 for fake)}}$$

Generator *minimizes*

  - where $D(x)$ is the discriminator outputs probability ([0,1]) of **real** image
  - x is a **real image** and $G(z)$ is a **generated** image

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Mini-max Two Player Game

- Since we have two networks competing, this is a mini-max two player game
  - Ties to game theory
  - Not clear what (even local) Nash equilibria are for this game

- The full mini-max objective is:

Sample from real **Sample from fake**

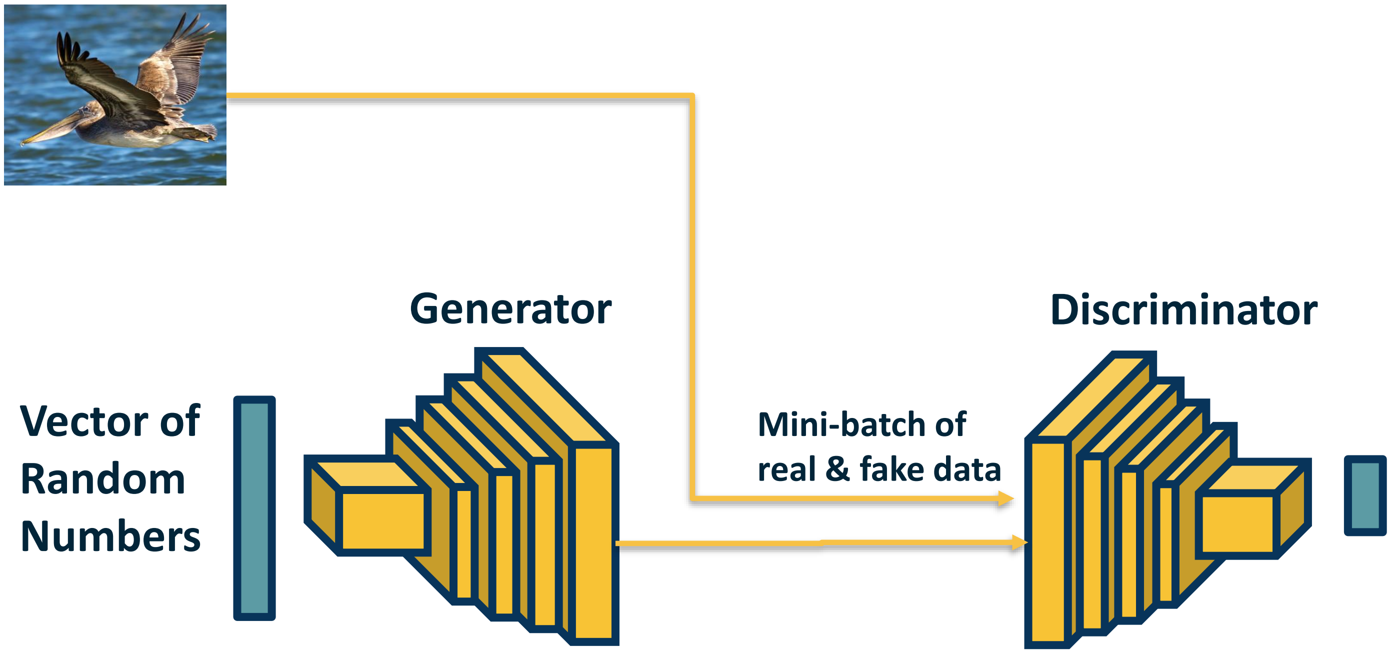
$$\min_{G}\max_{D} V(D,G) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{How well discriminator does (1 for real)}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]}_{\text{How well discriminator does (0 for fake)}}$$

Discriminator *maximizes*

  - where $D(x)$ is the discriminator outputs probability ([0,1]) of **real** image
  - x is a **real image** and $G(z)$ is a **generated** image

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Mini-max Two Player Game



Generator

Discriminator

Vector of Random Numbers

Mini-batch of real & fake data

Cross-entropy (Real or Fake?)

We know the answer (self-supervised)

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

Generator Loss

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

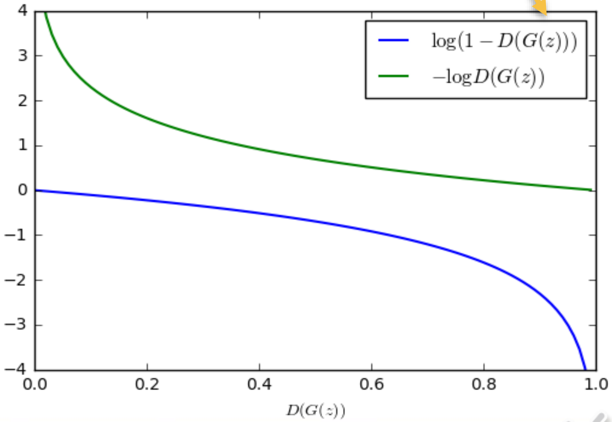
Discriminator Loss

Generative Adversarial Networks (GANs)

- The generator part of the objective does not have good gradient properties

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

  - High gradient when $D\big(G(z)\big)$ is high (that is, discriminator is wrong)
  - We want it to improve when samples are *bad* (discriminator is right)



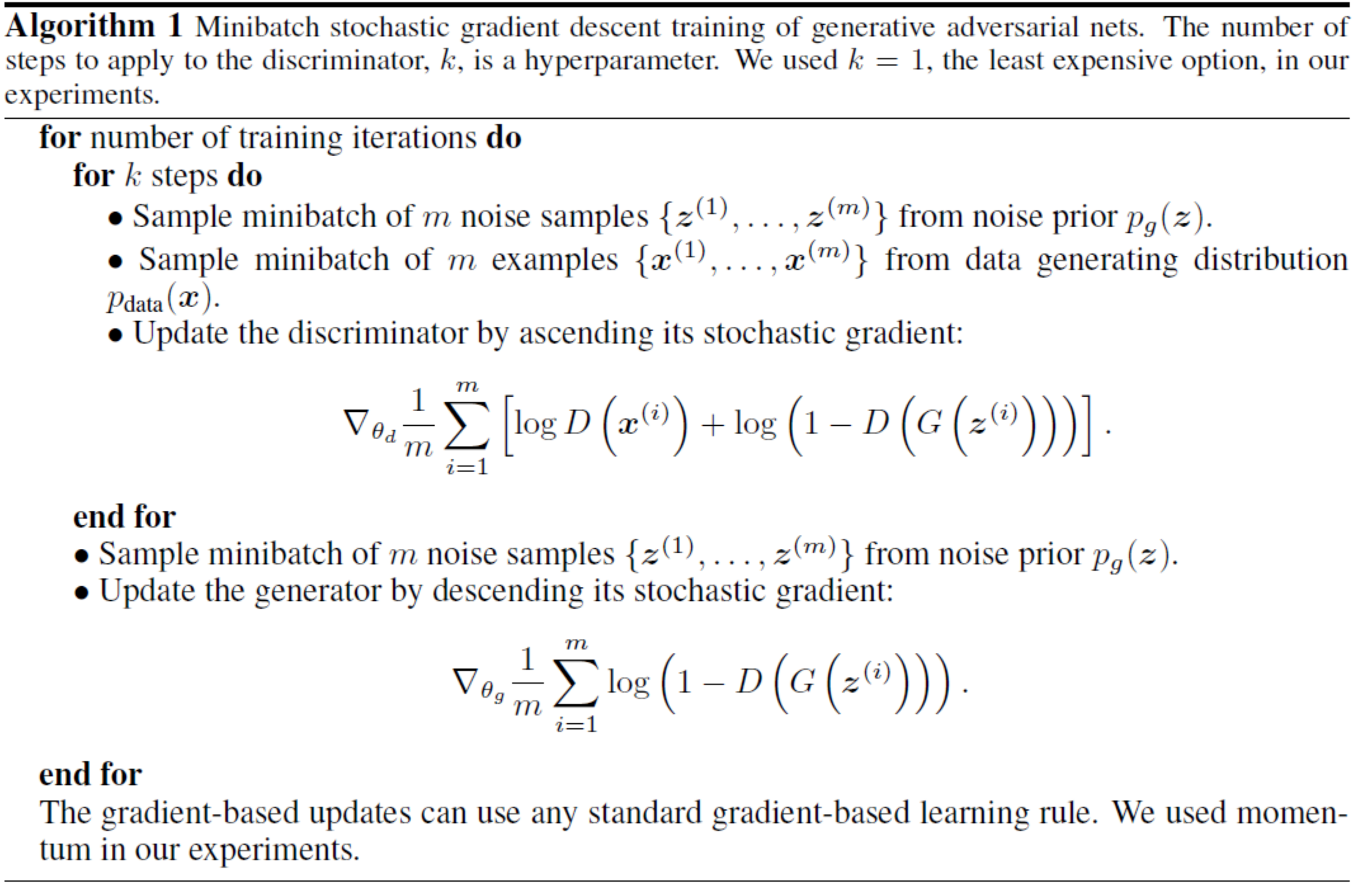
- Alternative objective, ***maximize***:

$$\max_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Plot from CS231n, Fei-Fei Li, Justin Johnson, Serena Yeung

Converting to Max-Max Game

Final Algorithm



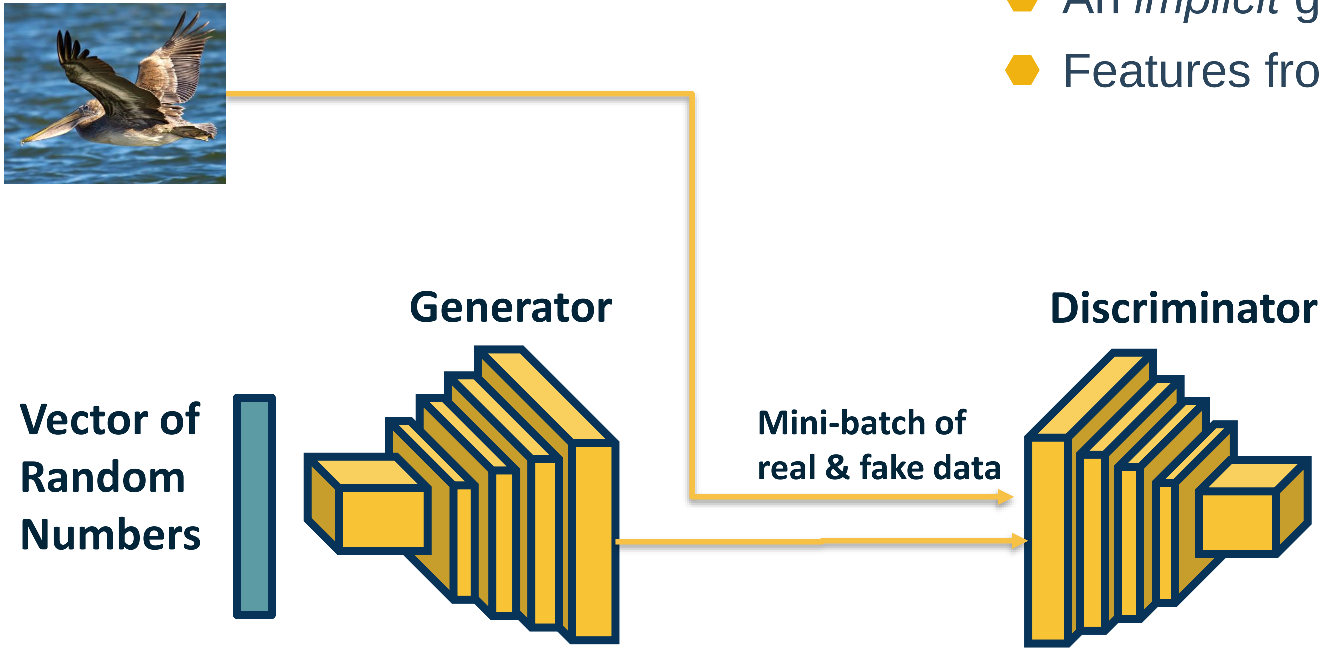
Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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.

Goodfellow, NeurIPS 2016 Generative Adversarial Nets

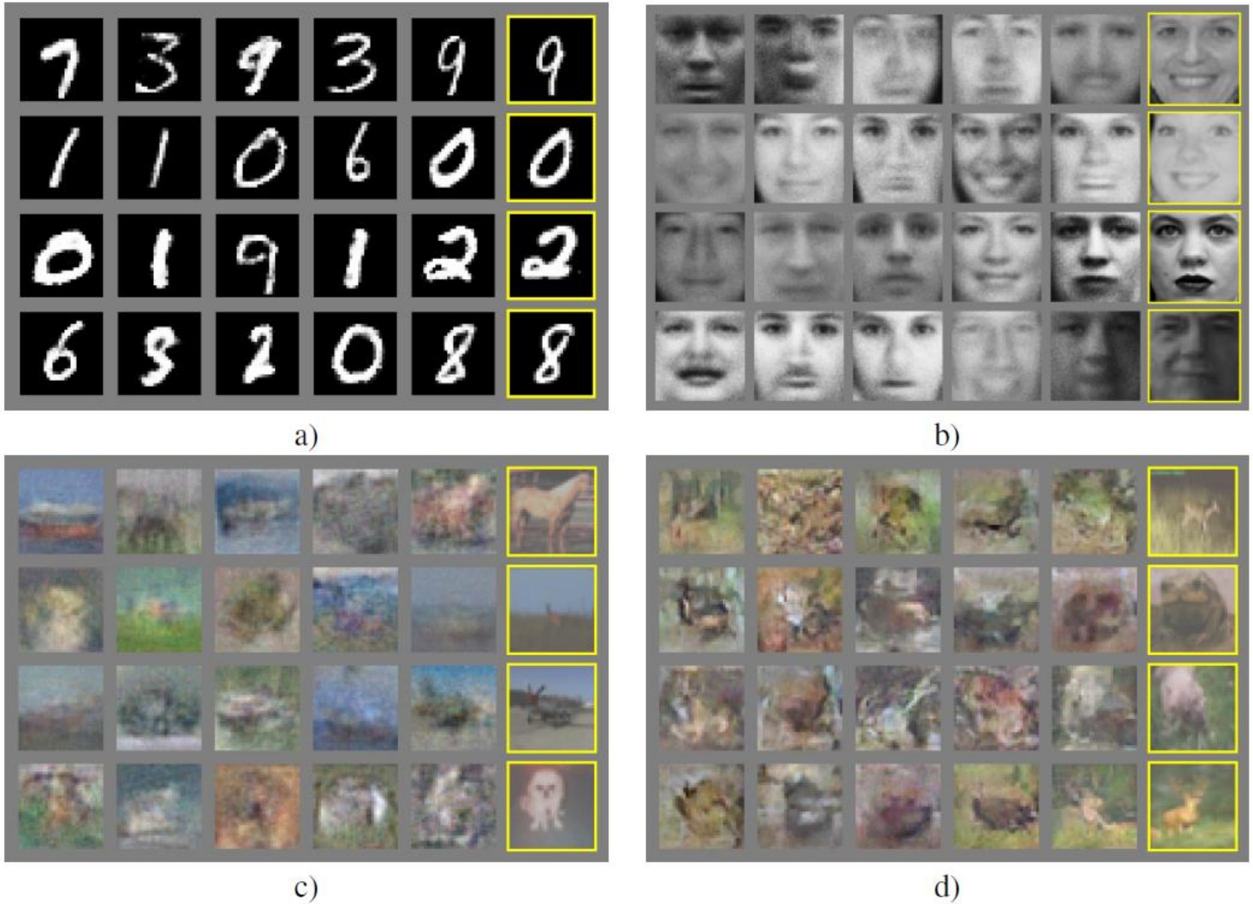
- At the end, we have:
  - An *implicit* generative model!
  - Features from discriminator



Cross-entropy (Real or Fake?)

We know the answer (self-supervised)

Early Results



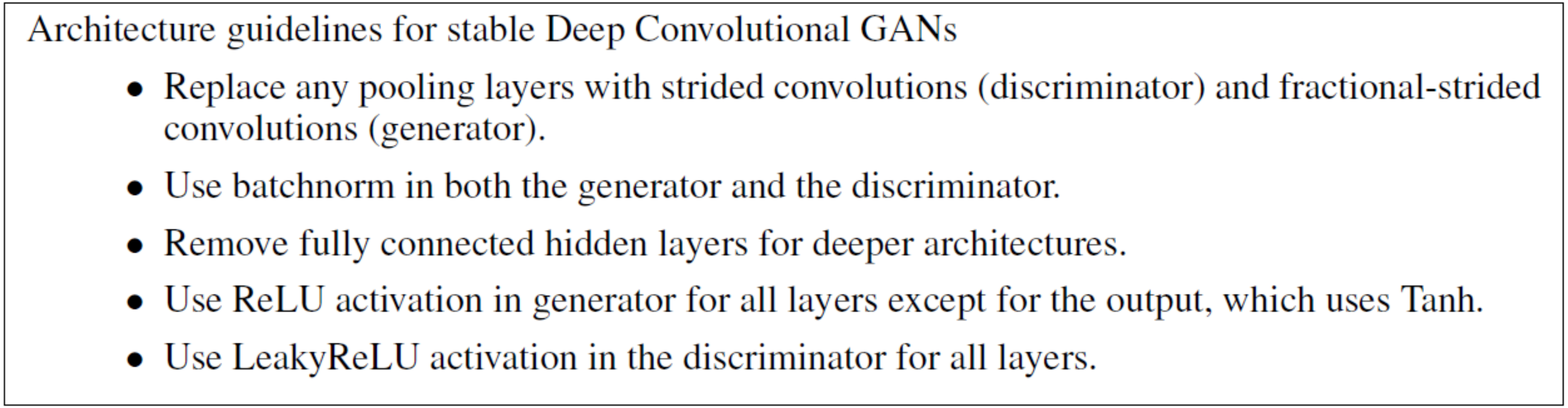
- Low-resolution images but look decent!
- Last column are nearest neighbor matches in dataset

- GANs are very difficult to train due to the mini-max objective

- Advancements include:
  - More stable architectures
  - Regularization methods to improve optimization
  - Progressive growing/training and scaling

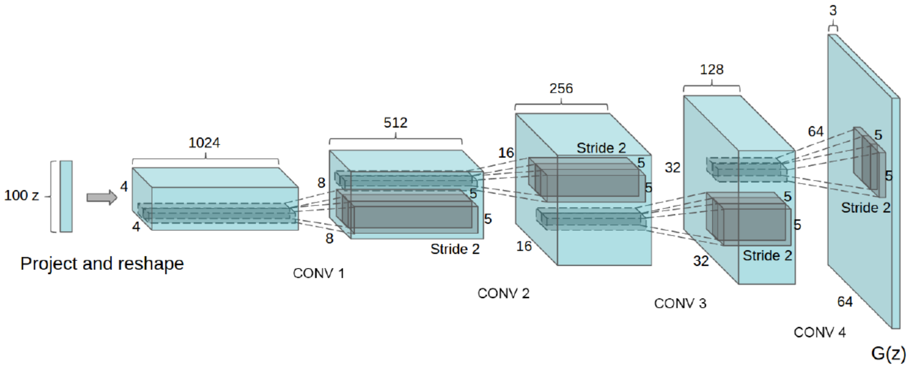
Goodfellow, NeurIPS 2016 Generative Adversarial Nets

Difficulty in Training



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.



Radford et al., Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

DCGAN

- Training GANs is difficult due to:
  - Minimax objective – For example, what if generator learns to memorize training data (no variety) or only generates part of the distribution?
  - Mode collapse – Capturing only some modes of distribution

- Several theoretically-motivated regularization methods
  - Simple example: Add noise to real samples!

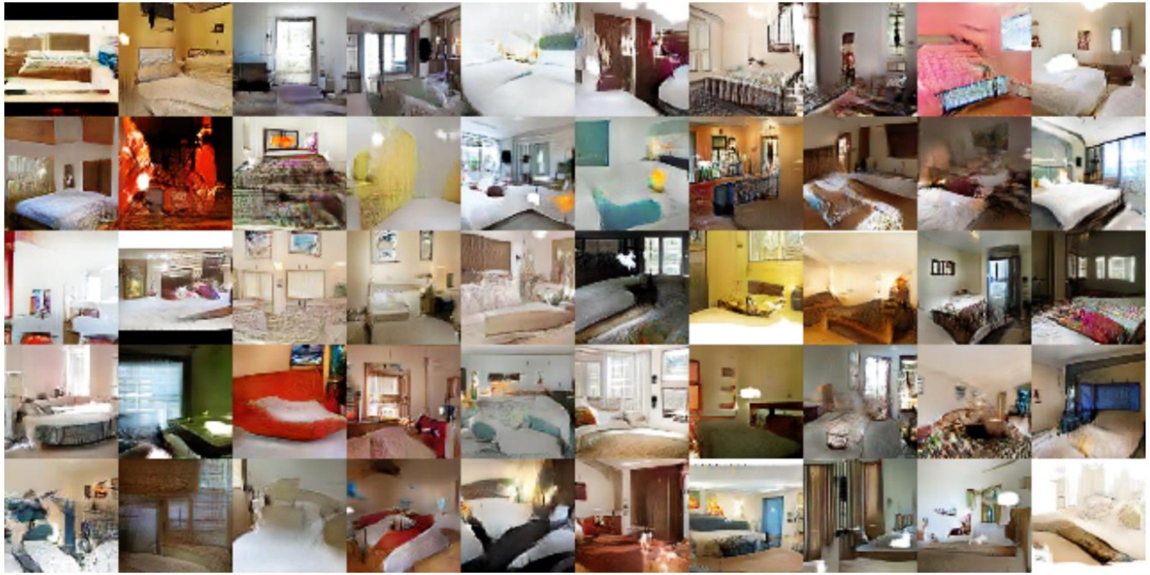
$$\lambda \cdot \mathbb{E}_{x \sim P_{real}, \delta \sim N_d(0, cI)} \left[\|\nabla_{\mathbf{x}} D_\theta(x + \delta)\| - k \right]^2$$

Kodali et al., On Convergence and Stability of GANs (also known as How to Train your DRAGAN)

Regularization

Generative Adversarial Nets: Convolutional Architectures

Samples from the model look much better!

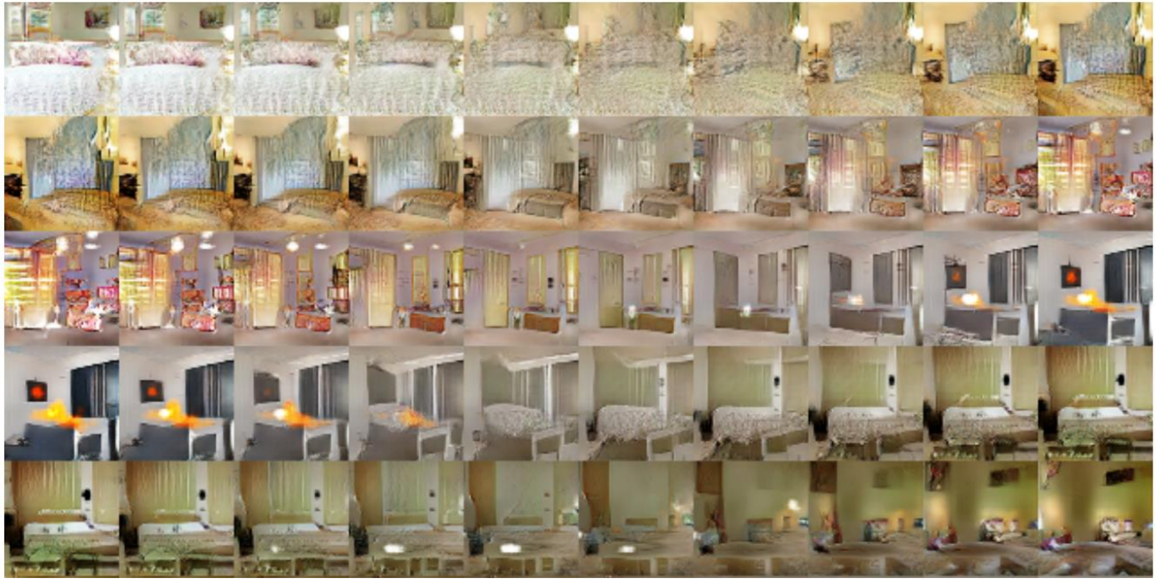


Radford et al, ICLR 2016

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Generative Adversarial Nets: Convolutional Architectures

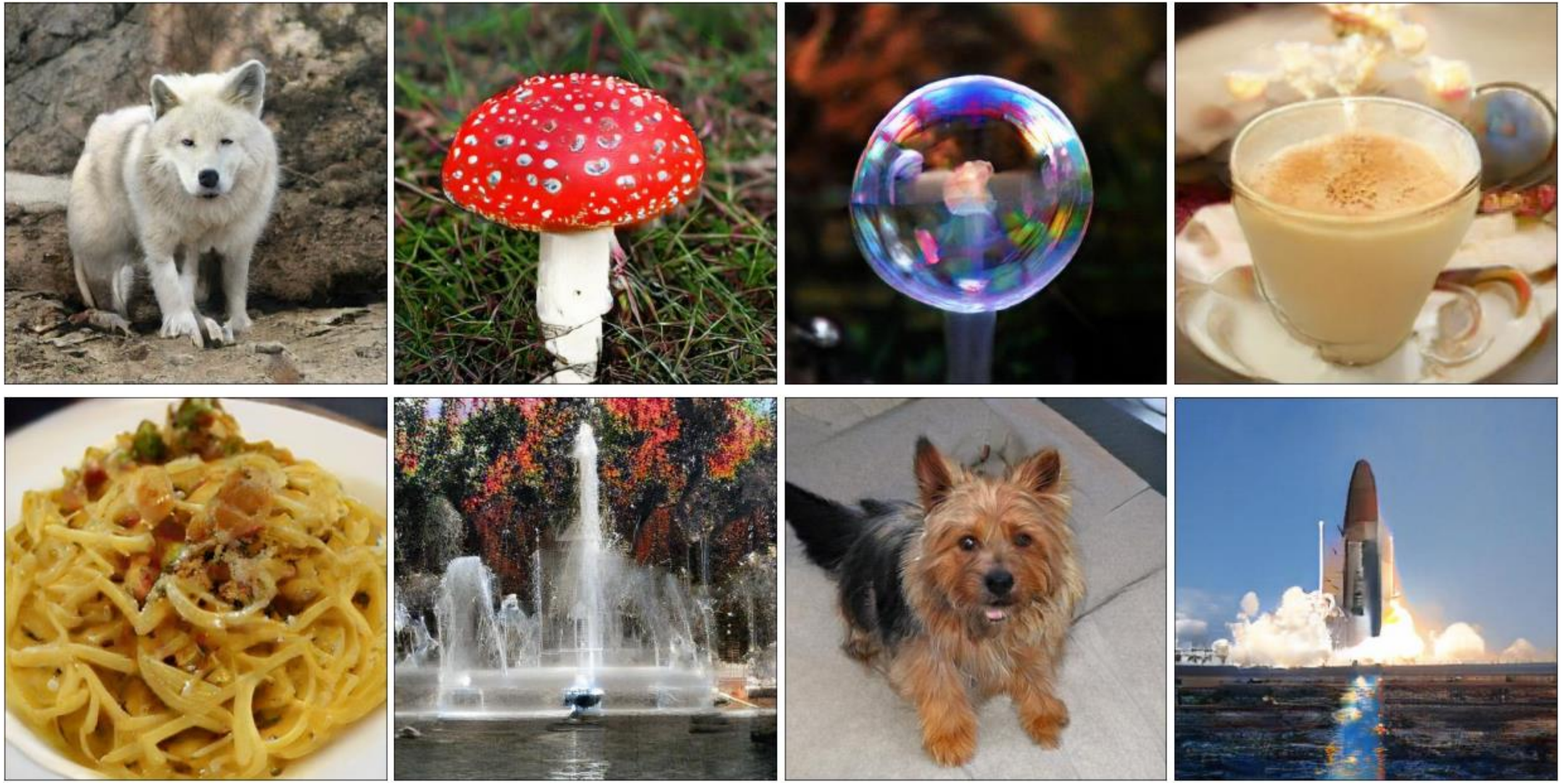
Interpolating between random points in latent space



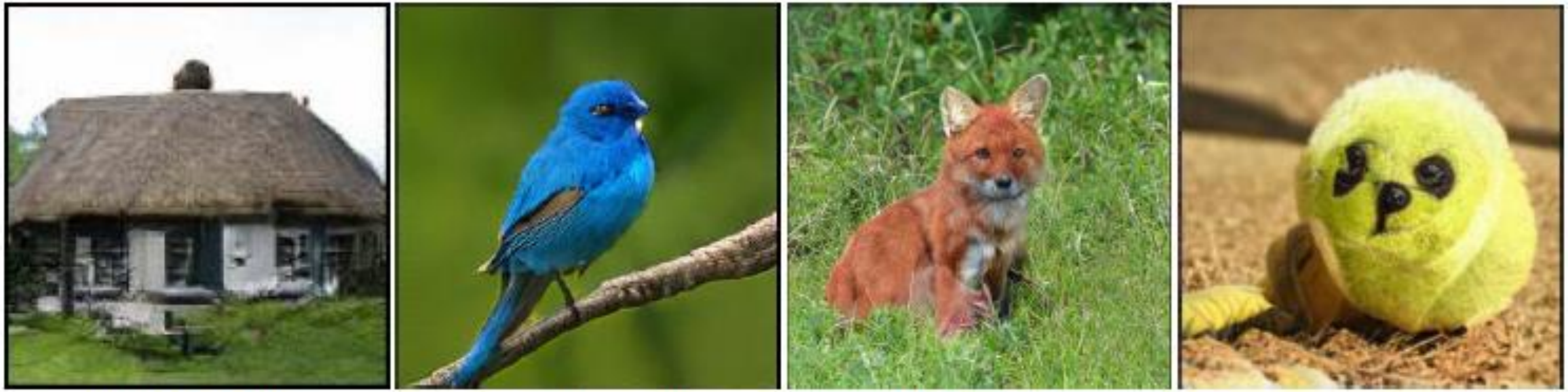
Radford et al,
ICLR 2016

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Example Generated Images - BigGAN



Brock et al., Large Scale GAN Training for High Fidelity Natural Image Synthesis

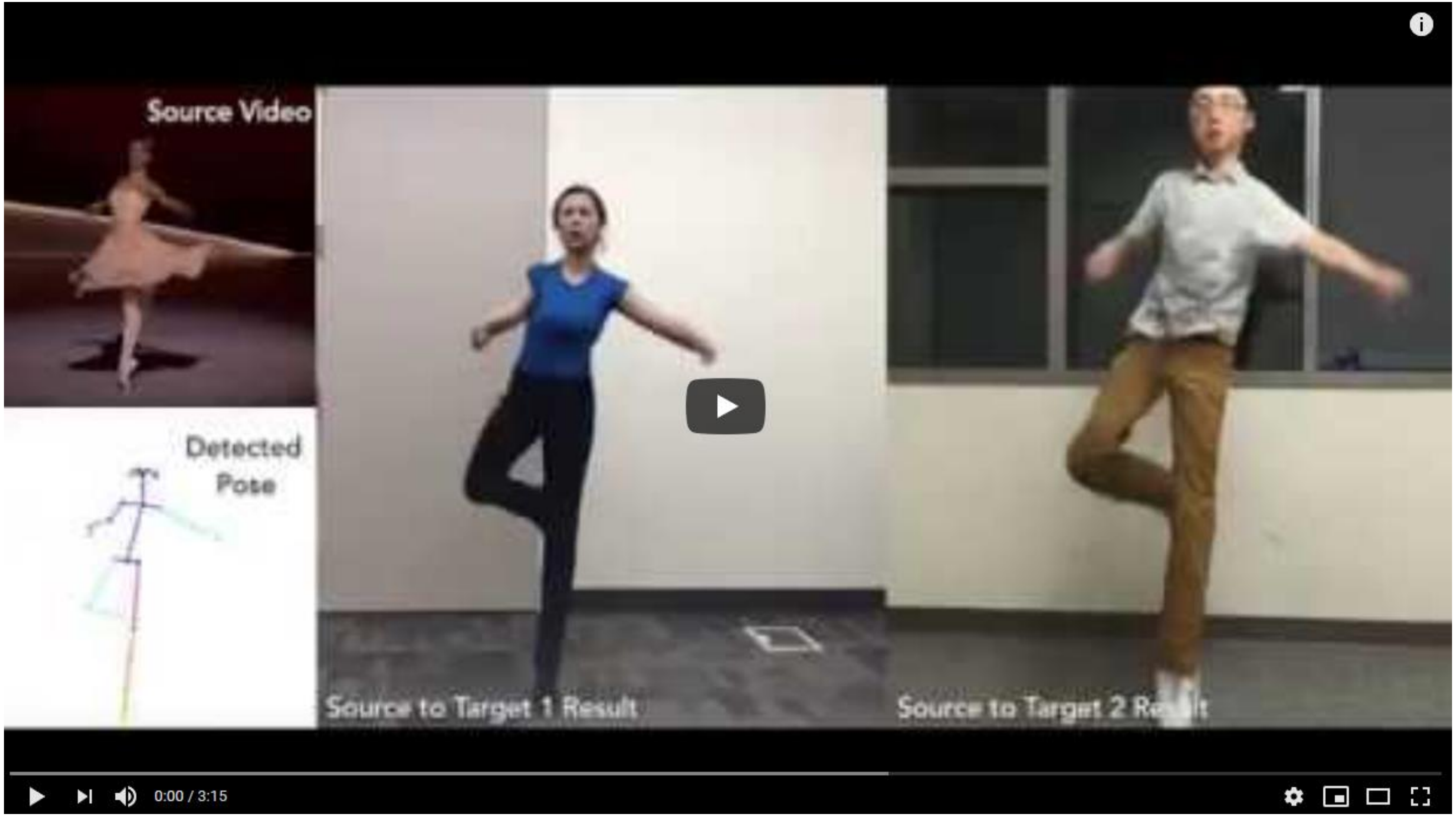


(a) 128×128 (b) 256×256 (c) 512×512 (d)

Figure 4: Samples from our model with truncation threshold 0.5 (a-c) and an example of class leakage in a partially trained model (d).

Brock et al., Large Scale GAN Training for High Fidelity Natural Image Synthesis

Failure Examples - BigGAN



https://www.youtube.com/watch?v=PCBTZh41Ris

Video Generation

Summary

- Generative Adversarial Networks (GANs) can produce amazing images!

- Several drawbacks
  - High-fidelity generation heavy to train
  - Training can be unstable
  - No explicit model for distribution

- Larger number of extensions:
  - GANs conditioned on labels or other information
  - Adversarial losses for other applications

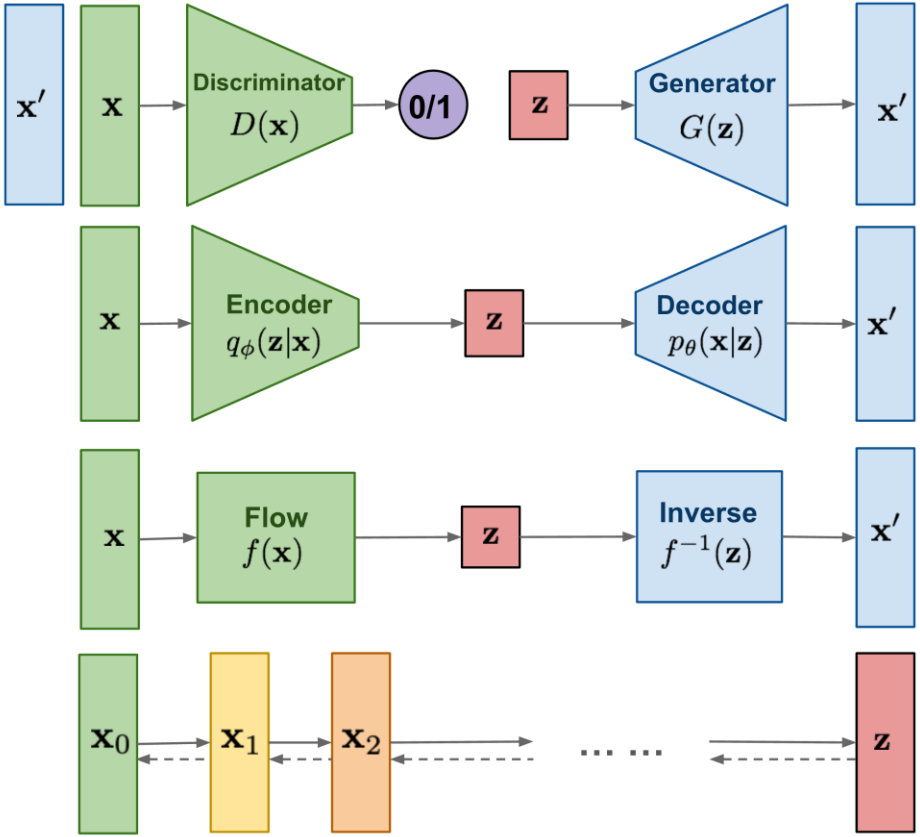
Comparison of Methods

GAN: Adversarial training

VAE: maximize variational lower bound

Flow-based models: Invertible transform of distributions

Diffusion models: Gradually add Gaussian noise and then reverse



$\mathbf{x}'$
$\mathbf{x}$
Discriminator
$D(\mathbf{x})$
0/1
$\mathbf{z}$
Generator
$G(\mathbf{z})$
$\mathbf{x}'$
$\mathbf{x}$
Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{z}$
Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
$\mathbf{x}$
Flow
$f(\mathbf{x})$
$\mathbf{z}$
Inverse
$f^{-1}(\mathbf{z})$
$\mathbf{x}'$
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
... ...
$\mathbf{z}$